In [ ]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (10, 8)
%matplotlib inline
```

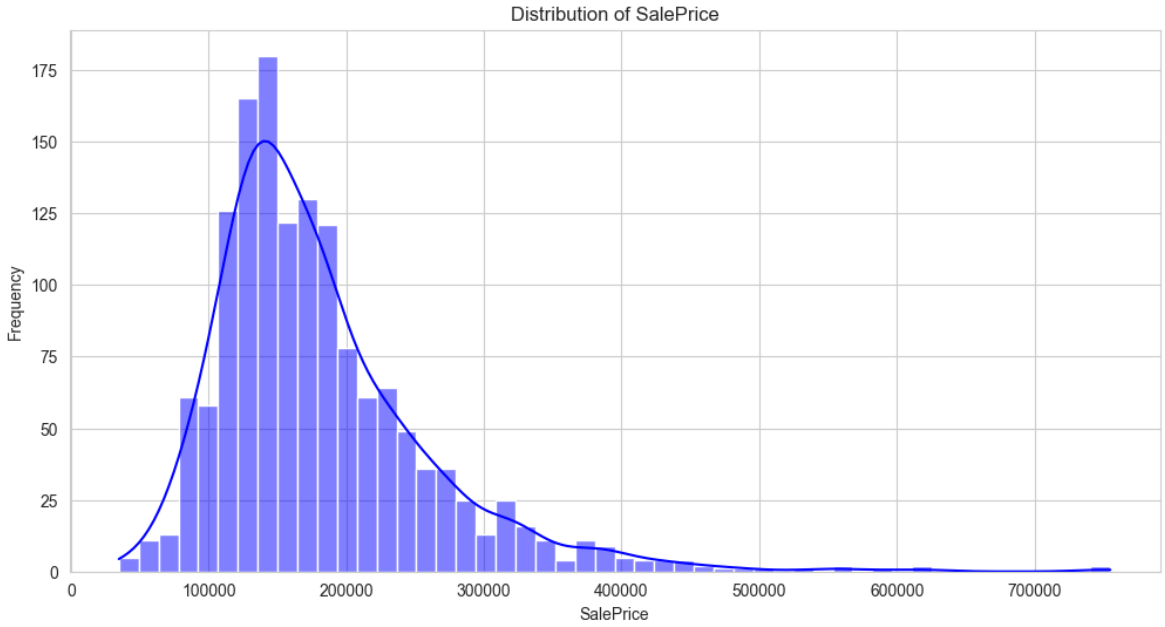
In [ ]:
```
# Loading the train dataset
train_data = pd.read_csv('train.csv')

# Displaying the first few rows of the dataset for an initial overview
train_data.head()
```

Out[ ]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCor
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	

5 rows × 81 columns

In [ ]:
```
# Loading the training dataset
train = pd.read_csv('train.csv')

# Visualizing the distribution of SalePrice
plt.figure(figsize=(12, 6))
sns.histplot(train['SalePrice'], bins=50, kde=True, color='blue')
plt.title('Distribution of SalePrice')
plt.xlabel('SalePrice')
plt.ylabel('Frequency')
plt.show()

# Computing basic statistics
saleprice_stats = train['SalePrice'].describe()
saleprice_stats
```

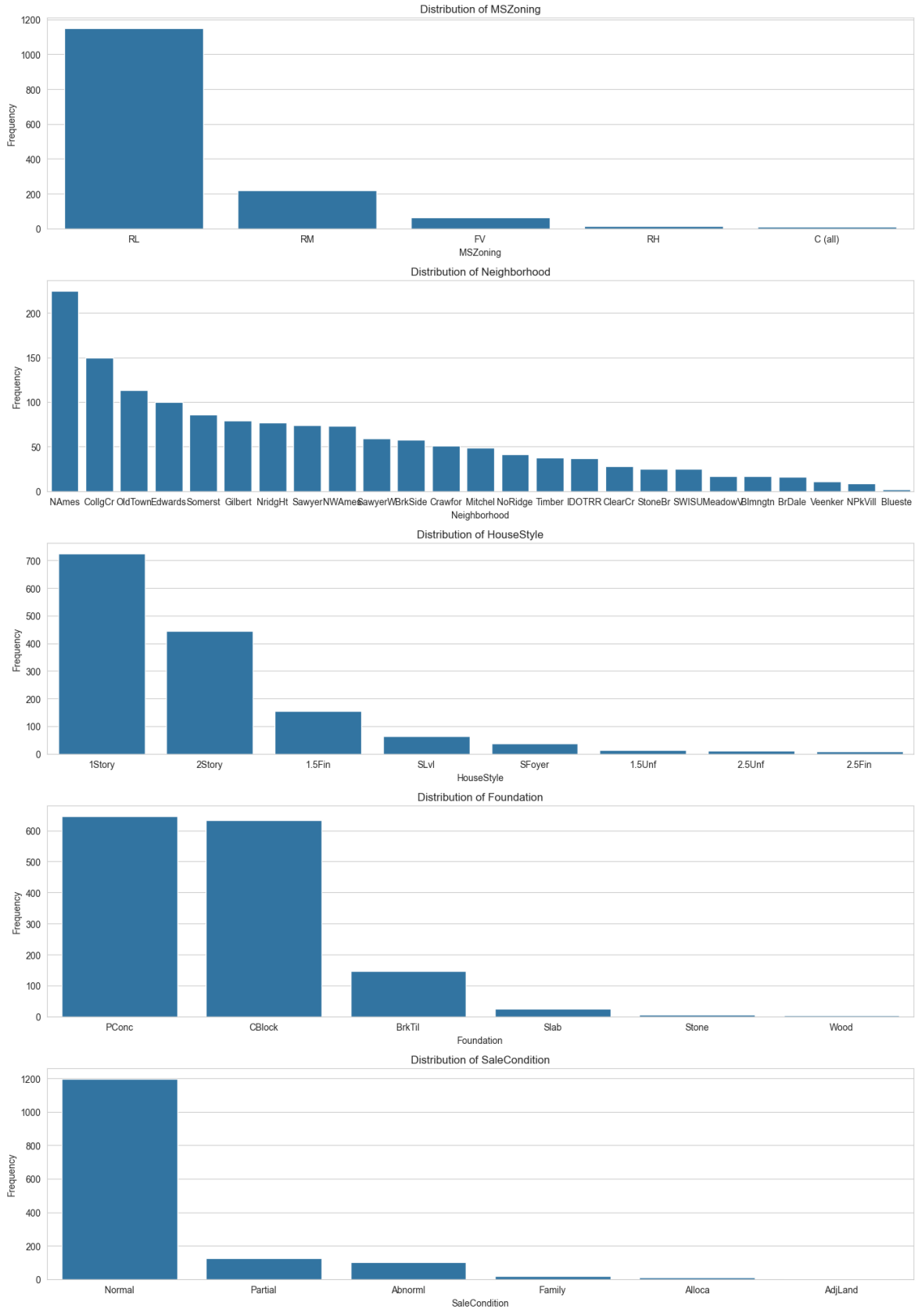


Out[ ]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [ ]:
```
# List of categorical features
categorical_features = train_data.select_dtypes(include=['object']).columns.toli

categorical_features
```

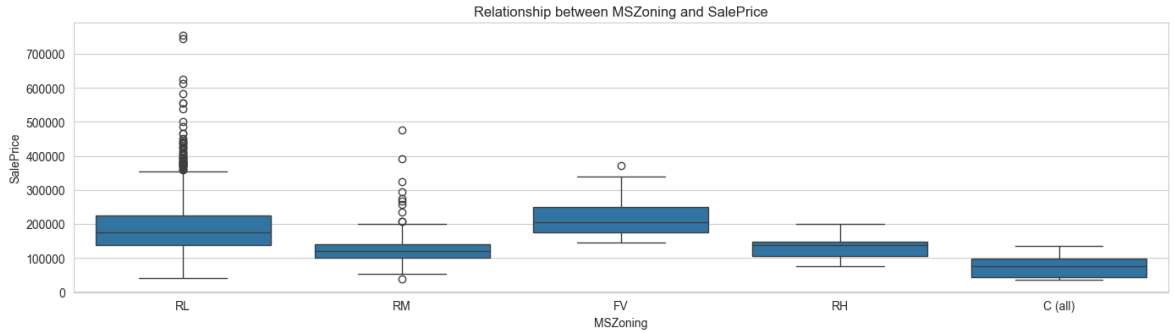
Out[ ]:
```
['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

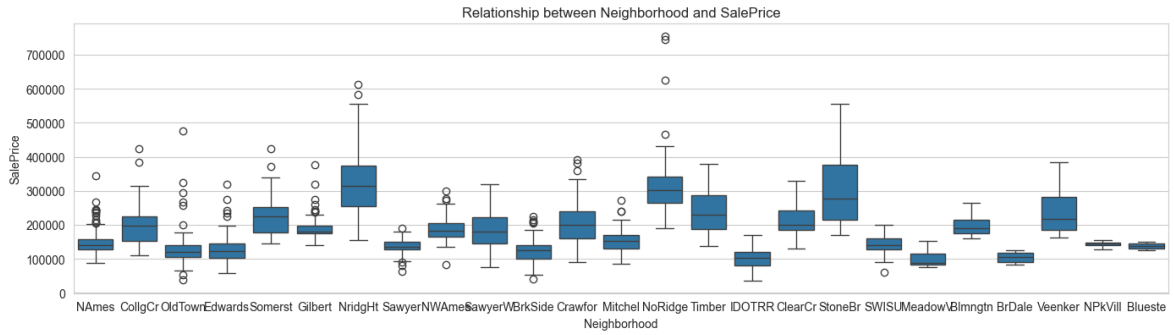
In [ ]:
```
# Selected categorical features
selected_features = ['MSZoning', 'Neighborhood', 'HouseStyle', 'Foundation', 'Sa

# Plotting the distribution of selected categorical features
fig, axes = plt.subplots(nrows=len(selected_features), figsize=(14, 20))

for i, feature in enumerate(selected_features):
    sns.countplot(data=train_data, x=feature, ax=axes[i], order=train_data[featu
    axes[i].set_title(f'Distribution of {feature}')
    axes[i].set_ylabel('Frequency')
    plt.tight_layout()

plt.show()
```

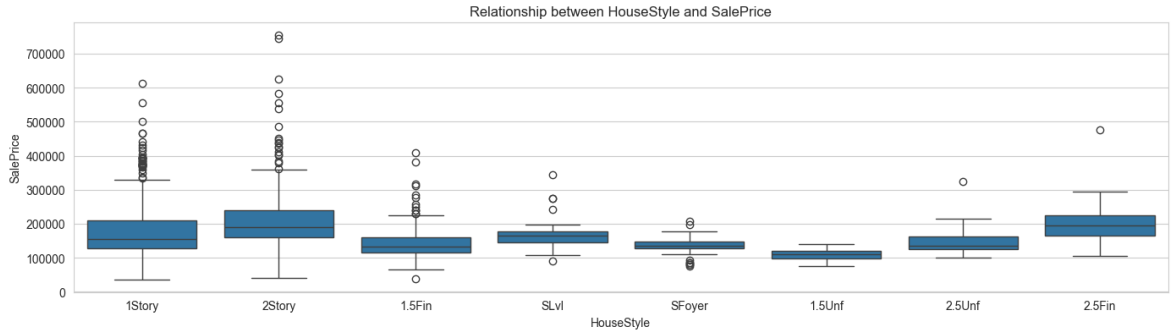


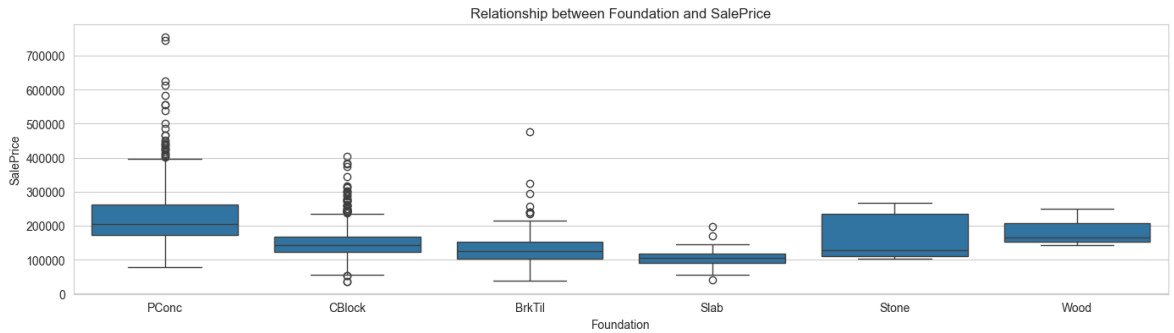
In [ ]:
```
# Plotting the relationship between selected categorical features and SalePrice
fig, axes = plt.subplots(nrows=len(selected_features), figsize=(14, 20))

for i, feature in enumerate(selected_features):
    sns.boxplot(data=train_data, x=feature, y='SalePrice', ax=axes[i], order=tra
    axes[i].set_title(f'Relationship between {feature} and SalePrice')
    axes[i].set_ylabel('SalePrice')
    plt.tight_layout()
```

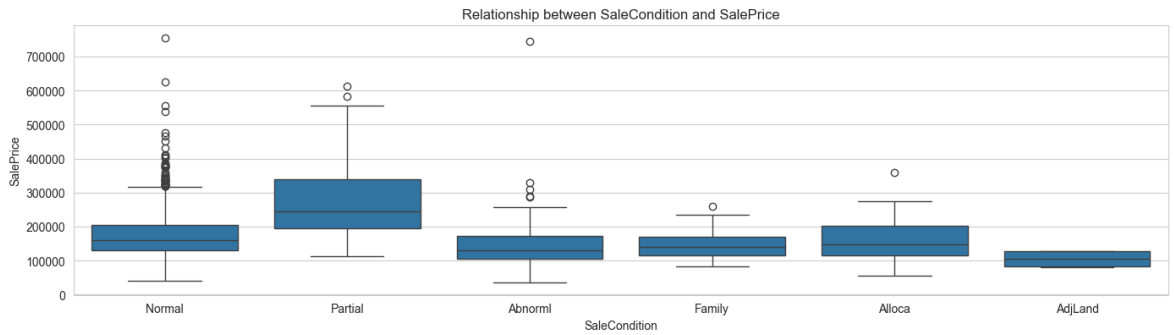
```
plt.show()
```





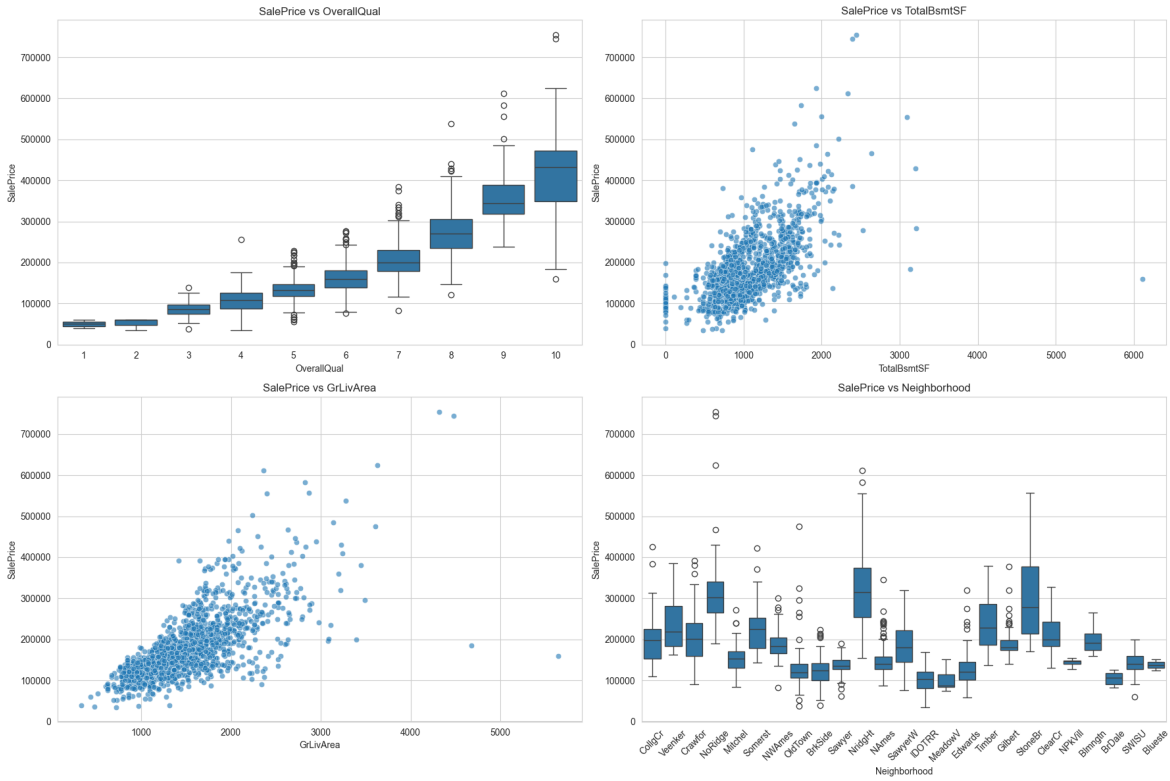






In [ ]:
```
# Visualization of relationships between selected features and SalePrice

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(18, 12))

# Relationship between OverallQual and SalePrice
sns.boxplot(data=train_data, x='OverallQual', y='SalePrice', ax=axes[0, 0])
axes[0, 0].set_title('SalePrice vs OverallQual')
```

```
# Relationship between TotalBsmtSF and SalePrice
sns.scatterplot(data=train_data, x='TotalBsmtSF', y='SalePrice', ax=axes[0, 1],
axes[0, 1].set_title('SalePrice vs TotalBsmtSF')

# Relationship between GrLivArea and SalePrice
sns.scatterplot(data=train_data, x='GrLivArea', y='SalePrice', ax=axes[1, 0], al
axes[1, 0].set_title('SalePrice vs GrLivArea')

# Relationship between Neighborhood and SalePrice
sns.boxplot(data=train_data, x='Neighborhood', y='SalePrice', ax=axes[1, 1])
axes[1, 1].set_title('SalePrice vs Neighborhood')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



In [ ]:
```
# Identifying missing values in the dataset
missing_data = train.isnull().sum().sort_values(ascending=False)

# Filtering out columns that have missing values
missing_data = missing_data[missing_data > 0]

# Calculating the percentage of missing values for each column
missing_percentage = (train[missing_data.index].isnull().mean() * 100).sort_valu

# Creating a DataFrame to display missing data count and percentage
missing_df = pd.DataFrame({
    'Missing Values': missing_data,
    'Percentage': missing_percentage
})

missing_df
```

Out[ ]:

	Missing Values	Percentage
Alley	1369	93.767123
BsmtCond	37	2.534247
BsmtExposure	38	2.602740
BsmtFinType1	37	2.534247
BsmtFinType2	38	2.602740
BsmtQual	37	2.534247
Electrical	1	0.068493
Fence	1179	80.753425
FireplaceQu	690	47.260274
GarageCond	81	5.547945
GarageFinish	81	5.547945
GarageQual	81	5.547945
GarageType	81	5.547945
GarageYrBlt	81	5.547945
LotFrontage	259	17.739726
MasVnrArea	8	0.547945
MasVnrType	872	59.726027
MiscFeature	1406	96.301370
PoolQC	1453	99.520548

In [ ]:

```
# Re-defining the column lists for missing value strategies

no_feature_cols = ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu']
garage_cols = ['GarageCond', 'GarageQual', 'GarageFinish', 'GarageType']
basement_cols = ['BsmtFinType2', 'BsmtExposure', 'BsmtFinType1', 'BsmtCond', 'Bs

# Re-applying the missing value handling strategies

# 1. For 'PoolQC', 'MiscFeature', 'Alley', 'Fence', and 'FireplaceQu'
for col in no_feature_cols:
    train[col].fillna('None', inplace=True)

# 2. 'LotFrontage': Using median of the neighborhood
train['LotFrontage'] = train.groupby('Neighborhood')['LotFrontage'].transform(la

# 3. For 'GarageCond', 'GarageQual', 'GarageFinish', 'GarageType', and 'GarageYr
for col in garage_cols:
    train[col].fillna('None', inplace=True)
train['GarageYrBlt'].fillna(train['YearBuilt'], inplace=True)

# 4. For 'BsmtFinType2', 'BsmtExposure', 'BsmtFinType1', 'BsmtCond', and 'BsmtQu
for col in basement_cols:
    train[col].fillna('None', inplace=True)
```

```
# 5. 'MasVnrType' and 'MasVnrArea'
train['MasVnrType'].fillna('None', inplace=True)
train['MasVnrArea'].fillna(0, inplace=True)

# 6. 'Electrical'
train['Electrical'].fillna(train['Electrical'].mode()[0], inplace=True)

# Checking again for any remaining missing values
remaining_missing = train.isnull().sum().sort_values(ascending=False)
remaining_missing[remaining_missing > 0]
```

Out[ ]:
```
Series([], dtype: int64)
```

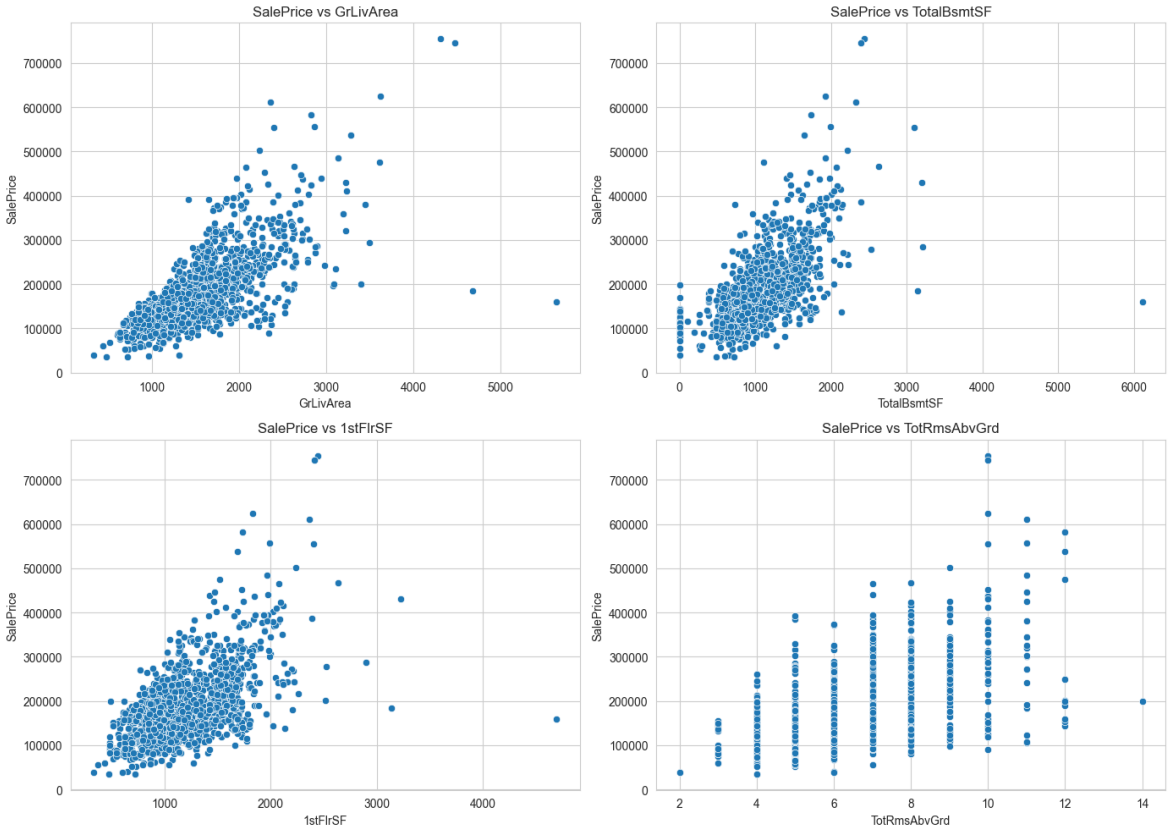
In [ ]:
```
# Encoding the ordinal features:

# Mapping the ordinal columns
ordinal_mappings = {
    'LotShape': {'Reg': 4, 'IR1': 3, 'IR2': 2, 'IR3': 1},
    'Utilities': {'AllPub': 4, 'NoSewr': 3, 'NoSeWa': 2, 'ELO': 1},
    'LandSlope': {'Gtl': 3, 'Mod': 2, 'Sev': 1},
    'ExterQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'ExterCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'BsmtQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'BsmtCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'BsmtExposure': {'Gd': 4, 'Av': 3, 'Mn': 2, 'No': 1, 'NA': 0},
    'BsmtFinType1': {'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1,
    'BsmtFinType2': {'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1,
    'HeatingQC': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'KitchenQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'Functional': {'Typ': 8, 'Min1': 7, 'Min2': 6, 'Mod': 5, 'Maj1': 4, 'Maj2':
    'FireplaceQu': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'GarageFinish': {'Fin': 3, 'RFn': 2, 'Unf': 1, 'NA': 0},
    'GarageQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'GarageCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'PavedDrive': {'Y': 3, 'P': 2, 'N': 1},
    'PoolQC': {'Ex': 4, 'Gd': 3, 'TA': 2, 'Fa': 1, 'NA': 0},
    'Fence': {'GdPrv': 4, 'MnPrv': 3, 'GdWo': 2, 'MnWw': 1, 'NA': 0}
}

# Applying the mappings to the train dataset
train.replace(ordinal_mappings, inplace=True)


nominal_features = [
    'MSZoning', 'Street', 'LandContour', 'LotConfig', 'Neighborhood',
    'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle',
    'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'Foundation',
    'Heating', 'CentralAir', 'Electrical', 'GarageType', 'SaleType', 'SaleCondit
]

# Encoding the nominal features using One-Hot Encoding
train_encoded = pd.get_dummies(train, columns=nominal_features, drop_first=True)

train_encoded.head()
```

Out[ ]:

	Id	MSSubClass	LotFrontage	LotArea	Alley	LotShape	Utilities	LandSlope	Overal
0	1	60	65.0	8450	None	4	4	3	
1	2	20	80.0	9600	None	4	4	3	
2	3	60	68.0	11250	None	3	4	3	
3	4	70	60.0	9550	None	3	4	3	
4	5	60	84.0	14260	None	3	4	3	

5 rows × 200 columns

In [ ]:
```
# Scatter plots for the selected features against SalePrice
features = ['GrLivArea', 'TotalBsmtSF', '1stFlrSF', 'TotRmsAbvGrd']

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 10))

for i, feature in enumerate(features):
    sns.scatterplot(data=train_data, x=feature, y='SalePrice', ax=axes[i//2, i%2
    axes[i//2, i%2].set_title(f'SalePrice vs {feature}')
    axes[i//2, i%2].set_xlabel(feature)
    axes[i//2, i%2].set_ylabel('SalePrice')

plt.tight_layout()
plt.show()
```

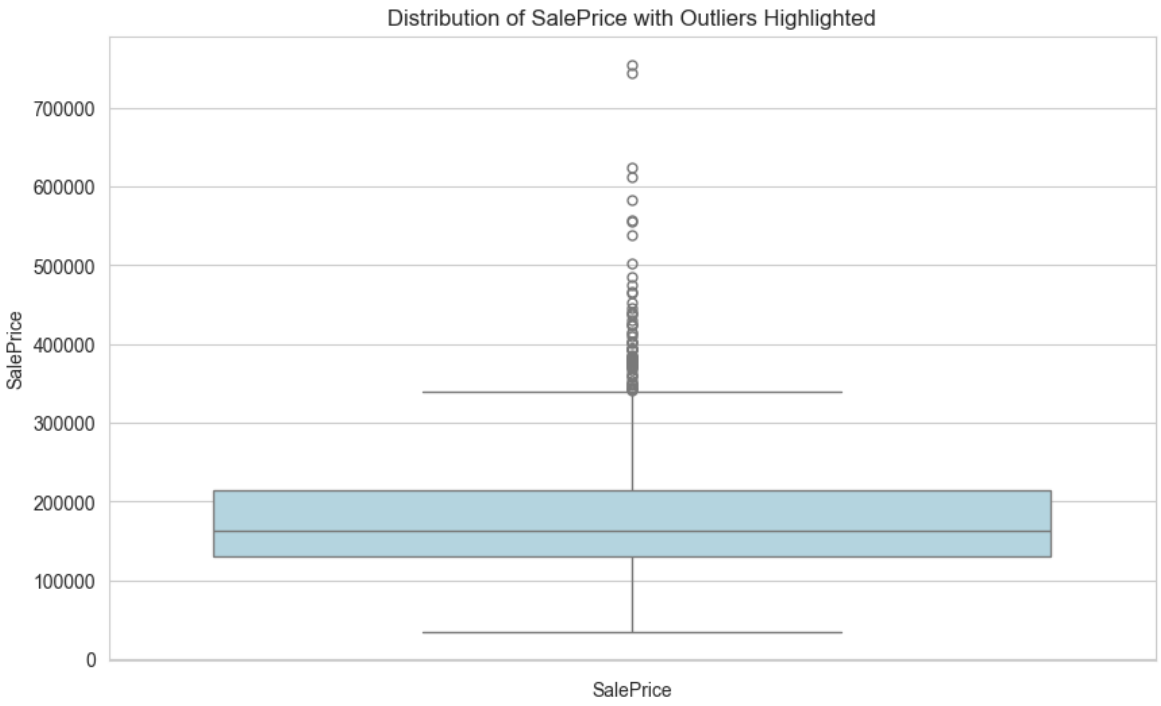


In [ ]:
```
# Calculate IQR for SalePrice
Q1 = train['SalePrice'].quantile(0.25)
Q3 = train['SalePrice'].quantile(0.75)
IQR = Q3 - Q1
```

```
# Define bounds for the outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers
outliers = train[(train['SalePrice'] < lower_bound) | (train['SalePrice'] > uppe

# Number of outliers detected
num_outliers = outliers.shape[0]

num_outliers
```

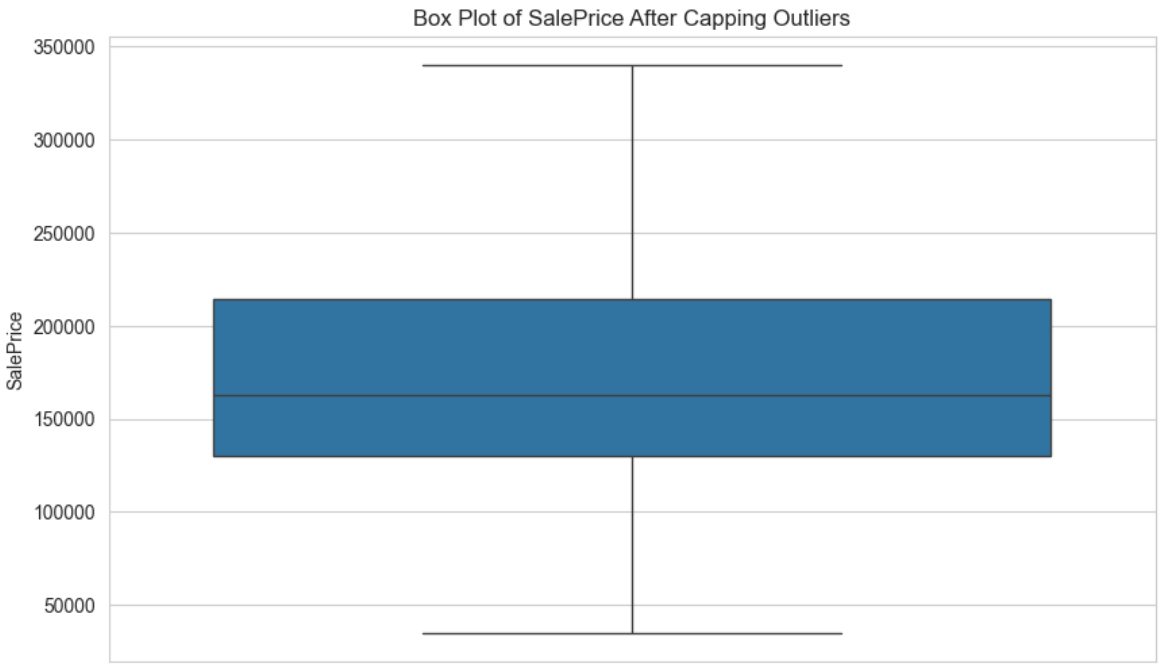
Out[ ]: 61

In [ ]:
```
# Plotting the SalePrice distribution with outliers highlighted

plt.figure(figsize=(10, 6))
sns.boxplot(train['SalePrice'], color="lightblue", fliersize=5)
plt.title('Distribution of SalePrice with Outliers Highlighted')
plt.xlabel('SalePrice')
plt.show()
```



In [ ]:
```
# Calculate the IQR for SalePrice
Q1 = train['SalePrice'].quantile(0.25)
Q3 = train['SalePrice'].quantile(0.75)
IQR = Q3 - Q1

# Determine the upper threshold
upper_threshold = Q3 + 1.5 * IQR

# Cap values above the upper threshold
train['SalePrice'] = np.where(train['SalePrice'] > upper_threshold, upper_thresh

# Check the modified distribution of SalePrice
plt.figure(figsize=(10, 6))
sns.boxplot(train['SalePrice'])
```

```
plt.title('Box Plot of SalePrice After Capping Outliers')
plt.show()
```

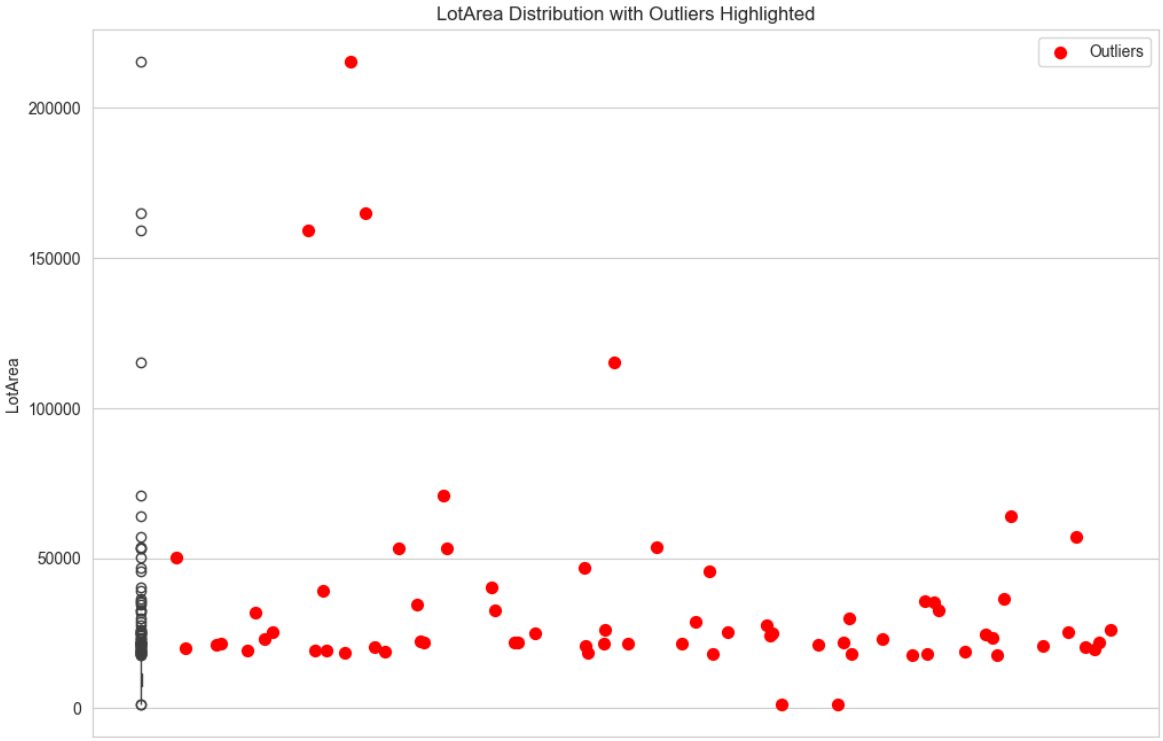


In [ ]:
```
# Identify outliers for all numerical columns using IQR method

outliers_dict = {}  # Dictionary to store number of outliers for each feature

# Iterate over numerical features
for col in train.select_dtypes(include=['float64', 'int64']).columns:
    # Skip the target variable
    if col == 'SalePrice':
        continue

    # Calculate IQR
    Q1 = train[col].quantile(0.25)
    Q3 = train[col].quantile(0.75)
    IQR = Q3 - Q1

    # Define bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Find outliers
    outliers = train[(train[col] < lower_bound) | (train[col] > upper_bound)]
    outliers_dict[col] = len(outliers)

outliers_dict
```

Out[ ]:
```
{'Id': 0,
 'MSSubClass': 103,
 'LotFrontage': 93,
 'LotArea': 69,
 'LotShape': 10,
 'Utilities': 1,
 'LandSlope': 78,
 'OverallQual': 2,
 'OverallCond': 125,
 'YearBuilt': 7,
 'YearRemodAdd': 0,
 'MasVnrArea': 98,
 'ExterQual': 0,
 'ExterCond': 178,
 'BsmtFinSF1': 7,
 'BsmtFinSF2': 167,
 'BsmtUnfSF': 29,
 'TotalBsmtSF': 61,
 'HeatingQC': 0,
 '1stFlrSF': 20,
 '2ndFlrSF': 2,
 'LowQualFinSF': 26,
 'GrLivArea': 31,
 'BsmtFullBath': 1,
 'BsmtHalfBath': 82,
 'FullBath': 0,
 'HalfBath': 0,
 'BedroomAbvGr': 35,
 'KitchenAbvGr': 68,
 'KitchenQual': 0,
 'TotRmsAbvGrd': 30,
 'Functional': 100,
 'Fireplaces': 5,
 'GarageYrBlt': 2,
 'GarageCars': 5,
 'GarageArea': 21,
 'PavedDrive': 120,
 'WoodDeckSF': 32,
 'OpenPorchSF': 77,
 'EnclosedPorch': 208,
 '3SsnPorch': 24,
 'ScreenPorch': 116,
 'PoolArea': 7,
 'MiscVal': 52,
 'MoSold': 0,
 'YrSold': 0}
```

In [ ]:
```
# Calculate the IQR for LotArea
Q1 = train['LotArea'].quantile(0.25)
Q3 = train['LotArea'].quantile(0.75)
IQR = Q3 - Q1

# Calculate boundaries
lower_boundary = Q1 - 1.5 * IQR
upper_boundary = Q3 + 1.5 * IQR

# Identify the outliers
outliers_lot_area = train[(train['LotArea'] < lower_boundary) | (train['LotArea'

# Plot the distribution with outliers highlighted
```

```
plt.figure(figsize=(12, 8))
sns.boxplot(train['LotArea'])
plt.scatter(outliers_lot_area.index, outliers_lot_area['LotArea'], color='red',
plt.title('LotArea Distribution with Outliers Highlighted')
plt.legend()
plt.show()

outliers_lot_area.shape[0], lower_boundary, upper_boundary
```



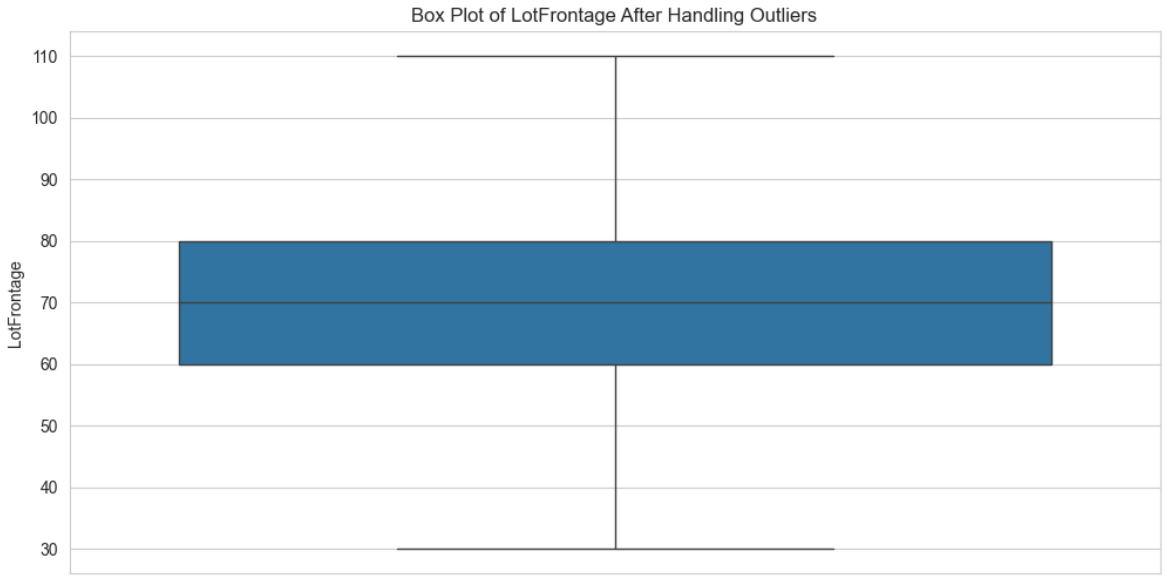
Out[ ]: (69, 1481.5, 17673.5)

In [ ]:
```
# Capping the outliers for 'LotArea' feature correctly
train['LotArea'] = np.where(train['LotArea'] < lower_bound, lower_bound,
                            np.where(train['LotArea'] > upper_bound, upper_bound

# Re-check if outliers have been capped
outliers_after_corrected = ((train['LotArea'] < lower_bound) | (train['LotArea']

outliers_after_corrected
```

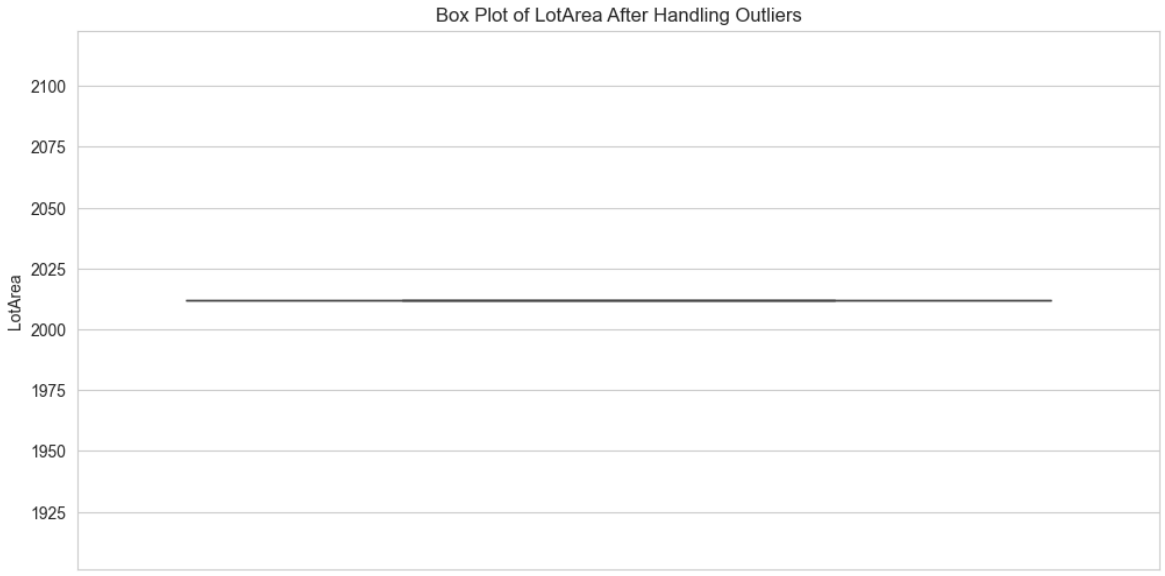
Out[ ]: 0

In [ ]:
```
# Calculating IQR for LotFrontage
Q1 = train['LotFrontage'].quantile(0.25)
Q3 = train['LotFrontage'].quantile(0.75)
IQR = Q3 - Q1

# Defining bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Capping outliers
train['LotFrontage'] = np.where(train['LotFrontage'] < lower_bound, lower_bound,
                                np.where(train['LotFrontage'] > upper_bound, upp

# Visualizing after capping
plt.figure(figsize=(12, 6))
```

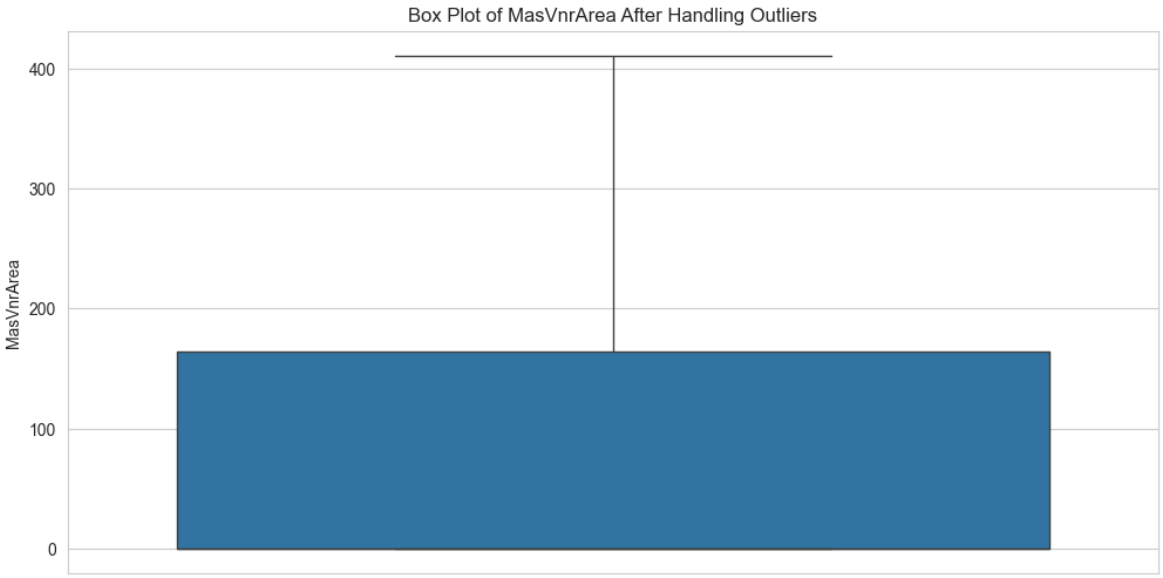
```
sns.boxplot(train['LotFrontage'])
plt.title('Box Plot of LotFrontage After Handling Outliers')
plt.show()
```



In [ ]:
```
# Calculating IQR for LotArea
Q1_lotarea = train['LotArea'].quantile(0.25)
Q3_lotarea = train['LotArea'].quantile(0.75)
IQR_lotarea = Q3_lotarea - Q1_lotarea

# Defining bounds
lower_bound_lotarea = Q1_lotarea - 1.5 * IQR_lotarea
upper_bound_lotarea = Q3_lotarea + 1.5 * IQR_lotarea

# Capping outliers
train['LotArea'] = np.where(train['LotArea'] < lower_bound_lotarea, lower_bound_
                            np.where(train['LotArea'] > upper_bound_lotarea, upp

# Visualizing after capping
plt.figure(figsize=(12, 6))
sns.boxplot(train['LotArea'])
plt.title('Box Plot of LotArea After Handling Outliers')
plt.show()
```



In [ ]:
```
# Calculating IQR for MasVnrArea
Q1_masvnr = train['MasVnrArea'].quantile(0.25)
Q3_masvnr = train['MasVnrArea'].quantile(0.75)
IQR_masvnr = Q3_masvnr - Q1_masvnr

# Defining bounds
lower_bound_masvnr = Q1_masvnr - 1.5 * IQR_masvnr
upper_bound_masvnr = Q3_masvnr + 1.5 * IQR_masvnr

# Capping outliers
train['MasVnrArea'] = np.where(train['MasVnrArea'] < lower_bound_masvnr, lower_b
                               np.where(train['MasVnrArea'] > upper_bound_masvnr

# Visualizing after capping
plt.figure(figsize=(12, 6))
sns.boxplot(train['MasVnrArea'])
plt.title('Box Plot of MasVnrArea After Handling Outliers')
plt.show()
```



In [ ]:
```
# List of features to handle outliers for
features_to_clean = ["BsmtFinSF1", "TotalBsmtSF", "1stFlrSF", "GarageArea", "Ope

# Function to clean outliers using IQR
def cap_outliers_iqr(data, feature):
    Q1 = data[feature].quantile(0.25)
    Q3 = data[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data[feature] = np.where(data[feature] < lower_bound, lower_bound, data[feat
    data[feature] = np.where(data[feature] > upper_bound, upper_bound, data[feat

# Applying the function on each feature
for feature in features_to_clean:
    cap_outliers_iqr(train, feature)

# Display summary statistics after cleaning
train[features_to_clean].describe()
```

Out[ ]:

	BsmtFinSF1	TotalBsmtSF	1stFlrSF	GarageArea	OpenPorchSF	WoodDeckSF
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	439.997517	1050.254795	1157.018151	470.670719	42.603425	91.806849
std	433.219435	397.937878	362.583002	207.105729	52.501584	116.658120
min	0.000000	42.000000	334.000000	0.000000	0.000000	0.000000
25%	0.000000	795.750000	882.000000	334.500000	0.000000	0.000000
50%	383.500000	991.500000	1087.000000	480.000000	25.000000	0.000000
75%	712.250000	1298.250000	1391.250000	576.000000	68.000000	168.000000
max	1780.625000	2052.000000	2155.125000	938.250000	170.000000	420.000000

In [ ]:

```
# Get categorical columns
categorical_cols = train.select_dtypes(include=['object']).columns

# Display the unique values for each categorical column to determine its nature
categorical_values = {col: train[col].unique() for col in categorical_cols}
categorical_values
```

Out[ ]:
{'MSZoning': array(['RL', 'RM', 'C (all)', 'FV', 'RH'], dtype=object),
 'Street': array(['Pave', 'Grvl'], dtype=object),
 'Alley': array(['None', 'Grvl', 'Pave'], dtype=object),
 'LandContour': array(['Lvl', 'Bnk', 'Low', 'HLS'], dtype=object),
 'LotConfig': array(['Inside', 'FR2', 'Corner', 'CulDSac', 'FR3'], dtype=objec
t),
 'Neighborhood': array(['CollgCr', 'Veenker', 'Crawfor', 'NoRidge', 'Mitchel',
'Somerst',
        'NWAmes', 'OldTown', 'BrkSide', 'Sawyer', 'NridgHt', 'NAmes',
        'SawyerW', 'IDOTRR', 'MeadowV', 'Edwards', 'Timber', 'Gilbert',
        'StoneBr', 'ClearCr', 'NPkVill', 'Blmngtn', 'BrDale', 'SWISU',
        'Blueste'], dtype=object),
 'Condition1': array(['Norm', 'Feedr', 'PosN', 'Artery', 'RRAe', 'RRNn', 'RRA
n', 'PosA',
        'RRNe'], dtype=object),
 'Condition2': array(['Norm', 'Artery', 'RRNn', 'Feedr', 'PosN', 'PosA', 'RRA
n', 'RRAe'],
       dtype=object),
 'BldgType': array(['1Fam', '2fmCon', 'Duplex', 'TwnhsE', 'Twnhs'], dtype=objec
t),
 'HouseStyle': array(['2Story', '1Story', '1.5Fin', '1.5Unf', 'SFoyer', 'SLvl',
'2.5Unf',
        '2.5Fin'], dtype=object),
 'RoofStyle': array(['Gable', 'Hip', 'Gambrel', 'Mansard', 'Flat', 'Shed'], dty
pe=object),
 'RoofMatl': array(['CompShg', 'WdShngl', 'Metal', 'WdShake', 'Membran', 'Tar&G
rv',
        'Roll', 'ClyTile'], dtype=object),
 'Exterior1st': array(['VinylSd', 'MetalSd', 'Wd Sdng', 'HdBoard', 'BrkFace',
'WdShing',
        'CemntBd', 'Plywood', 'AsbShng', 'Stucco', 'BrkComm', 'AsphShn',
        'Stone', 'ImStucc', 'CBlock'], dtype=object),
 'Exterior2nd': array(['VinylSd', 'MetalSd', 'Wd Shng', 'HdBoard', 'Plywood',
'Wd Sdng',
        'CmentBd', 'BrkFace', 'Stucco', 'AsbShng', 'Brk Cmn', 'ImStucc',
        'AsphShn', 'Stone', 'Other', 'CBlock'], dtype=object),
 'MasVnrType': array(['BrkFace', 'None', 'Stone', 'BrkCmn'], dtype=object),
 'Foundation': array(['PConc', 'CBlock', 'BrkTil', 'Wood', 'Slab', 'Stone'], dt
ype=object),
 'BsmtQual': array([4, 3, 5, 'None', 2], dtype=object),
 'BsmtCond': array([3, 4, 'None', 2, 1], dtype=object),
 'BsmtExposure': array([1, 4, 2, 3, 'None'], dtype=object),
 'BsmtFinType1': array([6, 5, 1, 3, 4, 'None', 2], dtype=object),
 'BsmtFinType2': array([1, 4, 'None', 5, 3, 2, 6], dtype=object),
 'Heating': array(['GasA', 'GasW', 'Grav', 'Wall', 'OthW', 'Floor'], dtype=obje
ct),
 'CentralAir': array(['Y', 'N'], dtype=object),
 'Electrical': array(['SBrkr', 'FuseF', 'FuseA', 'FuseP', 'Mix'], dtype=objec
t),
 'FireplaceQu': array(['None', 3, 4, 2, 5, 1], dtype=object),
 'GarageType': array(['Attchd', 'Detchd', 'BuiltIn', 'CarPort', 'None', 'Basmen
t',
        '2Types'], dtype=object),
 'GarageFinish': array([2, 1, 3, 'None'], dtype=object),
 'GarageQual': array([3, 2, 4, 'None', 5, 1], dtype=object),
 'GarageCond': array([3, 2, 'None', 4, 1, 5], dtype=object),
 'PoolQC': array(['None', 4, 1, 3], dtype=object),
 'Fence': array(['None', 3, 2, 4, 1], dtype=object),
 'MiscFeature': array(['None', 'Shed', 'Gar2', 'Othr', 'TenC'], dtype=object),
 'SaleType': array(['WD', 'New', 'COD', 'ConLD', 'ConLI', 'CWD', 'ConLw', 'Co

```
n', 'Oth'],
       dtype=object),
 'SaleCondition': array(['Normal', 'Abnorml', 'Partial', 'AdjLand', 'Alloca',
'Family'],
       dtype=object)}
```

In [ ]:
```
# Ordinal encoding for ordinal variables
ordinal_mappings = {
    'ExterQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'ExterCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'BsmtQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'BsmtCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'BsmtExposure': {'Gd': 4, 'Av': 3, 'Mn': 2, 'No': 1, 'NA': 0},
    'BsmtFinType1': {'GLQ': 6, 'ALQ': 5, 'BLQ': 4, 'Rec': 3, 'LwQ': 2, 'Unf': 1,
    'HeatingQC': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'KitchenQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1},
    'Functional': {'Typ': 8, 'Min1': 7, 'Min2': 6, 'Mod': 5, 'Maj1': 4, 'Maj2':
}

for col, mapping in ordinal_mappings.items():
    train[col] = train[col].map(mapping)

# One-hot encoding for nominal variables
train = pd.get_dummies(train, columns=['Electrical', 'GarageType'], drop_first=T

# Checking if any categorical variables remain
remaining_categorical = train.select_dtypes(include=['object']).columns
remaining_categorical
```

Out[ ]:
```
Index(['MSZoning', 'Street', 'Alley', 'LandContour', 'LotConfig',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'Foundation', 'BsmtFinType2', 'Heating', 'CentralAir', 'FireplaceQu',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
       'MiscFeature', 'SaleType', 'SaleCondition'],
      dtype='object')
```

In [ ]:
```
# Ordinal encoding for additional ordinal variables
additional_ordinal_mappings = {
    'FireplaceQu': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'GarageFinish': {'Fin': 3, 'RFn': 2, 'Unf': 1, 'NA': 0},
    'GarageQual': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'GarageCond': {'Ex': 5, 'Gd': 4, 'TA': 3, 'Fa': 2, 'Po': 1, 'NA': 0},
    'PoolQC': {'Ex': 4, 'Gd': 3, 'TA': 2, 'Fa': 1, 'NA': 0},
    'Fence': {'GdPrv': 4, 'MnPrv': 3, 'GdWo': 2, 'MnWw': 1, 'NA': 0},
    'Street': {'Pave': 2, 'Grvl': 1},
    'Alley': {'Pave': 2, 'Grvl': 1, 'NA': 0},
    'CentralAir': {'Y': 1, 'N': 0}
}

for col, mapping in additional_ordinal_mappings.items():
    train[col] = train[col].map(mapping)

# One-hot encoding for the remaining nominal variables
nominal_features = list(set(remaining_categorical) - set(additional_ordinal_mapp
train = pd.get_dummies(train, columns=nominal_features, drop_first=True)

# Checking if any categorical variables remain
remaining_categorical_final = train.select_dtypes(include=['object']).columns
remaining_categorical_final
```

Out[ ]:
```
Index([], dtype='object')
```

In [ ]:
```
# Checking data types of all columns
non_numerical_columns = train.select_dtypes(exclude=['int64', 'float64']).columr

non_numerical_columns
```

Out[ ]:
```
Index(['Electrical_FuseF', 'Electrical_FuseP', 'Electrical_Mix',
       'Electrical_SBrkr', 'GarageType_Attchd', 'GarageType_Basment',
       'GarageType_BuiltIn', 'GarageType_CarPort', 'GarageType_Detchd',
       'GarageType_None',
       ...
       'MiscFeature_TenC', 'BsmtFinType2_2', 'BsmtFinType2_3',
       'BsmtFinType2_4', 'BsmtFinType2_5', 'BsmtFinType2_6',
       'BsmtFinType2_None', 'MasVnrType_BrkFace', 'MasVnrType_None',
       'MasVnrType_Stone'],
      dtype='object', length=148)
```

In [ ]:
```
# Converting the non-numerical columns to numeric
train[non_numerical_columns] = train[non_numerical_columns].astype('int64')

# Check again for any non-numerical columns
remaining_non_numerical = train.select_dtypes(exclude=['int64', 'float64']).colu

remaining_non_numerical
```

Out[ ]:
```
Index([], dtype='object')
```

In [ ]:
```
# Extracting unique values for each column
unique_values = {col: train[col].unique() for col in train.columns}

# Displaying the unique values
unique_values
```

Out[ ]:
```
{'Id': array([   1,    2,    3, ..., 1458, 1459, 1460], dtype=int64),
 'MSSubClass': array([ 60,  20,  70,  50, 190,  45,  90, 120,  30,  85,  80, 160,  75,
        180,  40], dtype=int64),
 'LotFrontage': array([ 65. ,  80. ,  68. ,  60. ,  84. ,  85. ,  75. ,  51. ,  50. ,
         70. ,  71. ,  91. ,  73. ,  72. ,  66. , 101. ,  57. ,  44. ,
        110. ,  98. ,  47. , 108. ,  74. ,  66.5,  61. ,  48. ,  33. ,
         52. , 100. ,  30. ,  89. ,  63. ,  76. ,  81. ,  95. ,  69. ,
         32. ,  78. ,  40. , 105. ,  77. ,  64. ,  94. ,  34. ,  90. ,
         55. ,  88. ,  82. , 107. ,  92. ,  62. ,  86. ,  97. ,  73.5,
         54. ,  41. ,  79. ,  99. ,  67. ,  83. ,  43. , 103. ,  93. ,
         65.5,  35. ,  37. ,  87. ,  49. ,  96. ,  59. ,  36. ,  56. ,
        102. ,  58. ,  38. , 109. ,  53. ,  61.5,  45. , 106. , 104. ,
         42. ,  39. ,  88.5,  46. ]),
 'LotArea': array([2012.]),
 'Street': array([2, 1], dtype=int64),
 'Alley': array([nan,  1.,  2.]),
 'LotShape': array([4, 3, 2, 1], dtype=int64),
 'Utilities': array([4, 2], dtype=int64),
 'LandSlope': array([3, 2, 1], dtype=int64),
 'OverallQual': array([ 7,  6,  8,  5,  9,  4, 10,  3,  1,  2], dtype=int64),
 'OverallCond': array([5, 8, 6, 7, 4, 2, 3, 9, 1], dtype=int64),
 'YearBuilt': array([2003, 1976, 2001, 1915, 2000, 1993, 2004, 1973, 1931, 1939, 1965,
        2005, 1962, 2006, 1960, 1929, 1970, 1967, 1958, 1930, 2002, 1968,
        2007, 1951, 1957, 1927, 1920, 1966, 1959, 1994, 1954, 1953, 1955,
        1983, 1975, 1997, 1934, 1963, 1981, 1964, 1999, 1972, 1921, 1945,
        1982, 1998, 1956, 1948, 1910, 1995, 1991, 2009, 1950, 1961, 1977,
        1985, 1979, 1885, 1919, 1990, 1969, 1935, 1988, 1971, 1952, 1936,
        1923, 1924, 1984, 1926, 1940, 1941, 1987, 1986, 2008, 1908, 1892,
        1916, 1932, 1918, 1912, 1947, 1925, 1900, 1980, 1989, 1992, 1949,
        1880, 1928, 1978, 1922, 1996, 2010, 1946, 1913, 1937, 1942, 1938,
        1974, 1893, 1914, 1906, 1890, 1898, 1904, 1882, 1875, 1911, 1917,
        1872, 1905], dtype=int64),
 'YearRemodAdd': array([2003, 1976, 2002, 1970, 2000, 1995, 2005, 1973, 1950, 1965, 2006,
        1962, 2007, 1960, 2001, 1967, 2004, 2008, 1997, 1959, 1990, 1955,
        1983, 1980, 1966, 1963, 1987, 1964, 1972, 1996, 1998, 1989, 1953,
        1956, 1968, 1981, 1992, 2009, 1982, 1961, 1993, 1999, 1985, 1979,
        1977, 1969, 1958, 1991, 1971, 1952, 1975, 2010, 1984, 1986, 1994,
        1988, 1954, 1957, 1951, 1978, 1974], dtype=int64),
 'MasVnrArea': array([196.   ,   0.   , 162.   , 350.   , 186.   , 240.   , 286.   ,
        306.   , 212.   , 180.   , 380.   , 281.   , 410.625, 200.   ,
        246.   , 132.   , 101.   , 272.   , 178.   , 344.   , 287.   ,
        167.   ,  40.   , 104.   ,  66.   ,  22.   , 284.   ,  76.   ,
        203.   ,  68.   , 183.   ,  48.   ,  28.   , 336.   , 220.   ,
        184.   , 116.   , 135.   , 266.   ,  85.   , 309.   , 136.   ,
        288.   ,  70.   , 320.   ,  50.   , 120.   , 252.   ,  84.   ,
        226.   , 300.   , 112.   , 268.   ,  98.   , 275.   , 138.   ,
        205.   , 262.   , 128.   , 260.   , 153.   ,  64.   , 312.   ,
         16.   , 142.   , 290.   , 127.   , 297.   , 254.   ,  36.   ,
        102.   , 108.   , 302.   , 172.   , 399.   , 270.   ,  46.   ,
        210.   , 174.   , 348.   , 315.   , 299.   , 340.   , 166.   ,
         72.   ,  31.   ,  34.   , 238.   , 365.   ,  56.   , 150.   ,
        278.   , 256.   , 225.   , 370.   , 388.   , 175.   , 296.   ,
        146.   , 113.   , 176.   ,  30.   , 106.   , 362.   , 247.   ,
        305.   , 255.   , 125.   , 100.   , 126.   ,  74.   , 145.   ,
        232.   , 376.   ,  42.   , 161.   , 110.   ,  18.   , 224.   ,
```

```
        248.   ,  80.   , 304.   , 215.   , 378.   , 168.   ,  89.   ,
        285.   , 360.   ,  94.   , 333.   , 219.   , 188.   , 182.   ,
        250.   , 292.   , 245.   , 207.   ,  82.   ,  97.   , 335.   ,
        208.   , 170.   , 280.   ,  99.   , 192.   , 204.   , 233.   ,
        156.   , 261.   , 164.   , 259.   , 209.   , 263.   , 216.   ,
        351.   , 381.   ,  54.   , 258.   ,  57.   , 147.   , 293.   ,
        109.   ,  41.   , 160.   , 289.   , 169.   ,  95.   , 202.   ,
        338.   , 328.   ,   1.   , 375.   ,  90.   ,  38.   , 157.   ,
         11.   , 140.   , 130.   , 148.   , 243.   , 387.   , 223.   ,
        158.   , 137.   , 115.   , 189.   , 274.   , 117.   ,  60.   ,
        122.   ,  92.   ,  27.   ,  75.   , 361.   , 105.   , 342.   ,
        298.   , 236.   , 144.   ,  44.   , 151.   , 230.   ,  24.   ,
         53.   , 206.   ,  14.   , 324.   , 295.   , 396.   ,  67.   ,
        154.   ,  45.   , 337.   , 149.   , 143.   ,  51.   , 171.   ,
        234.   ,  63.   ,  32.   ,  81.   , 163.   , 218.   , 114.   ,
        359.   ,  86.   , 391.   , 228.   ,  88.   , 165.   , 410.   ,
        368.   , 318.   ,  65.   , 408.   , 244.   , 123.   , 366.   ,
        294.   , 310.   , 237.   ,  96.   , 194.   , 119.   ]),
 'ExterQual': array([nan]),
 'ExterCond': array([nan]),
 'BsmtQual': array([nan]),
 'BsmtCond': array([nan]),
 'BsmtExposure': array([nan]),
 'BsmtFinType1': array([nan]),
 'BsmtFinSF1': array([ 706.   ,  978.   ,  486.   ,  216.   ,  655.   ,  732.
,
       1369.   ,  859.   ,    0.   ,  851.   ,  906.   ,  998.   ,
        737.   ,  733.   ,  578.   ,  646.   ,  504.   ,  840.   ,
        188.   ,  234.   , 1218.   , 1277.   , 1018.   , 1153.   ,
       1213.   ,  731.   ,  643.   ,  967.   ,  747.   ,  280.   ,
        179.   ,  456.   , 1351.   ,   24.   ,  763.   ,  182.   ,
        104.   , 1780.625,  384.   ,  490.   ,  649.   ,  632.   ,
        941.   ,  739.   ,  912.   , 1013.   ,  603.   ,  565.   ,
        320.   ,  462.   ,  228.   ,  336.   ,  448.   , 1201.   ,
         33.   ,  588.   ,  600.   ,  713.   , 1046.   ,  648.   ,
        310.   , 1162.   ,  520.   ,  108.   ,  569.   , 1200.   ,
        224.   ,  705.   ,  444.   ,  250.   ,  984.   ,   35.   ,
        774.   ,  419.   ,  170.   , 1470.   ,  938.   ,  570.   ,
        300.   ,  120.   ,  116.   ,  512.   ,  567.   ,  445.   ,
        695.   ,  405.   , 1005.   ,  668.   ,  821.   ,  432.   ,
       1300.   ,  507.   ,  679.   , 1332.   ,  209.   ,  680.   ,
        716.   , 1400.   ,  416.   ,  429.   ,  222.   ,   57.   ,
        660.   , 1016.   ,  370.   ,  351.   ,  379.   , 1288.   ,
        360.   ,  639.   ,  495.   ,  288.   , 1398.   ,  477.   ,
        831.   ,  436.   ,  352.   ,  611.   , 1086.   ,  297.   ,
        626.   ,  560.   ,  390.   ,  566.   , 1126.   , 1036.   ,
       1088.   ,  641.   ,  617.   ,  662.   ,  312.   , 1065.   ,
        787.   ,  468.   ,   36.   ,  822.   ,  378.   ,  946.   ,
        341.   ,   16.   ,  550.   ,  524.   ,   56.   ,  321.   ,
        842.   ,  689.   ,  625.   ,  358.   ,  402.   ,   94.   ,
       1078.   ,  329.   ,  929.   ,  697.   , 1573.   ,  270.   ,
        922.   ,  503.   , 1334.   ,  361.   ,  672.   ,  506.   ,
        714.   ,  403.   ,  751.   ,  226.   ,  620.   ,  546.   ,
        392.   ,  421.   ,  905.   ,  904.   ,  430.   ,  614.   ,
        450.   ,  210.   ,  292.   ,  795.   , 1285.   ,  819.   ,
        420.   ,  841.   ,  281.   ,  894.   , 1464.   ,  700.   ,
        262.   , 1274.   ,  518.   , 1236.   ,  425.   ,  692.   ,
        987.   ,  970.   ,   28.   ,  256.   , 1619.   ,   40.   ,
        846.   , 1124.   ,  720.   ,  828.   , 1249.   ,  810.   ,
        213.   ,  585.   ,  129.   ,  498.   , 1270.   ,  573.   ,
```

```
1410.   , 1082.   ,  236.   ,  388.   ,  334.   ,  874.   ,
 956.   ,  773.   ,  399.   ,  162.   ,  712.   ,  609.   ,
 371.   ,  540.   ,   72.   ,  623.   ,  428.   ,  350.   ,
 298.   , 1445.   ,  218.   ,  985.   ,  631.   , 1280.   ,
 241.   ,  690.   ,  266.   ,  777.   ,  812.   ,  786.   ,
1116.   ,  789.   , 1056.   ,   50.   , 1128.   ,  775.   ,
1309.   , 1246.   ,  986.   ,  616.   , 1518.   ,  664.   ,
 387.   ,  471.   ,  385.   ,  365.   , 1767.   ,  133.   ,
 642.   ,  247.   ,  331.   ,  742.   , 1606.   ,  916.   ,
 185.   ,  544.   ,  553.   ,  326.   ,  778.   ,  386.   ,
 426.   ,  368.   ,  459.   , 1350.   , 1196.   ,  630.   ,
 994.   ,  168.   , 1261.   , 1567.   ,  299.   ,  897.   ,
 607.   ,  836.   ,  515.   ,  374.   , 1231.   ,  111.   ,
 356.   ,  400.   ,  698.   , 1247.   ,  257.   ,  380.   ,
  27.   ,  141.   ,  991.   ,  650.   ,  521.   , 1436.   ,
 719.   ,  377.   , 1330.   ,  348.   , 1219.   ,  783.   ,
 969.   ,  673.   , 1358.   , 1260.   ,  144.   ,  584.   ,
 554.   , 1002.   ,  619.   ,  180.   ,  559.   ,  308.   ,
 866.   ,  895.   ,  637.   ,  604.   , 1302.   , 1071.   ,
 290.   ,  728.   ,    2.   , 1441.   ,  943.   ,  231.   ,
 414.   ,  349.   ,  442.   ,  328.   ,  594.   ,  816.   ,
1460.   , 1324.   , 1338.   ,  685.   , 1422.   , 1283.   ,
  81.   ,  454.   ,  903.   ,  605.   ,  990.   ,  206.   ,
 150.   ,  457.   ,   48.   ,  871.   ,   41.   ,  674.   ,
 624.   ,  480.   , 1154.   ,  738.   ,  493.   , 1121.   ,
 282.   ,  500.   ,  131.   , 1696.   ,  806.   , 1361.   ,
 920.   , 1721.   ,  187.   , 1138.   ,  988.   ,  193.   ,
 551.   ,  767.   , 1186.   ,  892.   ,  311.   ,  827.   ,
 543.   , 1003.   , 1059.   ,  239.   ,  945.   ,   20.   ,
1455.   ,  965.   ,  980.   ,  863.   ,  533.   , 1084.   ,
1173.   ,  523.   , 1148.   ,  191.   , 1234.   ,  375.   ,
 808.   ,  724.   ,  152.   , 1180.   ,  252.   ,  832.   ,
 575.   ,  919.   ,  439.   ,  381.   ,  438.   ,  549.   ,
 612.   , 1163.   ,  437.   ,  394.   , 1416.   ,  422.   ,
 762.   ,  975.   , 1097.   ,  251.   ,  686.   ,  656.   ,
 568.   ,  539.   ,  862.   ,  197.   ,  516.   ,  663.   ,
 608.   , 1636.   ,  784.   ,  249.   , 1040.   ,  483.   ,
 196.   ,  572.   ,  338.   ,  330.   ,  156.   , 1390.   ,
 513.   ,  460.   ,  659.   ,  364.   ,  564.   ,  306.   ,
 505.   ,  932.   ,  750.   ,   64.   ,  633.   , 1170.   ,
 899.   ,  902.   , 1238.   ,  528.   , 1024.   , 1064.   ,
 285.   ,  465.   ,  322.   ,  860.   ,  599.   ,  354.   ,
  63.   ,  223.   ,  301.   ,  443.   ,  489.   ,  284.   ,
 294.   ,  814.   ,  165.   ,  552.   ,  833.   ,  464.   ,
 936.   ,  772.   , 1440.   ,  748.   ,  982.   ,  398.   ,
 562.   ,  484.   ,  417.   ,  699.   ,  696.   ,  896.   ,
 556.   , 1106.   ,  651.   ,  867.   ,  854.   , 1646.   ,
1074.   ,  536.   , 1172.   ,  915.   ,  595.   , 1237.   ,
 273.   ,  684.   ,  324.   , 1165.   ,  138.   , 1513.   ,
 317.   , 1012.   , 1022.   ,  509.   ,  900.   , 1085.   ,
1104.   ,  240.   ,  383.   ,  644.   ,  397.   ,  740.   ,
 837.   ,  220.   ,  586.   ,  535.   ,  410.   ,   75.   ,
 824.   ,  592.   , 1039.   ,  510.   ,  423.   ,  661.   ,
 248.   ,  704.   ,  412.   , 1032.   ,  219.   ,  708.   ,
 415.   , 1004.   ,  353.   ,  702.   ,  369.   ,  622.   ,
 212.   ,  645.   ,  852.   , 1150.   , 1258.   ,  275.   ,
 176.   ,  296.   ,  538.   , 1157.   ,  492.   , 1198.   ,
1387.   ,  522.   ,  658.   , 1216.   , 1480.   , 1159.   ,
 440.   , 1456.   ,  883.   ,  547.   ,  788.   ,  485.   ,
 340.   , 1220.   ,  427.   ,  344.   ,  756.   , 1540.   ,
```

```
         666.   ,  803.   , 1000.   ,  885.   , 1386.   ,  319.   ,
         534.   ,  125.   , 1314.   ,  602.   ,  192.   ,  593.   ,
         804.   , 1053.   ,  532.   , 1158.   , 1014.   ,  194.   ,
         167.   ,  776.   ,  694.   , 1572.   ,  746.   , 1406.   ,
         925.   ,  482.   ,  189.   ,  765.   ,   80.   , 1443.   ,
         259.   ,  735.   ,  734.   , 1447.   ,  548.   ,  315.   ,
        1282.   ,  408.   ,  309.   ,  203.   ,  865.   ,  204.   ,
         790.   , 1320.   ,  769.   , 1070.   ,  264.   ,  759.   ,
        1373.   ,  976.   ,  781.   ,   25.   , 1110.   ,  404.   ,
         580.   ,  678.   ,  958.   , 1336.   , 1079.   ,   49.   ,
         830.   ]),
 'BsmtFinSF2': array([   0,   32,  668,  486,   93,  491,  506,  712,  362,   4
1,  169,
         869,  150,  670,   28, 1080,  181,  768,  215,  374,  208,  441,
         184,  279,  306,  180,  580,  690,  692,  228,  125, 1063,  620,
         175,  820, 1474,  264,  479,  147,  232,  380,  544,  294,  258,
         121,  391,  531,  344,  539,  713,  210,  311, 1120,  165,  532,
          96,  495,  174, 1127,  139,  202,  645,  123,  551,  219,  606,
         612,  480,  182,  132,  336,  468,  287,   35,  499,  723,  119,
          40,  117,  239,   80,  472,   64, 1057,  127,  630,  128,  377,
         764,  345, 1085,  435,  823,  500,  290,  324,  634,  411,  841,
        1061,  466,  396,  354,  149,  193,  273,  465,  400,  682,  557,
         230,  106,  791,  240,  547,  469,  177,  108,  600,  492,  211,
         168, 1031,  438,  375,  144,   81,  906,  608,  276,  661,   68,
         173,  972,  105,  420,  546,  334,  352,  872,  110,  627,  163,
        1029], dtype=int64),
 'BsmtUnfSF': array([ 150,  284,  434,  540,  490,   64,  317,  216,  952,  14
0,  134,
         177,  175, 1494,  520,  832,  426,    0,  468,  525, 1158,  637,
        1777,  200,  204, 1566,  180,  486,  207,  649, 1228, 1234,  380,
         408, 1117, 1097,   84,  326,  445,  383,  167,  465, 1296,   83,
        1632,  736,  192,  612,  816,   32,  935,  321,  860, 1410,  148,
         217,  530, 1346,  576,  318, 1143, 1035,  440,  747,  701,  343,
         280,  404,  840,  724,  295, 1768,  448,   36, 1530, 1065,  384,
        1288,  684, 1013,  402,  635,  163,  168,  176,  370,  350,  381,
         410,  741, 1226, 1053,  641,  516,  793, 1139,  550,  905,  104,
         310,  252, 1125,  203,  728,  732,  510,  899, 1362,   30,  958,
         556,  413,  479,  297,  658,  262,  891, 1304,  519, 1907,  336,
         107,  432,  403,  811,  396,  970,  506,  884,  400,  896,  253,
         409,   93, 1200,  572,  774,  769, 1335,  340,  882,  779,  112,
         470,  294, 1686,  360,  441,  354,  700,  725,  320,  554,  312,
         968,  504, 1107,  577,  660,   99,  871,  474,  289,  600,  755,
         625, 1121,  276,  186, 1424, 1140,  375,   92,  305, 1176,   78,
         274,  311,  710,  686,  457, 1232, 1498, 1010,  160, 2336,  630,
         638,  162,   70, 1357, 1194,  773,  483,  235,  125, 1390,  594,
        1694,  488,  357,  626,  916, 1020, 1367,  798,  452,  392,  975,
         361,  270,  602, 1482,  680,  606,   88,  342,  212, 1095,   96,
         628, 1560,  744, 2121,  768,  386, 1468, 1145,  244,  698, 1079,
         570,  476,  131,  184,  143, 1092,  324, 1541, 1470,  536,  319,
         599,  622,  179,  292,  286,   80,  712,  291,  153, 1088, 1249,
         166,  906,  604,  100,  818,  844,  596,  210, 1603,  115,  103,
         673,  726,  995,  967,  721, 1656,  972,  460,  208,  191,  438,
        1869,  371,  624,  552,  322,  598,  268,  130,  484,  785,  733,
         953,  847,  333, 1580,  411,  982,  808, 1293,  939,  784,  595,
         229,  114,  522,  735,  405,  117,  961, 1286,  672, 1141,  806,
         165, 1064, 1063,  245, 1276,  892, 1008,  499, 1316,  463,  242,
         444,  281,   35,  356,  988,  580,  651,  619,  544,  387,  901,
         926,  135,  648,   75,  788, 1307, 1078, 1258,  273, 1436,  557,
         930,  780,  813,  878,  122,  248,  588,  524,  288,  389,  424,
        1375, 1626,  406,  298, 2153,  417,  739,  225,  611,  237,  290,
```

```
         264,  238,  363,  190, 1969,  697,  414,  316,  466,  420,  254,
         960,  397, 1191,  548,   50,  178, 1368,  169,  748,  689, 1264,
         467,  605, 1257,  551,  678,  707,  880,  378,  223,  578,  969,
         379,  765,  149,  912,  620, 1709,  132,  993,  197, 1374,   90,
         195,  706, 1163,  367, 1122, 1515,   55, 1497,  450,  846,   23,
         390,  861,  285, 1050,  331, 2042, 1237,  113,  742,  924,  512,
         119,  314,  308,  293,  537,  126,  427,  309,  914,  173, 1774,
         823,  485, 1116,  978,  636,  564,  108, 1184,  796,  366,  300,
         542,  645,  664,  756,  247,  776,  849, 1392,   38, 1406,  111,
         545,  121, 2046,  161,  261,  567, 1195,  874, 1342,  151,  989,
        1073,  927,  219,  224,  526, 1164,  761,  461,  876,  859,  171,
         718,  138,  941,  464,  250,   72,  508, 1584,  415,   82,  948,
         893,  864, 1349,   76,  487,  652, 1240,  801,  279, 1030,  348,
         234, 1198,  740,   89,  586,  323, 1836,  480,  456, 1935,  338,
        1594,  102,  374, 1413,  491, 1129,  255, 1496,  650, 1926,  154,
         999, 1734,  124, 1417,   15,  834, 1649,  936,  778, 1489,  442,
        1434,  352,  458, 1221, 1099,  416, 1800,  227,  907,  528,  189,
        1273,  563,  372,  702, 1090,  435,  198, 1372,  174, 1638,  894,
         299,  105,  676, 1120,  431,  218,  110,  795, 1098, 1043,  481,
         666,  142,  447,  783, 1670,  277,  412,  794,  239,  662, 1072,
         717,  546,  430,  422,  188,  266, 1181, 1753,  964, 1450, 1905,
        1480,  772, 1032,  220,  187,   29,  495,  640,  193,  196,  720,
         918, 1428,   77, 1266, 1128,  692,  770,  750, 1442, 1007,  501,
         691, 1550, 1680, 1330, 1710,  746,  814,  515,  571,  359,  355,
         301,  668,  920, 1055, 1420, 1752,  304, 1302,  833,  133,  549,
         705,  722,  799,  462,  429,  810,  155,  170,  230, 1459, 1082,
         758, 1290, 1074,  251,  172,  868,  797,  365,  418,  730,  533,
         671, 1012, 1528, 1005, 1373,  500,  762,  752,  399, 1042,   40,
          26,  932,  278,  459,  568, 1502,  543,  574,  977,  449,  983,
         731,  120,  538,  831,  994,  341,  879,  815, 1212,  866, 1630,
         328,  141,  364, 1380,   81,  303,  940,  764, 1048,  334, 1689,
         690,  792,  585,  473,  246, 1045, 1405,  201,   14,  841, 1104,
         241,  925, 2002,   74,  661,  708, 1152,  256,  804,  812, 1085,
         344,  425, 1616,  976,  496,  349,  971, 1393, 1622, 1352, 1795,
        1017, 1588,  428,  803,  693,  858, 1284, 1203, 1652,   39,  539,
        1217,  257,  715,  616,  240,  315, 1351, 1026, 1571,  156,   61,
          95,  482, 1094,   60,  862,  221,  791,  398,  777,  503,  734,
         709, 1252,  656, 1319, 1422,  560, 1573,  589,  877,  136],
       dtype=int64),
 'TotalBsmtSF': array([ 856., 1262.,  920.,  756., 1145.,  796., 1686., 1107.,
952.,
         991., 1040., 1175.,  912., 1494., 1253.,  832., 1004.,   42.,
        1114., 1029., 1158.,  637., 1777., 1060., 1566.,  900., 1704.,
        1484.,  520.,  649., 1228., 1234., 1398., 1561., 1117., 1097.,
        1297., 1057., 1088., 1350.,  840.,  938., 1150., 1752., 1434.,
        1656.,  736.,  955.,  794.,  816., 1842.,  384., 1425.,  970.,
         860., 1410.,  780.,  530., 1370.,  576., 1143., 1947., 1453.,
         747., 1304., 2052.,  845., 1086.,  462.,  672., 1768.,  440.,
         896., 1237., 1563., 1065., 1288.,  684.,  612., 1013.,  990.,
        1235.,  876., 1214.,  824.,  680., 1588.,  960.,  458.,  950.,
        1610.,  741., 1226., 1053.,  641.,  789.,  793., 1844.,  994.,
        1264., 1809., 1028.,  729., 1092., 1125., 1673.,  728.,  732.,
        1080., 1199., 1362., 1078.,  660., 1008.,  924.,  992., 1063.,
        1267., 1461., 1907.,  928.,  864., 1734.,  910., 1490., 1728.,
         715.,  884.,  969., 1710.,  825., 1602., 1200.,  572.,  774.,
        1392., 1232., 1572., 1541.,  882., 1149.,  644., 1617., 1582.,
         720., 1064., 1606., 1202., 1151., 1052.,  968.,  504., 1188.,
        1593.,  853.,  725., 1431.,  855., 1726., 1360.,  755., 1713.,
        1121., 1196.,  617.,  848., 1424., 1140., 1100., 1157., 1212.,
         689., 1070., 1436.,  686.,  798., 1248., 1498., 1010.,  713.,
```

```
        630., 1203.,  483., 1373., 1194., 1462.,  894., 1414.,  996.,
       1694.,  735.,  540.,  626.,  948., 1845., 1020., 1367., 1444.,
       1573., 1302., 1314.,  975., 1604.,  963., 1482.,  506.,  926.,
       1422.,  802.,  740., 1095., 1385., 1152., 1240., 1560., 1160.,
        807., 1468., 1575.,  625.,  858.,  698., 1079.,  768.,  795.,
       1416., 1003.,  702., 1165., 1470., 2000.,  700.,  319.,  861.,
       1896.,  697.,  972.,  716., 1347., 1372., 1249., 1136., 1502.,
       1162.,  710., 1719., 1383.,  844.,  596., 1056., 1358.,  943.,
       1499., 1922., 1536., 1208., 1215.,  967.,  721., 1684.,  536.,
        958., 1478.,  764., 1848., 1869.,  616.,  624.,  940., 1142.,
       1062.,  888.,  883., 1394., 1099., 1268.,  953.,  744.,  608.,
        847.,  683.,  870., 1580., 1856.,  982., 1026., 1293.,  939.,
        784., 1256.,  658., 1041., 1682.,  804.,  788., 1144.,  961.,
       1260., 1310., 1141.,  806., 1281., 1034., 1276., 1340., 1344.,
        988.,  651., 1518.,  907.,  901.,  765.,  799.,  648., 1440.,
       1258.,  915., 1517.,  930.,  813., 1533.,  872., 1242., 1364.,
        588.,  709.,  560., 1375., 1277., 1626., 1488.,  808.,  547.,
       1976., 1705., 1833., 1792., 1216.,  999., 1113., 1073.,  954.,
        264., 1269.,  190.,  866., 1501.,  777., 1218., 1368., 1084.,
       2006., 1244., 1379., 1257., 1452.,  528., 2035.,  611.,  707.,
        880., 1051., 1581., 1838., 1650.,  723.,  654., 1204., 1069.,
       1709.,  998.,  993., 1374., 1389., 1163., 1122., 1496.,  846.,
        372., 1164., 1050., 2042., 1868., 1437.,  742.,  770., 1722.,
       1814., 1430., 1058.,  908.,  600.,  965., 1032., 1299., 1120.,
        936.,  783., 1822., 1522.,  980., 1116.,  978., 1156.,  636.,
       1554., 1386.,  811., 1520., 1952., 1766.,  981., 1094.,  525.,
        776., 1486., 1629., 1138., 1406., 1021., 1408.,  738., 1477.,
       2046.,  923., 1291., 1195., 1190.,  874.,  551., 1419., 1210.,
        927., 1112., 1391., 1800.,  360., 1473., 1643., 1324.,  270.,
        859.,  718., 1176., 1311.,  971., 1742.,  941., 1698., 1584.,
       1595.,  868., 1153.,  893., 1349., 1337., 1720., 1479., 1030.,
       1318., 1252.,  983., 1860.,  836., 1935., 1614.,  761., 1413.,
        956.,  712.,  650.,  773., 1926.,  731., 1417., 1024.,  849.,
       1442., 1649., 1568.,  778., 1489., 1454., 1516., 1067., 1559.,
       1127., 1390., 1273.,  918., 1763., 1090., 1054., 1039., 1148.,
       1002., 1638.,  105.,  676., 1184., 1109.,  892., 1505., 1059.,
        951., 1670., 1623., 1017., 1105., 1001.,  546.,  480., 1134.,
       1104., 1272., 1316., 1126., 1181., 1753.,  964., 1466.,  925.,
       1905., 1500.,  585., 1632.,  819., 1616., 1161.,  828.,  945.,
        979.,  561.,  696., 1330.,  817., 1098., 1428.,  673., 1241.,
        944., 1225., 1266., 1128.,  485., 1930., 1396.,  916.,  822.,
        750., 1700., 1007., 1187.,  691., 1574., 1680., 1346.,  985.,
       1657.,  602., 1022., 1082.,  810., 1504., 1220., 1132., 1565.,
       1338., 1654., 1620., 1055.,  800., 1306., 1475., 1992., 1193.,
        973.,  854.,  662., 1103., 1154.,  942., 1048.,  727.,  690.,
       1096., 1459., 1251., 1247., 1074., 1271.,  290.,  655., 1463.,
       1836.,  803.,  833.,  408.,  533., 1012., 1552., 1005., 1530.,
        974., 1567., 1006., 1042., 1298.,  704.,  932., 1219., 1296.,
       1198.,  959., 1261., 1598., 1683.,  818., 1600., 1624.,  831.,
       1224.,  663.,  879.,  815., 1630.,  931., 1660.,  559., 1300.,
       1702., 1075., 1361., 1106., 1476., 1689.,  792., 1405., 1192.,
        746., 1986.,  841., 2002., 1332.,  935., 1019.,  661., 1309.,
       1328., 1085., 1246.,  771.,  976., 1652., 1278., 1902., 1274.,
       1393., 1622., 1352.,  420., 1795.,  544., 1510.,  911.,  693.,
       1284., 1732., 2033.,  570., 1980.,  814.,  873.,  757., 1108.,
       1571.,  984., 1205.,  714., 1746., 1525.,  482., 1356.,  862.,
        839., 1286., 1485., 1594.,  622.,  791.,  708., 1223.,  913.,
        656., 1319., 1932.,  539., 1221., 1542.]),
'HeatingQC': array([nan]),
'CentralAir': array([1, 0], dtype=int64),
```

```
'1stFlrSF': array([ 856.   , 1262.   ,  920.   ,  961.   , 1145.   ,  796.   ,
       1694.   , 1107.   , 1022.   , 1077.   , 1040.   , 1182.   ,
        912.   , 1494.   , 1253.   ,  854.   , 1004.   , 1296.   ,
       1114.   , 1339.   , 1158.   , 1108.   , 1795.   , 1060.   ,
       1600.   ,  900.   , 1704.   ,  520.   ,  649.   , 1228.   ,
       1234.   , 1700.   , 1561.   , 1132.   , 1097.   , 1297.   ,
       1057.   , 1152.   , 1324.   , 1328.   ,  884.   ,  938.   ,
       1150.   , 1752.   , 1518.   , 1656.   ,  736.   ,  955.   ,
        794.   ,  816.   , 1842.   , 1360.   , 1425.   ,  983.   ,
        860.   , 1426.   ,  780.   ,  581.   , 1370.   ,  902.   ,
       1143.   , 2155.125, 1479.   ,  747.   , 1304.   ,  845.   ,
        885.   , 1086.   ,  840.   ,  526.   ,  952.   , 1072.   ,
       1768.   ,  682.   , 1337.   , 1563.   , 1065.   ,  804.   ,
       1301.   ,  684.   ,  612.   , 1013.   ,  990.   , 1235.   ,
        964.   , 1260.   ,  905.   ,  680.   , 1588.   ,  960.   ,
        835.   , 1225.   , 1610.   ,  977.   , 1535.   , 1226.   ,
       1053.   , 1047.   ,  789.   ,  997.   , 1844.   , 1216.   ,
        774.   , 1282.   , 1436.   ,  729.   , 1092.   , 1125.   ,
       1699.   ,  728.   ,  988.   ,  772.   , 1080.   , 1199.   ,
       1586.   ,  958.   ,  660.   , 1327.   , 1721.   , 1682.   ,
       1214.   , 1959.   ,  928.   ,  864.   , 1734.   ,  910.   ,
       1501.   , 1728.   ,  970.   ,  875.   ,  896.   ,  969.   ,
       1710.   , 1252.   , 1200.   ,  572.   ,  991.   , 1392.   ,
       1232.   , 1572.   , 1541.   ,  882.   , 1149.   ,  808.   ,
       1867.   , 1707.   , 1064.   , 1362.   , 1651.   , 1164.   ,
        968.   ,  769.   ,  901.   , 1340.   ,  936.   , 1217.   ,
       1224.   , 1593.   , 1549.   ,  725.   , 1431.   ,  855.   ,
       1726.   ,  929.   , 1713.   , 1121.   , 1279.   ,  865.   ,
        848.   ,  720.   , 1442.   , 1696.   , 1100.   , 1180.   ,
       1212.   ,  932.   ,  689.   , 1236.   ,  810.   , 1137.   ,
       1248.   , 1498.   , 1010.   ,  811.   ,  630.   ,  483.   ,
       1555.   , 1194.   , 1490.   ,  894.   , 1414.   , 1014.   ,
        798.   , 1566.   ,  866.   ,  889.   ,  626.   , 1222.   ,
       1872.   ,  908.   , 1375.   , 1444.   , 1306.   , 1625.   ,
       1302.   , 1314.   , 1005.   , 1604.   ,  963.   , 1382.   ,
       1482.   ,  926.   ,  764.   , 1422.   ,  802.   , 1052.   ,
        778.   , 1113.   , 1095.   , 1363.   , 1632.   , 1560.   ,
       2121.   , 1156.   , 1175.   , 1468.   , 1575.   ,  625.   ,
       1085.   ,  858.   ,  698.   , 1079.   , 1148.   , 1644.   ,
       1003.   ,  975.   , 1041.   , 1336.   , 1210.   , 1675.   ,
       2000.   , 1122.   , 1035.   ,  861.   , 1944.   ,  697.   ,
        972.   ,  793.   , 2036.   ,  832.   ,  716.   , 1153.   ,
       1088.   , 1372.   , 1472.   , 1249.   , 1136.   , 1553.   ,
       1163.   , 1898.   ,  803.   , 1719.   , 1383.   , 1445.   ,
        596.   , 1056.   , 1629.   , 1358.   ,  943.   , 1619.   ,
       1922.   , 1536.   , 1621.   , 1215.   ,  993.   ,  841.   ,
       1684.   ,  536.   , 1478.   , 1848.   , 1869.   , 1453.   ,
        616.   , 1192.   , 1167.   , 1142.   , 1352.   ,  495.   ,
        790.   ,  672.   , 1394.   , 1268.   , 1287.   ,  953.   ,
       1120.   ,  752.   , 1319.   ,  847.   ,  904.   ,  914.   ,
       1580.   , 1856.   , 1007.   , 1026.   ,  939.   ,  784.   ,
       1269.   ,  658.   , 1742.   ,  788.   ,  735.   , 1144.   ,
        876.   , 1112.   , 1288.   , 1310.   , 1165.   ,  806.   ,
       1620.   , 1166.   , 1071.   , 1050.   , 1276.   , 1028.   ,
        756.   , 1344.   , 1602.   , 1470.   , 1196.   ,  707.   ,
        907.   , 1208.   , 1412.   ,  765.   ,  827.   ,  734.   ,
        694.   , 1440.   , 1128.   , 1258.   ,  933.   , 1689.   ,
       1888.   ,  956.   ,  679.   ,  813.   , 1533.   ,  888.   ,
        786.   , 1242.   ,  624.   , 1663.   ,  833.   ,  979.   ,
        575.   ,  849.   , 1277.   , 1634.   , 1502.   , 1161.   ,
```

```
1976.   , 1652.   , 1493.   , 2069.   , 1718.   , 1131.   ,
1850.   , 1792.   ,  916.   ,  999.   , 1073.   , 1484.   ,
1766.   ,  886.   , 1133.   ,  899.   , 1801.   , 1218.   ,
1368.   , 2020.   , 1378.   , 1244.   , 1266.   , 1476.   ,
 605.   , 1509.   ,  751.   ,  334.   ,  820.   ,  880.   ,
1159.   , 1601.   , 1838.   , 1680.   ,  767.   ,  664.   ,
1377.   ,  915.   ,  768.   ,  825.   , 1069.   , 1717.   ,
1126.   , 1006.   , 1048.   ,  897.   , 1557.   , 1389.   ,
 996.   , 1134.   , 1496.   ,  846.   ,  576.   ,  877.   ,
1320.   ,  703.   , 1429.   , 2042.   , 1521.   ,  989.   ,
2028.   ,  838.   , 1473.   ,  779.   ,  770.   ,  924.   ,
1826.   , 1402.   , 1647.   , 1058.   ,  927.   ,  600.   ,
1186.   , 1940.   , 1029.   , 1032.   , 1299.   , 1054.   ,
 807.   , 1828.   , 1548.   ,  980.   , 1012.   , 1116.   ,
1520.   , 1350.   , 1089.   , 1554.   , 1411.   ,  800.   ,
1567.   ,  981.   , 1094.   , 1051.   ,  822.   ,  755.   ,
 909.   , 2113.   ,  525.   ,  851.   , 1486.   , 1686.   ,
1181.   , 2097.   , 1454.   , 1465.   , 1679.   , 1437.   ,
 738.   , 1839.   ,  792.   , 2046.   ,  923.   , 1291.   ,
1668.   , 1195.   , 1190.   ,  874.   ,  551.   , 1419.   ,
1238.   , 1067.   , 1391.   , 1800.   , 1264.   ,  372.   ,
1824.   ,  859.   , 1576.   , 1178.   , 1325.   ,  971.   ,
1698.   , 1776.   , 1616.   , 1146.   ,  948.   , 1349.   ,
1464.   , 1720.   , 1038.   ,  742.   ,  757.   , 1506.   ,
1836.   , 1690.   , 1220.   , 1117.   , 1973.   , 1204.   ,
1614.   , 1430.   , 1110.   , 1342.   ,  966.   ,  976.   ,
1062.   , 1127.   , 1285.   ,  773.   , 1966.   , 1428.   ,
1075.   , 1309.   , 1044.   ,  686.   , 1661.   , 1008.   ,
 944.   , 1489.   , 2084.   , 1434.   , 1160.   ,  941.   ,
1516.   , 1559.   , 1099.   , 1701.   , 1307.   , 1456.   ,
 918.   , 1779.   ,  702.   , 1512.   , 1039.   , 1002.   ,
1646.   , 1547.   , 1036.   ,  676.   , 1184.   , 1462.   ,
1155.   , 1090.   , 1187.   ,  954.   ,  892.   , 1709.   ,
1712.   ,  872.   , 1505.   , 1068.   ,  951.   , 1670.   ,
1063.   , 1636.   , 1020.   , 1105.   , 1015.   , 1001.   ,
 546.   ,  480.   , 1229.   , 1272.   , 1316.   , 1617.   ,
1098.   , 1788.   , 1466.   ,  925.   , 1905.   , 1500.   ,
1207.   , 1188.   , 1381.   ,  965.   , 1168.   ,  561.   ,
 696.   , 1542.   ,  824.   ,  783.   ,  673.   ,  869.   ,
1241.   , 1118.   , 1407.   ,  750.   ,  691.   , 1574.   ,
1504.   ,  985.   , 1657.   , 1664.   , 1082.   , 1687.   ,
1654.   , 1055.   , 1803.   , 1532.   , 1733.   , 1992.   ,
1771.   ,  930.   , 1526.   , 1091.   , 1523.   , 1364.   ,
1130.   , 1096.   , 1338.   , 1103.   , 1154.   ,  799.   ,
 893.   ,  829.   , 1240.   , 1459.   , 1251.   , 1247.   ,
1390.   ,  438.   ,  950.   ,  887.   , 1021.   , 1552.   ,
 812.   , 1530.   ,  974.   ,  986.   , 1042.   , 1298.   ,
1811.   , 1265.   , 1640.   , 1432.   ,  959.   , 1831.   ,
1261.   , 1170.   , 2129.   ,  818.   , 1124.   ,  949.   ,
1624.   ,  831.   , 1622.   ,  842.   ,  663.   ,  879.   ,
 815.   , 1630.   , 1074.   , 1283.   , 1660.   , 1318.   ,
1211.   , 2136.   , 1138.   , 1702.   , 1507.   , 1361.   ,
1024.   , 1141.   , 1173.   , 2076.   , 1140.   , 1034.   ,
2110.   , 1405.   ,  760.   , 1987.   , 1104.   ,  713.   ,
2018.   , 1968.   , 1332.   ,  935.   , 1357.   ,  661.   ,
1724.   , 1573.   , 1582.   , 1659.   , 1246.   ,  753.   ,
1203.   , 1294.   , 1902.   , 1274.   , 1787.   , 1061.   ,
 708.   , 1584.   , 1334.   ,  693.   , 1284.   , 1172.   ,
2053.   ,  992.   , 1078.   , 1980.   , 1281.   ,  814.   ,
1571.   ,  984.   ,  754.   , 2117.   ,  998.   , 1416.   ,
```

```
        1746.   , 1525.   , 1221.   ,  741.   , 1569.   , 1223.   ,
         962.   , 1537.   , 1932.   , 1423.   ,  913.   , 1578.   ,
        2073.   , 1256.   ]),
 '2ndFlrSF': array([ 854,    0,  866,  756, 1053,  566,  983,  752, 1142, 1218,
668,
        1320,  631,  716,  676,  860, 1519,  530,  808,  977, 1330,  833,
         765,  462,  213,  548,  960,  670, 1116,  876,  612, 1031,  881,
         790,  755,  592,  939,  520,  639,  656, 1414,  884,  729, 1523,
         728,  351,  688,  941, 1032,  848,  836,  475,  739, 1151,  448,
         896,  524, 1194,  956, 1070, 1096,  467,  547,  551,  880,  703,
         901,  720,  316, 1518,  704, 1178,  754,  601, 1360,  929,  445,
         564,  882,  920,  518,  817, 1257,  741,  672, 1306,  504, 1304,
        1100,  730,  689,  591,  888, 1020,  828,  700,  842, 1286,  864,
         829, 1092,  709,  844, 1106,  596,  807,  625,  649,  698,  840,
         780,  568,  795,  648,  975,  702, 1242, 1818, 1121,  371,  804,
         325,  809, 1200,  871, 1274, 1347, 1332, 1177, 1080,  695,  167,
         915,  576,  605,  862,  495,  403,  838,  517, 1427,  784,  711,
         468, 1081,  886,  793,  665,  858,  874,  526,  590,  406, 1157,
         299,  936,  438, 1098,  766, 1101, 1028, 1017, 1254,  378, 1160,
         682,  110,  600,  678,  834,  384,  512,  930,  868,  224, 1103,
         560,  811,  878,  574,  910,  620,  687,  546,  902, 1000,  846,
        1067,  914,  660, 1538, 1015, 1237,  611,  707,  527, 1288,  832,
         806, 1182, 1040,  439,  717,  511, 1129, 1370,  636,  533,  745,
         584,  812,  684,  595,  988,  800,  677,  573, 1066,  778,  661,
        1440,  872,  788,  843,  713,  567,  651,  762,  482,  738,  586,
         679,  644,  900,  887, 1872, 1281,  472, 1312,  319,  978, 1093,
         473,  664, 1540, 1276,  441,  348, 1060,  714,  744, 1203,  783,
        1097,  734,  767, 1589,  742,  686, 1128, 1111, 1174,  787, 1072,
        1088, 1063,  545,  966,  623,  432,  581,  540,  769, 1051,  761,
         779,  514,  455, 1426,  785,  521,  252,  813, 1120, 1037, 1169,
        1001, 1215,  928, 1140, 1243,  571, 1196, 1038,  561,  979,  701,
         332,  368,  883, 1336, 1141,  634,  912,  798,  985,  826,  831,
         750,  456,  602,  855,  336,  408,  980,  998, 1168, 1208,  797,
         850,  898, 1054,  895,  954,  772, 1230,  727,  454,  370,  628,
         304,  582, 1122, 1134,  885,  640,  580, 1112,  653,  220,  240,
        1362,  534,  539,  650,  918,  933,  712, 1796,  971, 1175,  743,
         523, 1216, 2065,  272,  685,  776,  630,  984,  875,  913,  464,
        1039, 1259,  940,  892,  725,  924,  764,  925, 1479,  192,  589,
         992,  903,  430,  748,  587,  994,  950, 1323,  732, 1357,  557,
        1296,  390, 1185,  873, 1611,  457,  796,  908,  550,  989,  932,
         358, 1392,  349,  691, 1349,  768,  208,  622,  857,  556, 1044,
         708,  626,  904,  510, 1104,  830,  981,  870,  694, 1152],
       dtype=int64),
 'LowQualFinSF': array([  0, 360, 513, 234, 528, 572, 144, 392, 371, 390, 420,
473, 156,
         515,  80,  53, 232, 481, 120, 514, 397, 479, 205, 384], dtype=int64),
 'GrLivArea': array([1710, 1262, 1786, 1717, 2198, 1362, 1694, 2090, 1774, 107
7, 1040,
        2324,  912, 1494, 1253,  854, 1004, 1296, 1114, 1339, 2376, 1108,
        1795, 1060, 1600,  900, 1704,  520, 1317, 1228, 1234, 1700, 1561,
        2452, 1097, 1297, 1057, 1152, 1324, 1328,  884,  938, 1150, 1752,
        2149, 1656, 1452,  955, 1470, 1176,  816, 1842, 1360, 1425, 1739,
        1720, 2945,  780, 1158, 1111, 1370, 2034, 2473, 2207, 1479,  747,
        2287, 2223,  845, 1718, 1086, 1605,  988,  952, 1285, 1768, 1230,
        2142, 1337, 1563, 1065, 1474, 2417, 1560, 1224, 1526,  990, 1235,
         964, 2291, 1588,  960,  835, 1225, 1610, 1732, 1535, 1226, 1818,
        1992, 1047,  789, 1517, 1844, 1855, 1430, 2696, 2259, 2320, 1458,
        1092, 1125, 3222, 1456, 1123, 1080, 1199, 1586,  754,  958,  840,
        1348, 1053, 2157, 2054, 1327, 1721, 1682, 1214, 1959, 1852, 1764,
         864, 1734, 1385, 1501, 1728, 1709,  875, 2035, 1344,  969, 1993,
```

```
1252, 1200, 1096, 1968, 1947, 2462, 1232, 2668, 1541,  882, 1616,
1355, 1867, 2161, 1707, 1382, 1767, 1651, 2158, 2060, 1920, 2234,
 968, 1525, 1802, 1340, 2082, 3608, 1217, 1593, 2727, 1431, 1726,
3112, 2229, 1713, 1121, 1279, 1310,  848, 1284, 1442, 1696, 1100,
2062, 1212, 1392, 1236, 1436, 1954, 1248, 1498, 2267, 1552, 2392,
1302, 2520,  987, 1555, 1194, 2794,  894, 1960, 1414, 1744, 1487,
1566,  866, 1440, 2110, 1872, 1928, 1375, 1668, 2144, 1306, 1625,
1640, 1314, 1604, 1792, 2574, 1316,  764, 1422, 1511, 2192,  778,
1113, 1939, 1363, 2270, 1632, 1548, 2121, 2022, 1982, 1468, 1575,
1250,  858, 1396, 1919, 1716, 2263, 1644, 1003, 1558, 1950, 1743,
1336, 3493, 2000, 2243, 1406,  861, 1944,  972, 1118, 2036, 1641,
1432, 2353, 2646, 1472, 2596, 2468, 2730, 1163, 2978,  803, 1719,
1383, 2134, 1192, 1056, 1629, 1358, 1638, 1922, 1536, 1621, 1215,
1908,  841, 1684, 1112, 1577, 1478, 1626, 2728, 1869, 1453,  720,
1595, 1167, 1142, 1352, 1924, 1505, 1574, 1394, 1268, 1287, 1664,
 752, 1319,  904,  914, 2466, 1856, 1800, 1691, 1301, 1797,  784,
1953, 1269, 1184, 2332, 1367, 1961,  788, 1034, 1144, 1812, 1550,
1288,  672, 1572, 1620, 1639, 1680, 2172, 2078, 1276, 1028, 2097,
1400, 2624, 1134, 1602, 2630, 1196, 1389,  907, 1208, 1412, 1198,
1365,  630, 1661,  694, 2402, 1573, 1258, 1689, 1888, 1886, 1376,
1183,  813, 1533, 1756, 1590, 1242, 1663, 1666, 1203, 1935, 1135,
1660, 1277, 1634, 1502, 1969, 1072, 1976, 1652,  970, 1493, 2643,
1131, 1850, 1826, 1216,  999, 1073, 1484, 2414, 1304, 1578,  886,
3228, 1820,  899, 1218, 1801, 1322, 1911, 1378, 1041, 1368, 2020,
2119, 2344, 1796, 2080, 1294, 1244, 4676, 2398, 1266,  928, 2713,
 605, 2515, 1509,  827,  334, 1347, 1724, 1159, 1601, 1838, 2285,
 767, 1496, 2183, 1635,  768,  825, 2094, 1069, 1126, 2046, 1048,
1446, 1557,  996, 1674, 2295, 1647, 2504, 2132,  943, 1692, 1109,
1477, 1320, 1429, 2042, 2775, 2028,  838,  860, 1473,  935, 1582,
2296,  924, 1402, 1556, 1904, 1915, 1986, 2008, 3194, 1029, 2153,
1032, 1120, 1054,  832, 1828, 2262, 2614,  980, 1512, 1790, 1116,
1520, 1350, 1750, 1554, 1411, 3395,  800, 1387,  796, 1567, 1518,
1929, 2704, 1766,  981, 1094, 1839, 1665, 1510, 1469, 2113, 1486,
2448, 1181, 1936, 2380, 1679, 1437, 1180, 1476, 1369, 1136, 1441,
 792,  923, 1291, 1761, 1102, 1419, 4316, 2519, 1539, 1137,  616,
1148, 1391, 1164, 2576, 1824,  729, 1178, 2554, 2418,  971, 1742,
1698, 1776, 1146, 2031,  948, 1349, 1464, 2715, 2256, 2640, 1529,
1140, 2098, 1026, 1471, 1386, 2531, 1547, 2365, 1506, 1714, 1836,
3279, 1220, 1117, 1973, 1204, 1614, 1603, 1110, 1342, 2084,  901,
2087, 1145, 1062, 2013, 1895, 1564,  773, 3140, 1688, 2822, 1128,
1428, 1576, 2138, 1309, 1044, 1008, 1052,  936, 1733, 1489, 1434,
2126, 1223, 1829, 1516, 1067, 1559, 1099, 1482, 1165, 1416, 1701,
1775, 2358, 1646, 1445, 1779, 1481, 2654, 1426, 1039, 1372, 1002,
1949,  910, 2610, 2224, 1155, 1090, 2230,  892, 1712, 1393, 2217,
1683, 1068,  951, 2240, 2364, 1670,  902, 1063, 1636, 2057, 2274,
1015, 2002,  480, 1229, 2127, 2200, 1617, 1686, 2374, 1978, 1788,
2236, 1466,  925, 1905, 1500, 2069, 1971, 1962, 2403, 1381,  965,
1958, 2872, 1894, 1308, 1098, 1095,  918, 2019,  869, 1241, 2612,
2290, 1940, 2030, 1851, 1050,  944,  691, 1504,  985, 1657, 1522,
1271, 1022, 1082, 1132, 2898, 1264, 3082, 1654,  954, 1803, 2329,
2524, 2868, 1771,  930, 1977, 1989, 1523, 1364, 2184, 1991, 1338,
2337, 1103, 1154, 2260, 1571, 1611, 2521,  893, 1240, 1740, 1459,
1251, 1247, 1088,  438,  950, 2622, 2021, 1690, 1658, 1964,  833,
1012,  698, 1005, 1530, 1981,  974, 2210,  986, 1020, 1868, 2828,
1006, 1298,  932, 1811, 1265, 1580, 1876, 1671, 2108, 3627, 1261,
3086, 2345, 1343, 1124, 2514, 4476, 1130, 1221, 1699, 1624, 1804,
1622, 1863, 1630, 1074, 2196, 1283, 1845, 1902, 1211, 1846, 2136,
1490, 1138, 1933, 1702, 1507, 2620, 1190, 1188, 1784, 1948, 1141,
1173, 2076, 1553, 2058, 1405,  874, 2167, 1987, 1166, 1675, 1889,
2018, 3447, 1524, 1357, 1395, 2447, 1659, 1970, 2372, 5642, 1246,
```

```
        1983, 2526, 1708, 1122, 1274, 2810, 2599, 2112, 1787, 1923,  708,
         774, 2792, 1334,  693, 1861,  872, 2169, 1913, 2156, 2634, 3238,
        1865, 1078, 1980, 2601, 1738, 1475, 1374, 2633,  790, 2117, 1762,
        2784, 1746, 1584, 1912, 2482, 1687, 1513, 1608, 2093, 1840, 1848,
        1569, 2450, 2201,  804, 1537, 1932, 1725, 2555, 2007,  913, 1346,
        2073, 2340, 1256], dtype=int64),
 'BsmtFullBath': array([1, 0, 2, 3], dtype=int64),
 'BsmtHalfBath': array([0, 1, 2], dtype=int64),
 'FullBath': array([2, 1, 3, 0], dtype=int64),
 'HalfBath': array([1, 0, 2], dtype=int64),
 'BedroomAbvGr': array([3, 4, 1, 2, 0, 5, 6, 8], dtype=int64),
 'KitchenAbvGr': array([1, 2, 3, 0], dtype=int64),
 'KitchenQual': array([nan]),
 'TotRmsAbvGrd': array([ 8,  6,  7,  9,  5, 11,  4, 10, 12,  3,  2, 14], dtype=
int64),
 'Functional': array([nan]),
 'Fireplaces': array([0, 1, 2, 3], dtype=int64),
 'FireplaceQu': array([nan]),
 'GarageYrBlt': array([2003., 1976., 2001., 1998., 2000., 1993., 2004., 1973.,
1931.,
        1939., 1965., 2005., 1962., 2006., 1960., 1991., 1970., 1967.,
        1958., 1930., 2002., 1968., 2007., 2008., 1957., 1920., 1966.,
        1959., 1995., 1954., 1953., 1955., 1983., 1977., 1997., 1985.,
        1963., 1981., 1964., 1999., 1935., 1990., 1945., 1987., 1989.,
        1915., 1956., 1948., 1974., 2009., 1994., 1950., 1961., 1921.,
        1900., 1979., 1951., 1919., 1969., 1971., 1936., 1975., 1923.,
        1924., 1984., 1926., 1940., 1986., 1988., 1916., 1932., 1972.,
        1918., 1912., 1980., 1925., 1996., 1949., 1910., 1978., 1982.,
        1992., 1941., 1922., 2010., 1927., 1947., 1937., 1942., 1938.,
        1952., 1928., 1946., 1934., 1914., 1906., 1908., 1875., 1911.,
        1929., 1872., 1933.]),
 'GarageFinish': array([nan]),
 'GarageCars': array([2, 3, 1, 0, 4], dtype=int64),
 'GarageArea': array([548.  , 460.  , 608.  , 642.  , 836.  , 480.  , 636.  , 4
84.  ,
        468.  , 205.  , 384.  , 736.  , 352.  , 840.  , 576.  , 516.  ,
        294.  , 853.  , 280.  , 534.  , 572.  , 270.  , 890.  , 772.  ,
        319.  , 240.  , 250.  , 271.  , 447.  , 556.  , 691.  , 672.  ,
        498.  , 246.  ,   0.  , 440.  , 308.  , 504.  , 300.  , 670.  ,
        826.  , 386.  , 388.  , 528.  , 894.  , 565.  , 641.  , 288.  ,
        645.  , 852.  , 558.  , 220.  , 667.  , 360.  , 427.  , 490.  ,
        379.  , 297.  , 283.  , 509.  , 405.  , 758.  , 461.  , 400.  ,
        462.  , 420.  , 432.  , 506.  , 684.  , 472.  , 366.  , 476.  ,
        410.  , 740.  , 648.  , 273.  , 546.  , 325.  , 792.  , 450.  ,
        180.  , 430.  , 594.  , 390.  , 540.  , 264.  , 530.  , 435.  ,
        453.  , 750.  , 487.  , 624.  , 471.  , 318.  , 766.  , 660.  ,
        470.  , 720.  , 577.  , 380.  , 434.  , 866.  , 495.  , 564.  ,
        312.  , 625.  , 680.  , 678.  , 726.  , 532.  , 216.  , 303.  ,
        789.  , 511.  , 616.  , 521.  , 451.  , 938.25, 252.  , 497.  ,
        682.  , 666.  , 786.  , 795.  , 856.  , 473.  , 398.  , 500.  ,
        349.  , 454.  , 644.  , 299.  , 210.  , 431.  , 438.  , 675.  ,
        721.  , 336.  , 810.  , 494.  , 457.  , 818.  , 463.  , 604.  ,
        389.  , 538.  , 520.  , 309.  , 429.  , 673.  , 884.  , 868.  ,
        492.  , 413.  , 924.  , 439.  , 671.  , 338.  , 573.  , 732.  ,
        505.  , 575.  , 626.  , 898.  , 529.  , 685.  , 281.  , 539.  ,
        418.  , 588.  , 282.  , 375.  , 683.  , 843.  , 552.  , 870.  ,
        888.  , 746.  , 708.  , 513.  , 656.  , 872.  , 292.  , 441.  ,
        189.  , 880.  , 676.  , 301.  , 474.  , 706.  , 617.  , 445.  ,
        200.  , 592.  , 566.  , 514.  , 296.  , 244.  , 610.  , 834.  ,
        639.  , 501.  , 846.  , 560.  , 596.  , 600.  , 373.  , 350.  ,
```

```
        396.  , 864.  , 304.  , 784.  , 696.  , 569.  , 628.  , 550.  ,
        493.  , 578.  , 198.  , 422.  , 228.  , 526.  , 525.  , 908.  ,
        499.  , 508.  , 694.  , 874.  , 164.  , 402.  , 515.  , 286.  ,
        603.  , 900.  , 583.  , 889.  , 858.  , 502.  , 392.  , 403.  ,
        527.  , 765.  , 367.  , 426.  , 615.  , 871.  , 570.  , 406.  ,
        590.  , 612.  , 650.  , 275.  , 452.  , 842.  , 816.  , 621.  ,
        544.  , 486.  , 230.  , 261.  , 531.  , 393.  , 774.  , 749.  ,
        364.  , 627.  , 260.  , 256.  , 478.  , 442.  , 562.  , 512.  ,
        839.  , 330.  , 711.  , 416.  , 779.  , 702.  , 567.  , 832.  ,
        326.  , 551.  , 606.  , 739.  , 408.  , 475.  , 704.  , 768.  ,
        632.  , 541.  , 320.  , 800.  , 831.  , 554.  , 878.  , 752.  ,
        614.  , 481.  , 496.  , 423.  , 841.  , 895.  , 412.  , 865.  ,
        630.  , 605.  , 602.  , 618.  , 444.  , 397.  , 455.  , 409.  ,
        820.  , 598.  , 857.  , 595.  , 433.  , 776.  , 458.  , 613.  ,
        456.  , 436.  , 812.  , 686.  , 611.  , 425.  , 343.  , 479.  ,
        619.  , 902.  , 574.  , 523.  , 414.  , 738.  , 354.  , 483.  ,
        327.  , 756.  , 690.  , 284.  , 833.  , 601.  , 533.  , 522.  ,
        788.  , 555.  , 689.  , 796.  , 808.  , 510.  , 255.  , 424.  ,
        305.  , 368.  , 824.  , 328.  , 160.  , 437.  , 665.  , 290.  ,
        912.  , 905.  , 542.  , 716.  , 586.  , 467.  , 582.  , 254.  ,
        712.  , 719.  , 862.  , 928.  , 782.  , 466.  , 714.  , 225.  ,
        234.  , 324.  , 306.  , 830.  , 807.  , 358.  , 186.  , 693.  ,
        482.  , 813.  , 757.  , 459.  , 701.  , 322.  , 315.  , 668.  ,
        404.  , 543.  , 850.  , 477.  , 276.  , 518.  , 753.  , 213.  ,
        844.  , 860.  , 748.  , 248.  , 287.  , 825.  , 647.  , 342.  ,
        770.  , 663.  , 377.  , 804.  , 936.  , 722.  , 208.  , 662.  ,
        754.  , 622.  , 620.  , 370.  , 372.  , 923.  , 192.  ]),
 'GarageQual': array([nan]),
 'GarageCond': array([nan]),
 'PavedDrive': array([3, 1, 2], dtype=int64),
 'WoodDeckSF': array([  0., 298., 192.,  40., 255., 235.,  90., 147., 140., 16
0.,  48.,
        240., 171., 100., 406., 222., 288.,  49., 203., 113., 392., 145.,
        196., 168., 112., 106., 420., 115., 120.,  12., 301., 144., 300.,
         74., 127., 232., 158., 352., 182., 180., 166., 224.,  80., 367.,
         53., 188., 105.,  24.,  98., 276., 200., 409., 239., 400., 178.,
        237., 210., 116., 280., 104.,  87., 132., 238., 149., 355.,  60.,
        139., 108., 351., 209., 216., 248., 143., 365., 370.,  58., 197.,
        263., 123., 138., 333., 250., 292.,  95., 262.,  81., 289., 124.,
        172., 110., 208., 256., 302., 190., 340., 233., 184., 201., 142.,
        122., 155., 135., 306.,  64., 364., 353.,  66., 159., 146., 296.,
        125.,  44., 215., 264.,  88.,  89.,  96., 414., 206., 141., 260.,
        324., 156., 220.,  38., 261., 126.,  85., 270.,  78., 169., 320.,
        268.,  72., 349.,  42.,  35., 326., 382., 161., 179., 103., 253.,
        148., 335., 176., 390., 328., 312., 185., 269., 195.,  57., 236.,
        304., 198.,  28., 316., 322., 307., 257., 219., 416., 344., 380.,
         68., 114., 327., 165., 187., 181.,  92., 228., 245., 315., 241.,
        303., 133., 403.,  36.,  52., 265., 207., 150., 290., 278.,  70.,
        418., 234.,  26., 342.,  97., 272., 121., 243., 154., 164., 173.,
        384., 202.,  56., 321.,  86., 194., 305., 117., 153., 394., 371.,
         63., 252., 136., 186., 170., 214., 199.,  55., 361., 362., 162.,
        229., 379., 356.,  84., 325.,  33., 212., 314., 242., 294.,  30.,
        128.,  45., 177., 227., 218., 309., 404., 402., 283., 183., 175.,
        295.,  32., 366.]),
 'OpenPorchSF': array([ 61.,   0.,  42.,  35.,  84.,  30.,  57., 170.,   4.,  2
1.,  33.,
        112., 102., 154., 159., 110.,  90.,  56.,  32.,  50.,  54.,  65.,
         38.,  47.,  64.,  52., 138., 104.,  82.,  43., 146.,  75.,  72.,
         70.,  49.,  11.,  36., 151.,  29.,  94., 101.,  99., 162.,  63.,
         68.,  46.,  45., 122., 120.,  20.,  24., 130., 108.,  80.,  66.,
```

```
         48.,  25.,  96., 111., 106.,  40., 114.,   8., 136., 132.,  62.,
         60.,  27.,  74.,  16.,  26.,  83.,  34.,  55.,  22.,  98., 119.,
        105., 140., 168.,  28.,  39., 148.,  12.,  51., 150., 117.,  10.,
         81.,  44., 144., 128.,  76.,  17.,  59., 121.,  53., 134., 123.,
         78.,  85., 133., 113., 137., 125., 100.,  88., 155.,  73., 158.,
        142., 124.,  87.,  23., 152., 116., 160.,  18., 156., 166., 129.,
         77.,  67.,  69., 131.,  41., 118., 135.,  95., 169.,  58.,  93.,
         92., 103.,  91.,  86., 141.,  15., 126.]),
 'EnclosedPorch': array([  0, 272, 228, 205, 176,  87, 172, 102,  37, 144,  64,
114, 202,
        128, 156,  44,  77, 192, 140, 180, 183,  39, 184,  40, 552,  30,
        126,  96,  60, 150, 120, 112, 252,  52, 224, 234, 244, 268, 137,
         24, 108, 294, 177, 218, 242,  91, 160, 130, 169, 105,  34, 248,
        236,  32,  80, 115, 291, 116, 158, 210,  36, 200,  84, 148, 136,
        240,  54, 100, 189, 293, 164, 216, 239,  67,  90,  56, 129,  98,
        143,  70, 386, 154, 185, 134, 196, 264, 275, 230, 254,  68, 194,
        318,  48,  94, 138, 226, 174,  19, 170, 220, 214, 280, 190, 330,
        208, 145, 259,  81,  42, 123, 162, 286, 168,  20, 301, 198, 221,
        212,  50,  99], dtype=int64),
 '3SsnPorch': array([  0, 320, 407, 130, 180, 168, 140, 508, 238, 245, 196, 14
4, 182,
        162,  23, 216,  96, 153, 290, 304], dtype=int64),
 'ScreenPorch': array([  0, 176, 198, 291, 252,  99, 184, 168, 130, 142, 192, 4
10, 224,
        266, 170, 154, 153, 144, 128, 259, 160, 271, 234, 374, 185, 182,
         90, 396, 140, 276, 180, 161, 145, 200, 122,  95, 120,  60, 126,
        189, 260, 147, 385, 287, 156, 100, 216, 210, 197, 204, 225, 152,
        175, 312, 222, 265, 322, 190, 233,  63,  53, 143, 273, 288, 263,
         80, 163, 116, 480, 178, 440, 155, 220, 119, 165,  40], dtype=int64),
 'PoolArea': array([  0, 512, 648, 576, 555, 480, 519, 738], dtype=int64),
 'PoolQC': array([nan]),
 'Fence': array([nan]),
 'MiscVal': array([    0,   700,   350,   500,   400,   480,   450, 15500,  120
0,
          800,  2000,   600,  3500,  1300,    54,   620,   560,  1400,
         8300,  1150,  2500], dtype=int64),
 'MoSold': array([ 2,  5,  9, 12, 10,  8, 11,  4,  1,  7,  3,  6], dtype=int6
4),
 'YrSold': array([2008, 2007, 2006, 2009, 2010], dtype=int64),
 'SalePrice': array([208500. , 181500. , 223500. , 140000. , 250000. , 143000.
,
        307000. , 200000. , 129900. , 118000. , 129500. , 340037.5,
        144000. , 279500. , 157000. , 132000. , 149000. ,  90000. ,
        159000. , 139000. , 325300. , 139400. , 230000. , 154000. ,
        256300. , 134800. , 306000. , 207500. ,  68500. ,  40000. ,
        149350. , 179900. , 165500. , 277500. , 309000. , 145000. ,
        153000. , 109000. ,  82000. , 160000. , 170000. , 130250. ,
        141000. , 319900. , 239686. , 249700. , 113000. , 127000. ,
        177000. , 114500. , 110000. , 130000. , 180500. , 172500. ,
        196500. , 124900. , 158000. , 101000. , 202500. , 219500. ,
        317000. , 180000. , 226000. ,  80000. , 225000. , 244000. ,
        185000. , 144900. , 107400. ,  91000. , 135750. , 136500. ,
        193500. , 153500. , 245000. , 126500. , 168500. , 260000. ,
        174000. , 164500. ,  85000. , 123600. , 109900. ,  98600. ,
        163500. , 133900. , 204750. , 214000. ,  94750. ,  83000. ,
        128950. , 205000. , 178000. , 118964. , 198900. , 169500. ,
        100000. , 115000. , 190000. , 136900. , 217000. , 259500. ,
        176000. , 155000. , 320000. , 163990. , 136000. , 153900. ,
        181000. ,  84500. , 128000. ,  87000. , 150000. , 150750. ,
        220000. , 171000. , 231500. , 166000. , 204000. , 125000. ,
```

```
105000. , 222500. , 122000. , 235000. ,  79000. , 109500. ,
269500. , 254900. , 162500. , 103200. , 152000. , 127500. ,
325624. , 183500. , 228000. , 128500. , 215000. , 239000. ,
163000. , 184000. , 243000. , 211000. , 200100. , 120000. ,
173000. , 135000. , 153337. , 286000. , 315000. , 192000. ,
148500. , 311872. , 104000. , 274900. , 171500. , 112000. ,
143900. , 277000. ,  98000. , 186000. , 252678. , 156000. ,
161750. , 134450. , 210000. , 107000. , 311500. , 167240. ,
204900. ,  97000. , 290000. , 106000. , 192500. , 148000. ,
 94500. , 128200. , 216500. ,  89500. , 185500. , 194500. ,
318000. , 262500. , 110500. , 241500. , 137000. ,  76500. ,
276000. , 151000. ,  73000. , 175500. , 179500. , 120500. ,
266000. , 124500. , 201000. , 228500. , 244600. , 179200. ,
164700. ,  88000. , 153575. , 233230. , 135900. , 131000. ,
167000. , 142500. , 175000. , 158500. , 267000. , 149900. ,
295000. , 305900. ,  82500. , 165600. , 119900. , 188500. ,
270000. , 187500. , 301000. , 126175. , 242000. , 324000. ,
145250. , 214500. ,  78000. , 119000. , 284000. , 207000. ,
228950. , 202900. ,  87500. , 140200. , 151500. , 157500. ,
318061. ,  95000. , 105900. , 177500. , 134000. , 280000. ,
198500. , 147000. , 165000. , 162000. , 172400. , 134432. ,
123000. ,  61000. , 340000. , 179000. , 187750. , 213500. ,
 76000. , 240000. ,  81000. , 191000. , 106500. , 129000. ,
 67000. , 241000. , 245500. , 164990. , 108000. , 258000. ,
168000. , 339750. ,  60000. , 222000. , 181134. , 149500. ,
126000. , 142000. , 206300. , 275000. , 109008. , 195400. ,
 85400. ,  79900. , 122500. , 212000. , 116000. ,  90350. ,
162900. , 199900. , 119500. , 188000. , 256000. , 161000. ,
263435. ,  62383. , 188700. , 124000. , 178740. , 146500. ,
187000. , 251000. , 132500. , 208900. , 297000. ,  89471. ,
326000. , 164000. ,  86000. , 133000. , 172785. ,  91300. ,
 34900. , 226700. , 289000. , 208300. , 164900. , 202665. ,
 96500. , 265000. , 234000. , 106250. , 184750. , 315750. ,
200624. , 107500. ,  39300. , 111250. , 272000. , 248000. ,
213250. , 179665. , 229000. , 263000. , 112500. , 255500. ,
121500. , 268000. , 325000. , 316600. , 135960. , 142600. ,
224500. , 118500. , 146000. , 131500. , 181900. , 253293. ,
 79500. , 185900. , 138000. , 319000. , 114504. , 194201. ,
217500. , 221000. , 313000. , 261500. ,  75500. , 137500. ,
183200. , 105500. , 314813. , 305000. , 165150. , 139900. ,
209500. ,  93000. , 264561. , 274000. , 143250. ,  98300. ,
205950. , 145500. ,  97500. , 197900. , 230500. , 173500. ,
103600. , 257500. , 159434. , 285000. , 227875. , 148800. ,
194700. , 335000. , 108480. , 141500. ,  89000. , 123500. ,
138500. , 196000. , 312500. , 213000. ,  55000. , 302000. ,
254000. , 179540. ,  52000. , 102776. , 189000. , 130500. ,
159500. , 103000. , 236500. , 131400. ,  93500. , 239900. ,
299800. , 236000. , 265979. , 260400. , 275500. , 158900. ,
179400. , 215200. , 337000. , 264132. , 216837. , 134900. ,
102000. , 221500. , 175900. , 187100. , 161500. , 233000. ,
107900. , 160200. , 146800. , 269790. , 143500. , 227680. ,
135500. , 159950. , 144500. ,  55993. , 157900. , 224900. ,
271000. , 224000. , 183000. , 139500. , 232600. , 147400. ,
237000. , 139950. , 174900. , 133500. , 189950. , 250580. ,
248900. , 169000. , 200500. ,  66500. , 303477. , 132250. ,
328900. , 122900. , 154500. , 118858. , 142953. , 125500. ,
255000. , 154300. , 173733. ,  75000. ,  35311. , 238000. ,
176500. , 145900. , 169990. , 193000. , 117500. , 184900. ,
253000. , 239799. , 244400. , 150900. , 197500. , 172000. ,
116500. , 214900. , 178900. ,  37900. ,  99500. , 182000. ,
```

```
       167500. ,  85500. , 178400. , 336000. , 159895. , 255900. ,
       117000. , 195000. , 197000. , 173900. , 337500. , 121600. ,
       206000. , 232000. , 136905. , 119200. , 227000. , 203000. ,
       213490. , 194000. , 287000. , 293077. , 310000. , 119750. ,
        84000. , 315500. , 262280. , 278000. , 139600. ,  84900. ,
       176485. , 200141. , 185850. , 328000. , 167900. , 151400. ,
        91500. , 138800. , 155900. ,  83500. , 252000. ,  92900. ,
       176432. , 274725. , 134500. , 184100. , 133700. , 118400. ,
       212900. , 163900. , 259000. , 239500. ,  94000. , 174500. ,
       116900. , 201800. , 218000. , 235128. , 108959. , 233170. ,
       245350. , 171900. , 154900. , 186700. , 104900. , 262000. ,
       219210. , 116050. , 271900. , 229456. ,  80500. , 137900. ,
       101800. , 138887. , 265900. , 248328. , 186500. , 169900. ,
       171750. , 294000. , 165400. , 301500. ,  99900. , 128900. ,
       183900. , 185750. ,  68400. , 150500. , 281000. , 333168. ,
       206900. , 295493. , 111000. , 156500. ,  72500. ,  52500. ,
       155835. , 108500. , 283463. , 156932. , 144152. , 216000. ,
       274300. ,  58500. , 237500. , 246578. , 281213. , 137450. ,
       193879. , 282922. , 257000. , 223000. , 274970. , 182900. ,
       192140. , 143750. ,  64500. , 149700. , 149300. , 121000. ,
       179600. ,  92000. , 287090. , 266500. , 142125. , 147500. ]),
'Electrical_FuseF': array([0, 1], dtype=int64),
'Electrical_FuseP': array([0, 1], dtype=int64),
'Electrical_Mix': array([0, 1], dtype=int64),
'Electrical_SBrkr': array([1, 0], dtype=int64),
'GarageType_Attchd': array([1, 0], dtype=int64),
'GarageType_Basment': array([0, 1], dtype=int64),
'GarageType_BuiltIn': array([0, 1], dtype=int64),
'GarageType_CarPort': array([0, 1], dtype=int64),
'GarageType_Detchd': array([0, 1], dtype=int64),
'GarageType_None': array([0, 1], dtype=int64),
'LotConfig_CulDSac': array([0, 1], dtype=int64),
'LotConfig_FR2': array([0, 1], dtype=int64),
'LotConfig_FR3': array([0, 1], dtype=int64),
'LotConfig_Inside': array([1, 0], dtype=int64),
'RoofMatl_CompShg': array([1, 0], dtype=int64),
'RoofMatl_Membran': array([0, 1], dtype=int64),
'RoofMatl_Metal': array([0, 1], dtype=int64),
'RoofMatl_Roll': array([0, 1], dtype=int64),
'RoofMatl_Tar&Grv': array([0, 1], dtype=int64),
'RoofMatl_WdShake': array([0, 1], dtype=int64),
'RoofMatl_WdShngl': array([0, 1], dtype=int64),
'RoofStyle_Gable': array([1, 0], dtype=int64),
'RoofStyle_Gambrel': array([0, 1], dtype=int64),
'RoofStyle_Hip': array([0, 1], dtype=int64),
'RoofStyle_Mansard': array([0, 1], dtype=int64),
'RoofStyle_Shed': array([0, 1], dtype=int64),
'Exterior1st_AsphShn': array([0, 1], dtype=int64),
'Exterior1st_BrkComm': array([0, 1], dtype=int64),
'Exterior1st_BrkFace': array([0, 1], dtype=int64),
'Exterior1st_CBlock': array([0, 1], dtype=int64),
'Exterior1st_CemntBd': array([0, 1], dtype=int64),
'Exterior1st_HdBoard': array([0, 1], dtype=int64),
'Exterior1st_ImStucc': array([0, 1], dtype=int64),
'Exterior1st_MetalSd': array([0, 1], dtype=int64),
'Exterior1st_Plywood': array([0, 1], dtype=int64),
'Exterior1st_Stone': array([0, 1], dtype=int64),
'Exterior1st_Stucco': array([0, 1], dtype=int64),
'Exterior1st_VinylSd': array([1, 0], dtype=int64),
'Exterior1st_Wd Sdng': array([0, 1], dtype=int64),
```

```
'Exterior1st_WdShing': array([0, 1], dtype=int64),
'MSZoning_FV': array([0, 1], dtype=int64),
'MSZoning_RH': array([0, 1], dtype=int64),
'MSZoning_RL': array([1, 0], dtype=int64),
'MSZoning_RM': array([0, 1], dtype=int64),
'HouseStyle_1.5Unf': array([0, 1], dtype=int64),
'HouseStyle_1Story': array([0, 1], dtype=int64),
'HouseStyle_2.5Fin': array([0, 1], dtype=int64),
'HouseStyle_2.5Unf': array([0, 1], dtype=int64),
'HouseStyle_2Story': array([1, 0], dtype=int64),
'HouseStyle_SFoyer': array([0, 1], dtype=int64),
'HouseStyle_SLvl': array([0, 1], dtype=int64),
'SaleCondition_AdjLand': array([0, 1], dtype=int64),
'SaleCondition_Alloca': array([0, 1], dtype=int64),
'SaleCondition_Family': array([0, 1], dtype=int64),
'SaleCondition_Normal': array([1, 0], dtype=int64),
'SaleCondition_Partial': array([0, 1], dtype=int64),
'SaleType_CWD': array([0, 1], dtype=int64),
'SaleType_Con': array([0, 1], dtype=int64),
'SaleType_ConLD': array([0, 1], dtype=int64),
'SaleType_ConLI': array([0, 1], dtype=int64),
'SaleType_ConLw': array([0, 1], dtype=int64),
'SaleType_New': array([0, 1], dtype=int64),
'SaleType_Oth': array([0, 1], dtype=int64),
'SaleType_WD': array([1, 0], dtype=int64),
'LandContour_HLS': array([0, 1], dtype=int64),
'LandContour_Low': array([0, 1], dtype=int64),
'LandContour_Lvl': array([1, 0], dtype=int64),
'Condition2_Feedr': array([0, 1], dtype=int64),
'Condition2_Norm': array([1, 0], dtype=int64),
'Condition2_PosA': array([0, 1], dtype=int64),
'Condition2_PosN': array([0, 1], dtype=int64),
'Condition2_RRAe': array([0, 1], dtype=int64),
'Condition2_RRAn': array([0, 1], dtype=int64),
'Condition2_RRNn': array([0, 1], dtype=int64),
'Foundation_CBlock': array([0, 1], dtype=int64),
'Foundation_PConc': array([1, 0], dtype=int64),
'Foundation_Slab': array([0, 1], dtype=int64),
'Foundation_Stone': array([0, 1], dtype=int64),
'Foundation_Wood': array([0, 1], dtype=int64),
'Condition1_Feedr': array([0, 1], dtype=int64),
'Condition1_Norm': array([1, 0], dtype=int64),
'Condition1_PosA': array([0, 1], dtype=int64),
'Condition1_PosN': array([0, 1], dtype=int64),
'Condition1_RRAe': array([0, 1], dtype=int64),
'Condition1_RRAn': array([0, 1], dtype=int64),
'Condition1_RRNe': array([0, 1], dtype=int64),
'Condition1_RRNn': array([0, 1], dtype=int64),
'BldgType_2fmCon': array([0, 1], dtype=int64),
'BldgType_Duplex': array([0, 1], dtype=int64),
'BldgType_Twnhs': array([0, 1], dtype=int64),
'BldgType_TwnhsE': array([0, 1], dtype=int64),
'Neighborhood_Blueste': array([0, 1], dtype=int64),
'Neighborhood_BrDale': array([0, 1], dtype=int64),
'Neighborhood_BrkSide': array([0, 1], dtype=int64),
'Neighborhood_ClearCr': array([0, 1], dtype=int64),
'Neighborhood_CollgCr': array([1, 0], dtype=int64),
'Neighborhood_Crawfor': array([0, 1], dtype=int64),
'Neighborhood_Edwards': array([0, 1], dtype=int64),
'Neighborhood_Gilbert': array([0, 1], dtype=int64),
```

```
 'Neighborhood_IDOTRR': array([0, 1], dtype=int64),
 'Neighborhood_MeadowV': array([0, 1], dtype=int64),
 'Neighborhood_Mitchel': array([0, 1], dtype=int64),
 'Neighborhood_NAmes': array([0, 1], dtype=int64),
 'Neighborhood_NPkVill': array([0, 1], dtype=int64),
 'Neighborhood_NWAmes': array([0, 1], dtype=int64),
 'Neighborhood_NoRidge': array([0, 1], dtype=int64),
 'Neighborhood_NridgHt': array([0, 1], dtype=int64),
 'Neighborhood_OldTown': array([0, 1], dtype=int64),
 'Neighborhood_SWISU': array([0, 1], dtype=int64),
 'Neighborhood_Sawyer': array([0, 1], dtype=int64),
 'Neighborhood_SawyerW': array([0, 1], dtype=int64),
 'Neighborhood_Somerst': array([0, 1], dtype=int64),
 'Neighborhood_StoneBr': array([0, 1], dtype=int64),
 'Neighborhood_Timber': array([0, 1], dtype=int64),
 'Neighborhood_Veenker': array([0, 1], dtype=int64),
 'Heating_GasA': array([1, 0], dtype=int64),
 'Heating_GasW': array([0, 1], dtype=int64),
 'Heating_Grav': array([0, 1], dtype=int64),
 'Heating_OthW': array([0, 1], dtype=int64),
 'Heating_Wall': array([0, 1], dtype=int64),
 'Exterior2nd_AsphShn': array([0, 1], dtype=int64),
 'Exterior2nd_Brk Cmn': array([0, 1], dtype=int64),
 'Exterior2nd_BrkFace': array([0, 1], dtype=int64),
 'Exterior2nd_CBlock': array([0, 1], dtype=int64),
 'Exterior2nd_CmentBd': array([0, 1], dtype=int64),
 'Exterior2nd_HdBoard': array([0, 1], dtype=int64),
 'Exterior2nd_ImStucc': array([0, 1], dtype=int64),
 'Exterior2nd_MetalSd': array([0, 1], dtype=int64),
 'Exterior2nd_Other': array([0, 1], dtype=int64),
 'Exterior2nd_Plywood': array([0, 1], dtype=int64),
 'Exterior2nd_Stone': array([0, 1], dtype=int64),
 'Exterior2nd_Stucco': array([0, 1], dtype=int64),
 'Exterior2nd_VinylSd': array([1, 0], dtype=int64),
 'Exterior2nd_Wd Sdng': array([0, 1], dtype=int64),
 'Exterior2nd_Wd Shng': array([0, 1], dtype=int64),
 'MiscFeature_None': array([1, 0], dtype=int64),
 'MiscFeature_Othr': array([0, 1], dtype=int64),
 'MiscFeature_Shed': array([0, 1], dtype=int64),
 'MiscFeature_TenC': array([0, 1], dtype=int64),
 'BsmtFinType2_2': array([0, 1], dtype=int64),
 'BsmtFinType2_3': array([0, 1], dtype=int64),
 'BsmtFinType2_4': array([0, 1], dtype=int64),
 'BsmtFinType2_5': array([0, 1], dtype=int64),
 'BsmtFinType2_6': array([0, 1], dtype=int64),
 'BsmtFinType2_None': array([0, 1], dtype=int64),
 'MasVnrType_BrkFace': array([1, 0], dtype=int64),
 'MasVnrType_None': array([0, 1], dtype=int64),
 'MasVnrType_Stone': array([0, 1], dtype=int64)}
```

In [ ]:
```
# Finding missing values in the dataset
missing_data = train.isnull().sum()
missing_data = missing_data[missing_data > 0].sort_values(ascending=False)

missing_data
```

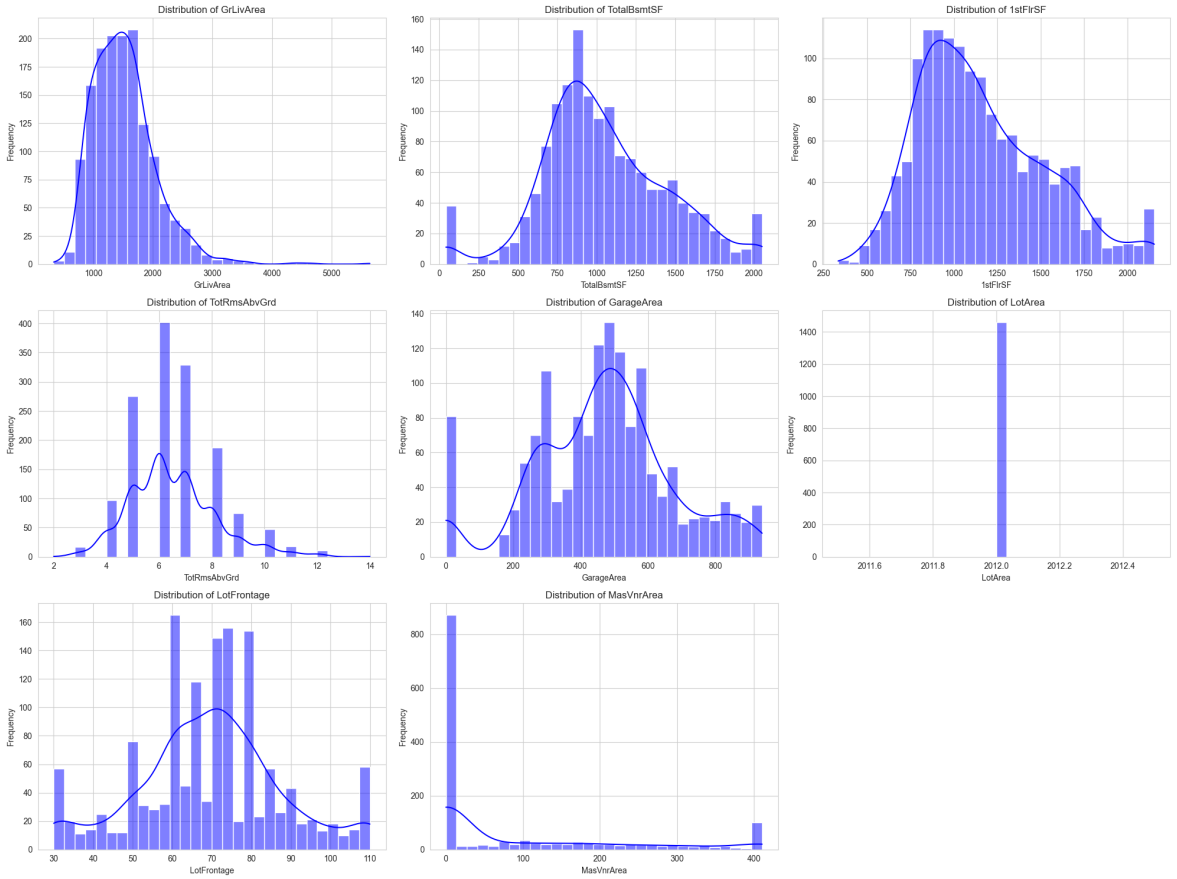
Out[ ]:
```
ExterQual       1460
ExterCond       1460
BsmtQual        1460
BsmtCond        1460
BsmtExposure    1460
BsmtFinType1    1460
HeatingQC       1460
KitchenQual     1460
Functional      1460
FireplaceQu     1460
GarageFinish    1460
GarageQual      1460
GarageCond      1460
PoolQC          1460
Fence           1460
Alley           1369
dtype: int64
```

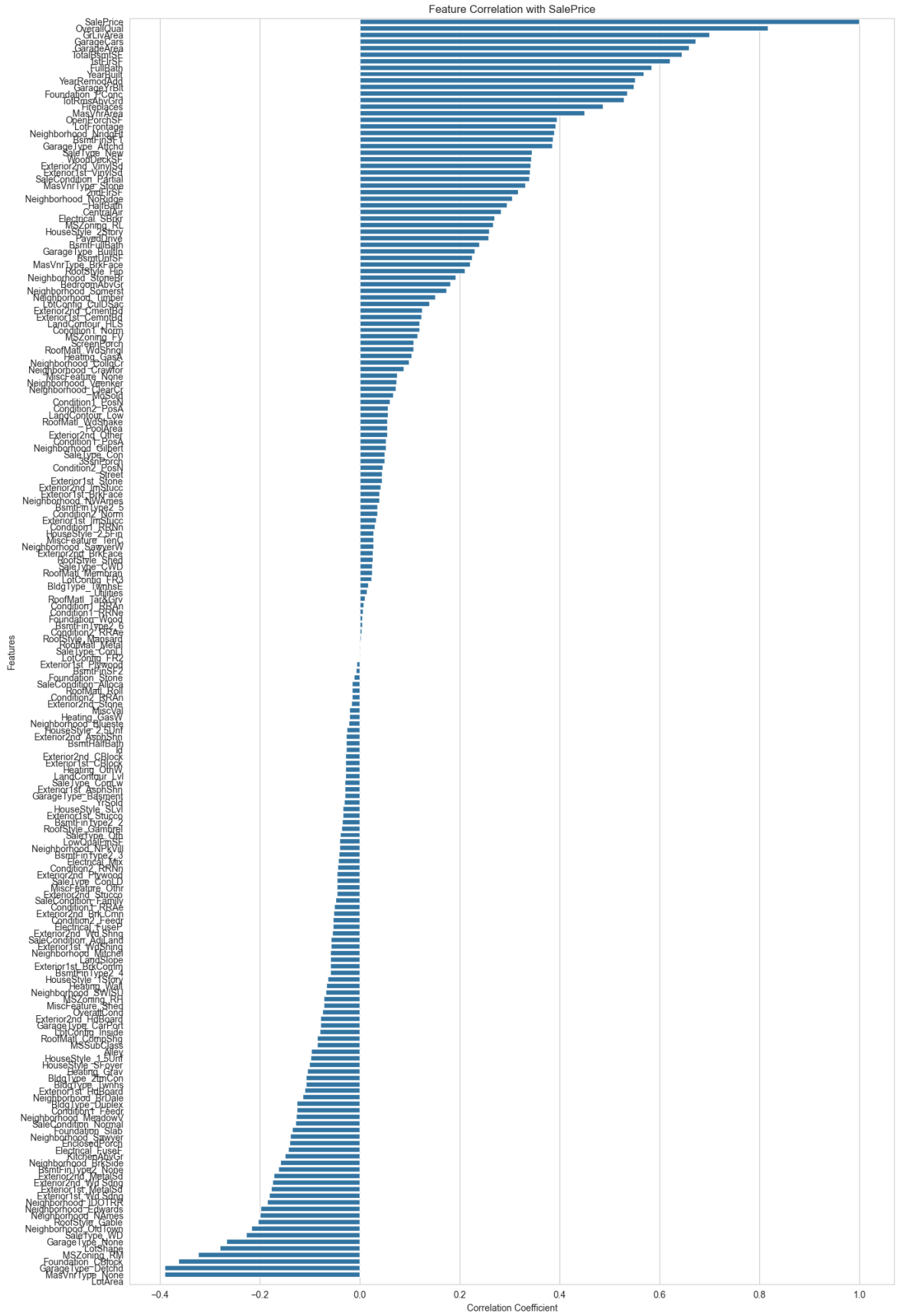
In [ ]:
```
# Drop the columns that have been encoded but not dropped
train.drop(columns=['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'BsmtExpos
                    'KitchenQual', 'Functional', 'FireplaceQu', 'GarageFinish',
                    'PoolQC', 'Fence'], inplace=True)

# Fill the NaN values in Alley column
train['Alley'].fillna(0, inplace=True)

# Check again for missing values
missing_data_final = train.isnull().sum()
missing_data_final = missing_data_final[missing_data_final > 0]

missing_data_final
```

Out[ ]:
```
Series([], dtype: int64)
```

In [ ]:
```
# Exploratory Data Analysis

# Distribution of main features
main_features = ['GrLivArea', 'TotalBsmtSF', '1stFlrSF', 'TotRmsAbvGrd', 'Garage
plt.figure(figsize=(20, 15))

for i, feature in enumerate(main_features):
    plt.subplot(3, 3, i+1)
    sns.histplot(train[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Correlation with SalePrice
correlation_with_saleprice = train.corr()['SalePrice'].sort_values(ascending=Fal
plt.figure(figsize=(15, 25))
sns.barplot(x=correlation_with_saleprice.values, y=correlation_with_saleprice.ir
plt.title('Feature Correlation with SalePrice')
plt.xlabel('Correlation Coefficient')
plt.ylabel('Features')
plt.show()
```

```
correlation_with_saleprice
```




Feature Correlation with SalePrice
Features
SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
YearBuilt
YearRemodAdd
GarageYrBlt
Foundation_PConc
TotRmsAbvGrd
Fireplaces
MasVnrArea
OpenPorchSF
LotFrontage
Neighborhood_NridgHt
BsmtFinSF1
GarageType_Attchd
SaleType_New
WoodDeckSF
Exterior2nd_VinylSd
Exterior1st_VinylSd
SaleCondition_Partial
MasVnrType_Stone
2ndFlrSF
Neighborhood_NoRidge
HalfBath
CentralAir
Electrical_SBrkr
MSZoning_RL
HouseStyle_2Story
PavedDrive
BsmtFullBath
GarageType_BuiltIn
BsmtUnfSF
MasVnrType_BrkFace
RoofStyle_Hip
Neighborhood_StoneBr
BedroomAbvGr
Neighborhood_Somerst
Neighborhood_Timber
LotConfig_CulDSac
Exterior2nd_CmentBd
Exterior1st_CemntBd
LandContour_HLS
Condition1_Norm
MSZoning_FV
ScreenPorch
RoofMatl_WdShngl
Heating_GasA
Neighborhood_BrkSide
Neighborhood_Crawfor
MiscFeature_None
Neighborhood_Veenker
Neighborhood_ClearCr
MoSold
Condition1_PosN
Condition2_PosA
LandContour_Low
RoofMatl_WdShake
PoolArea
Exterior2nd_Other
Condition1_PosA
Neighborhood_Gilbert
SaleType_Con
3SsnPorch
Condition2_PosN
Street
Exterior1st_Stone
Exterior2nd_ImStucc
Exterior1st_BrkFace
Neighborhood_NWAmes
BsmtFinType2_5
Condition2_Norm
Exterior1st_ImStucc
Condition1_RRAn
HouseStyle_2.5Fin
MiscFeature_TenC
Neighborhood_SawyerW
Exterior2nd_BrkFace
RoofStyle_Shed
SaleType_CWD
RoofMatl_Membran
LotConfig_FR3
BldgType_TwnhsE
Utilities
RoofMatl_Tar&Grv
Condition1_RRAn
Condition1_RRNe
Foundation_Wood
BsmtFinType2_6
Condition2_RRAe
RoofStyle_Mansard
RoofMatl_Metal
SaleType_ConLI
LotConfig_FR2
Exterior1st_Plywood
BsmtFinSF2
Foundation_Stone
SaleCondition_Alloca
RoofMatl_Roll
Condition2_RRAn
Exterior2nd_Stone
MiscVal
Heating_GasW
Neighborhood_Blueste
HouseStyle_2.5Unf
Exterior2nd_AsphShn
BsmtHalfBath
Id
Exterior2nd_CBlock
Exterior1st_CBlock
Heating_OthW
LandContour_Lvl
SaleType_ConLw
Exterior1st_AsphShn
GarageType_Basment
YrSold
HouseStyle_SLvl
Exterior1st_Stucco
BsmtFinType2_2
RoofStyle_Gambrel
SaleType_Oth
LowQualFinSF
Neighborhood_NPkVill
BsmtFinType2_3
Electrical_Mix
Condition2_RRNn
Exterior2nd_Plywood
SaleType_ConLD
MiscFeature_Othr
Exterior2nd_Stucco
SaleCondition_Family
Condition1_RRAe
Exterior2nd_Brk Cmn
Condition2_Feedr
Electrical_FuseP
Exterior2nd_Wd Shng
SaleCondition_AdjLand
Exterior1st_WdShing
Neighborhood_Mitchel
LandSlope
Exterior1st_BrkComm
BsmtFinType2_4
HouseStyle_1Story
Heating_Wall
Neighborhood_SWISU
MSZoning_RH
MiscFeature_Shed
OverallCond
Exterior2nd_HdBoard
GarageType_CarPort
LotConfig_Inside
RoofMatl_CompShg
MSSubClass
Alley
HouseStyle_1.5Unf
HouseStyle_SFoyer
Heating_GravA
BldgType_2fmCon
BldgType_Twnhs
Exterior1st_HdBoard
Neighborhood_BrDale
BldgType_Duplex
Condition1_Feedr
Neighborhood_MeadowV
SaleCondition_Normal
Foundation_Slab
Neighborhood_Sawyer
EnclosedPorch
Electrical_FuseF
KitchenAbvGr
Neighborhood_BrkSide
BsmtFinType2_None
Exterior2nd_MetalSd
Exterior2nd_Wd Sdng
Exterior1st_MetalSd
Exterior1st_Wd Sdng
Neighborhood_IDOTRR
Neighborhood_Edwards
Neighborhood_NAmes
RoofStyle_Gable
Neighborhood_OldTown
SaleType_WD
GarageType_None
LotShape
MSZoning_RM
Foundation_CBlock
GarageType_Detchd
MasVnrType_None
LotArea
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Correlation Coefficient

Out[ ]:
```
SalePrice            1.000000
OverallQual          0.816856
GrLivArea            0.699980
GarageCars           0.672293
GarageArea           0.660029
                         ...
MSZoning_RM         -0.323491
Foundation_CBlock   -0.362833
GarageType_Detchd   -0.389651
MasVnrType_None     -0.390085
LotArea                   NaN
Name: SalePrice, Length: 193, dtype: float64
```

In [ ]:
```
# 1. Polynomial Features
train['GrLivArea_Sq'] = train['GrLivArea'] ** 2
train['TotalBsmtSF_Sq'] = train['TotalBsmtSF'] ** 2
train['1stFlrSF_Sq'] = train['1stFlrSF'] ** 2
train['GarageArea_Sq'] = train['GarageArea'] ** 2

# 2. Log Transformation
for column in ['GrLivArea', 'TotalBsmtSF', '1stFlrSF', 'GarageArea', 'LotArea',
    train[column] = np.log1p(train[column])

# 3. Binning
# Binning YearBuilt and YearRemodAdd into decades
train['DecadeBuilt'] = (train['YearBuilt'] // 10) * 10
train['DecadeRemod'] = (train['YearRemodAdd'] // 10) * 10

# Removing original columns to avoid multicollinearity
train.drop(columns=['YearBuilt', 'YearRemodAdd'], inplace=True)

# Return the first few rows of the modified dataframe
train.head()
```

```
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:2: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['GrLivArea_Sq'] = train['GrLivArea'] ** 2
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:3: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['TotalBsmtSF_Sq'] = train['TotalBsmtSF'] ** 2
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:4: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['1stFlrSF_Sq'] = train['1stFlrSF'] ** 2
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:5: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['GarageArea_Sq'] = train['GarageArea'] ** 2
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:13: PerformanceWa
rning: DataFrame is highly fragmented.  This is usually the result of calling `fr
ame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['DecadeBuilt'] = (train['YearBuilt'] // 10) * 10
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\2439655777.py:14: PerformanceWa
rning: DataFrame is highly fragmented.  This is usually the result of calling `fr
ame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  train['DecadeRemod'] = (train['YearRemodAdd'] // 10) * 10
```

Out[ ]:

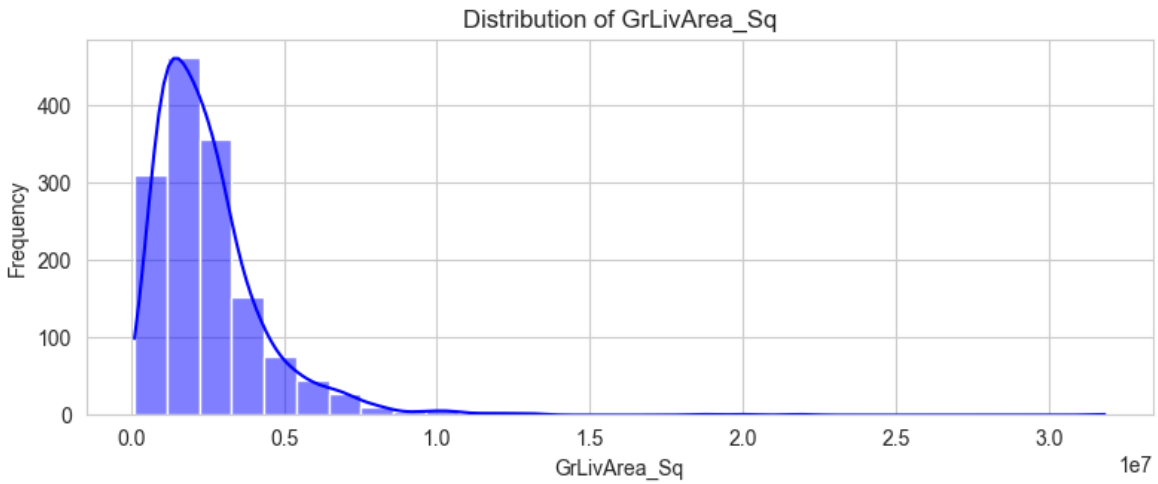
	Id	MSSubClass	LotFrontage	LotArea	Street	Alley	LotShape	Utilities	LandSlope
0	1	60	4.189655	7.607381	2	0.0	4	4	3
1	2	20	4.394449	7.607381	2	0.0	4	4	3
2	3	60	4.234107	7.607381	2	0.0	3	4	3
3	4	70	4.110874	7.607381	2	0.0	3	4	3
4	5	60	4.442651	7.607381	2	0.0	3	4	3

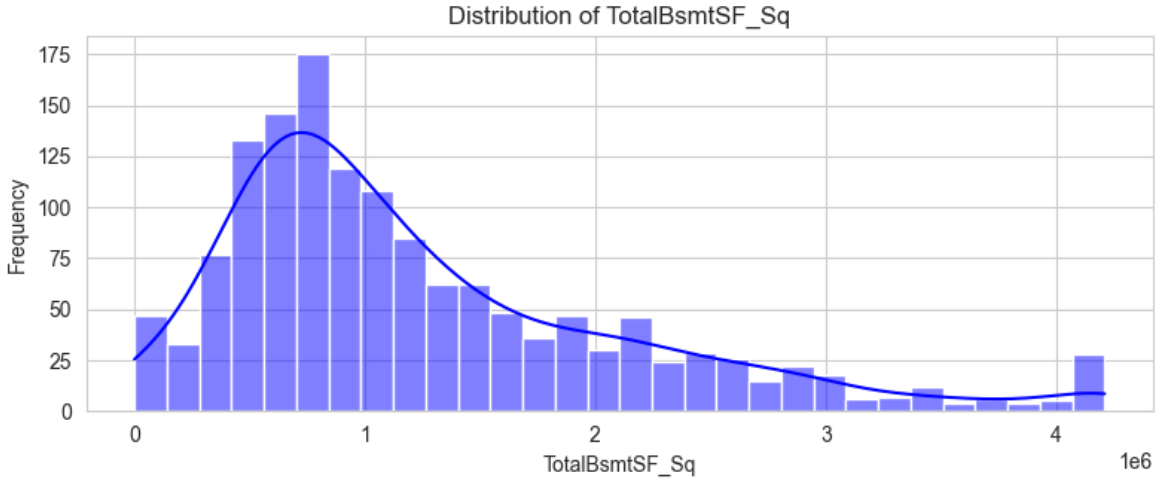
5 rows × 197 columns

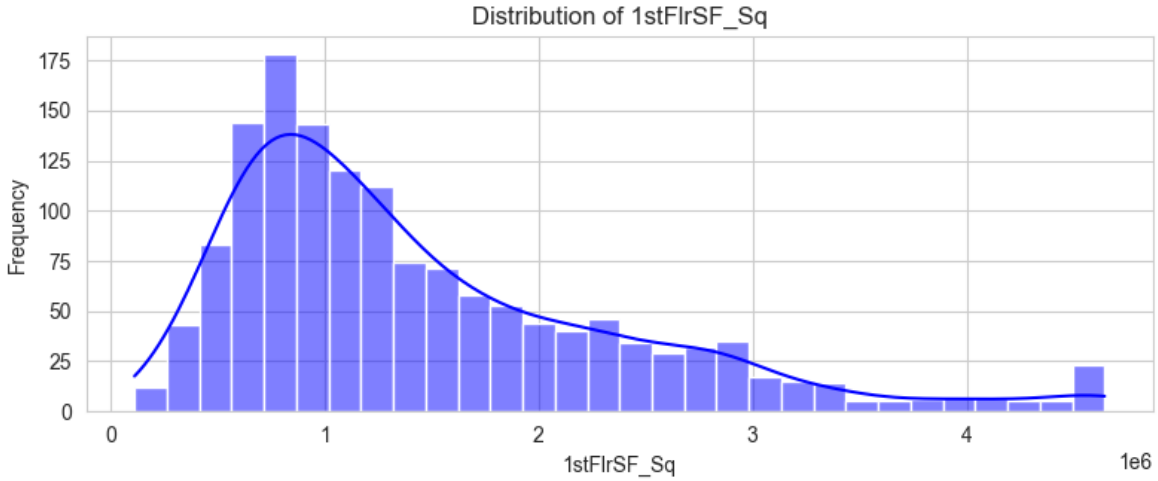
In [ ]:

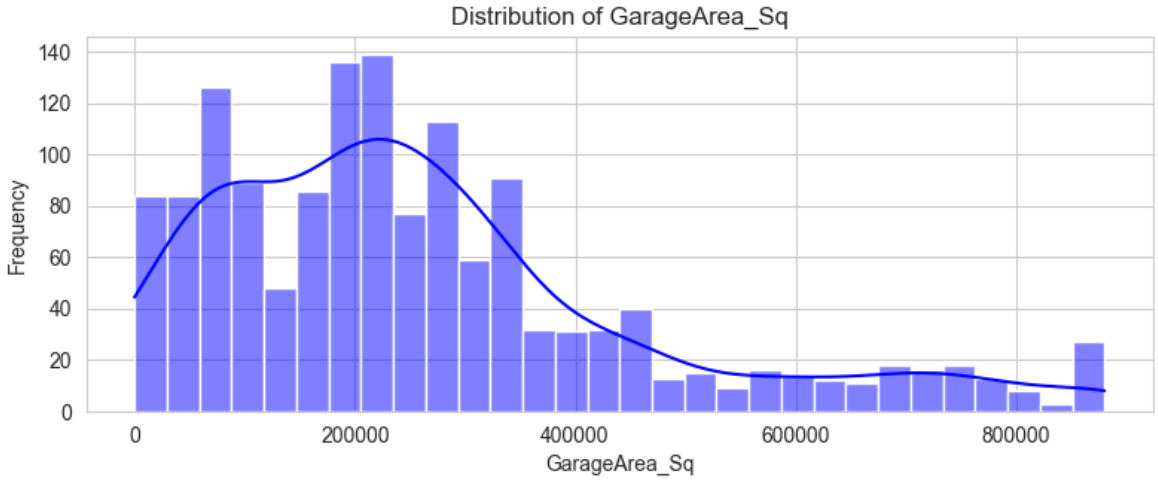
```
# Visualizing the distribution of newly created features

# Newly created features from previous feature engineering steps
new_features = ['GrLivArea_Sq', 'TotalBsmtSF_Sq', '1stFlrSF_Sq', 'GarageArea_Sq'

# Plotting distributions
fig, axes = plt.subplots(nrows=len(new_features), figsize=(8, 20))
```

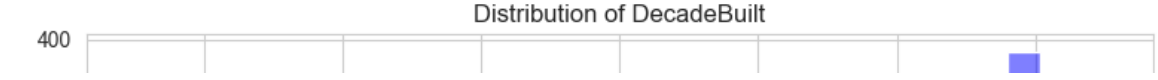
```
for i, feature in enumerate(new_features):
    sns.histplot(train[feature], ax=axes[i], bins=30, kde=True, color="blue")
    axes[i].set_title(f'Distribution of {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```

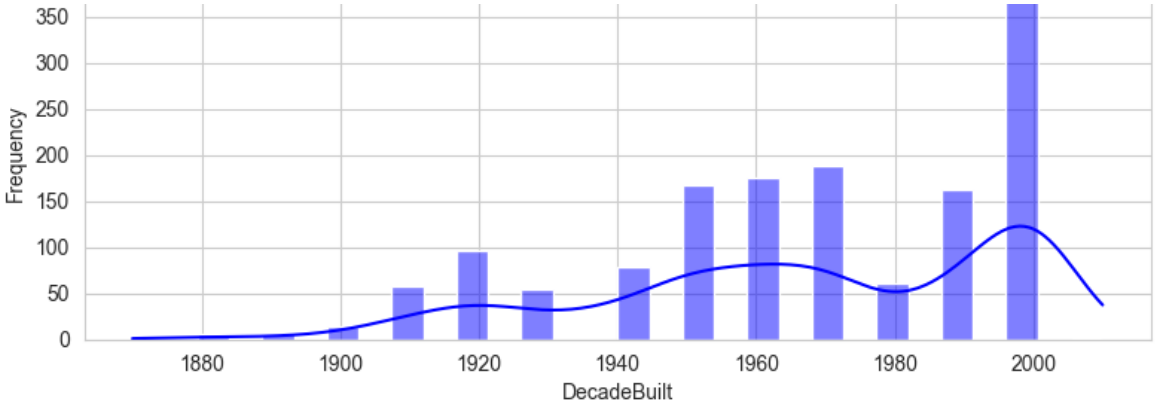

Distribution of GrLivArea_Sq
Frequency
GrLivArea_Sq
1e7

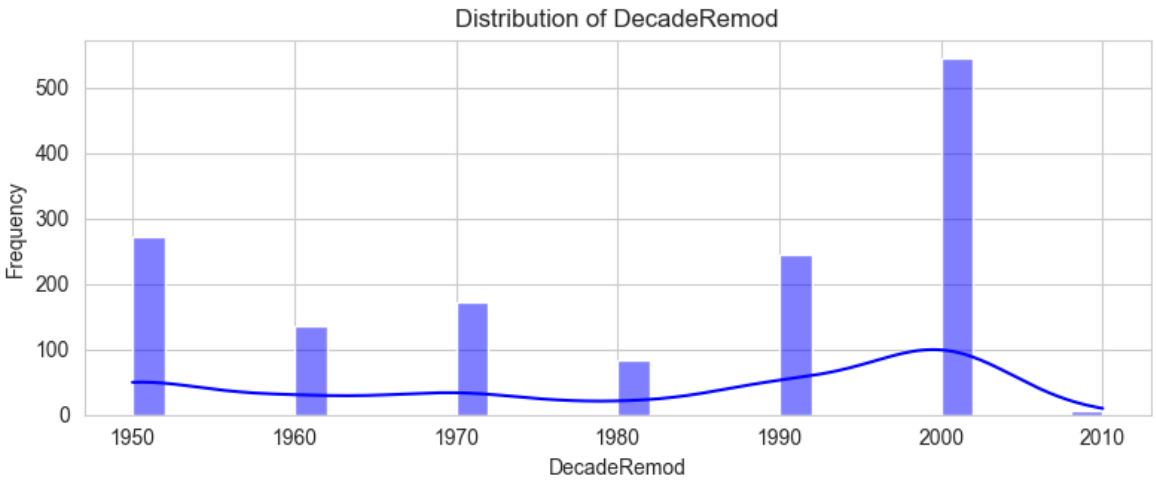

Distribution of TotalBsmtSF_Sq
Frequency
TotalBsmtSF_Sq
1e6


Distribution of 1stFlrSF_Sq
Frequency
1stFlrSF_Sq
1e6


Distribution of GarageArea_Sq
Frequency
GarageArea_Sq


Distribution of DecadeBuilt





In [ ]:
```
# Calculate correlation of new features with SalePrice
correlation_with_saleprice = train[new_features + ['SalePrice']].corr()['SalePri

correlation_with_saleprice
```

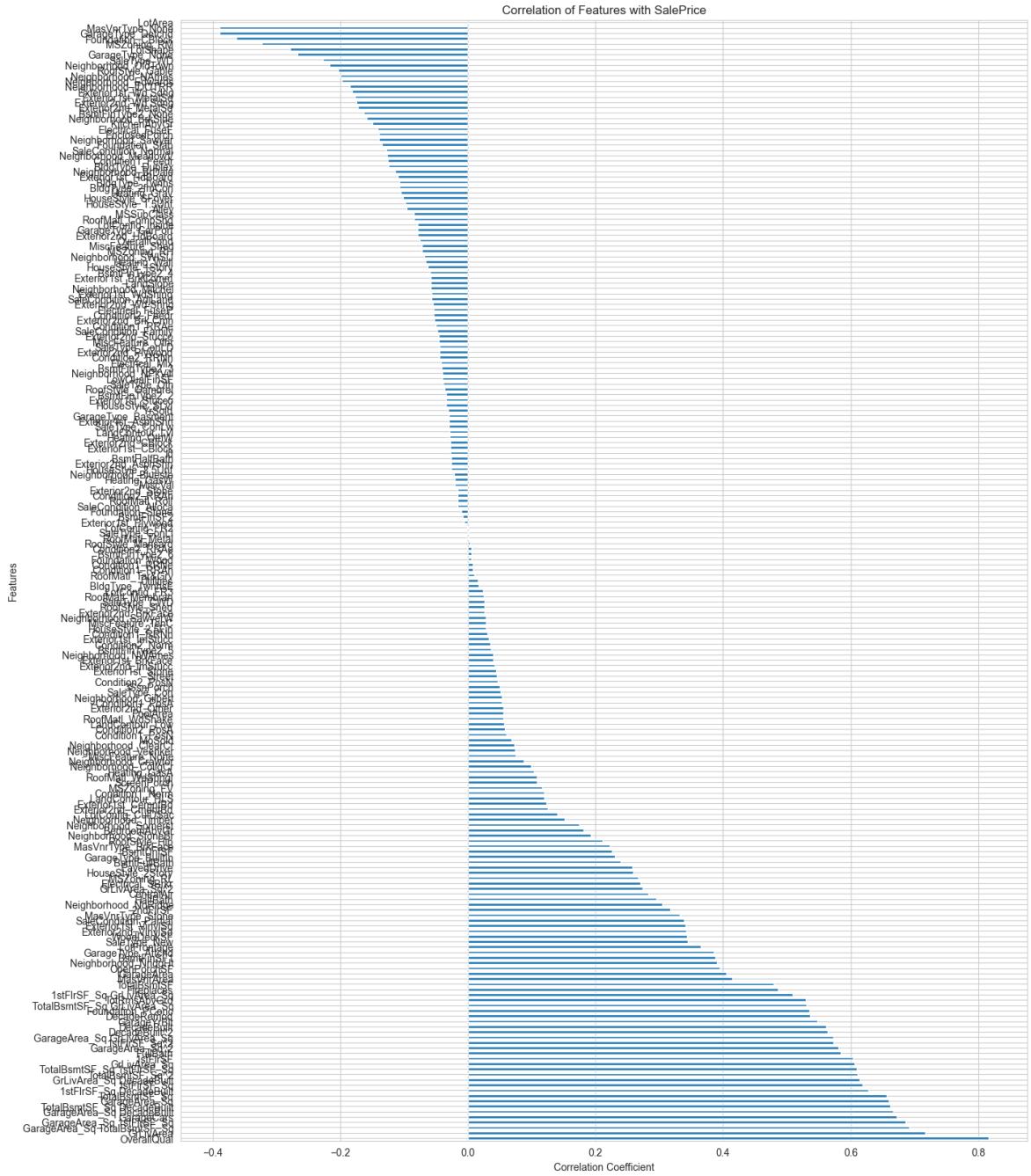
Out[ ]:
```
SalePrice         1.000000
GarageArea_Sq     0.660190
TotalBsmtSF_Sq    0.656559
1stFlrSF_Sq       0.619165
GrLivArea_Sq      0.606543
DecadeBuilt       0.561885
DecadeRemod       0.536574
Name: SalePrice, dtype: float64
```

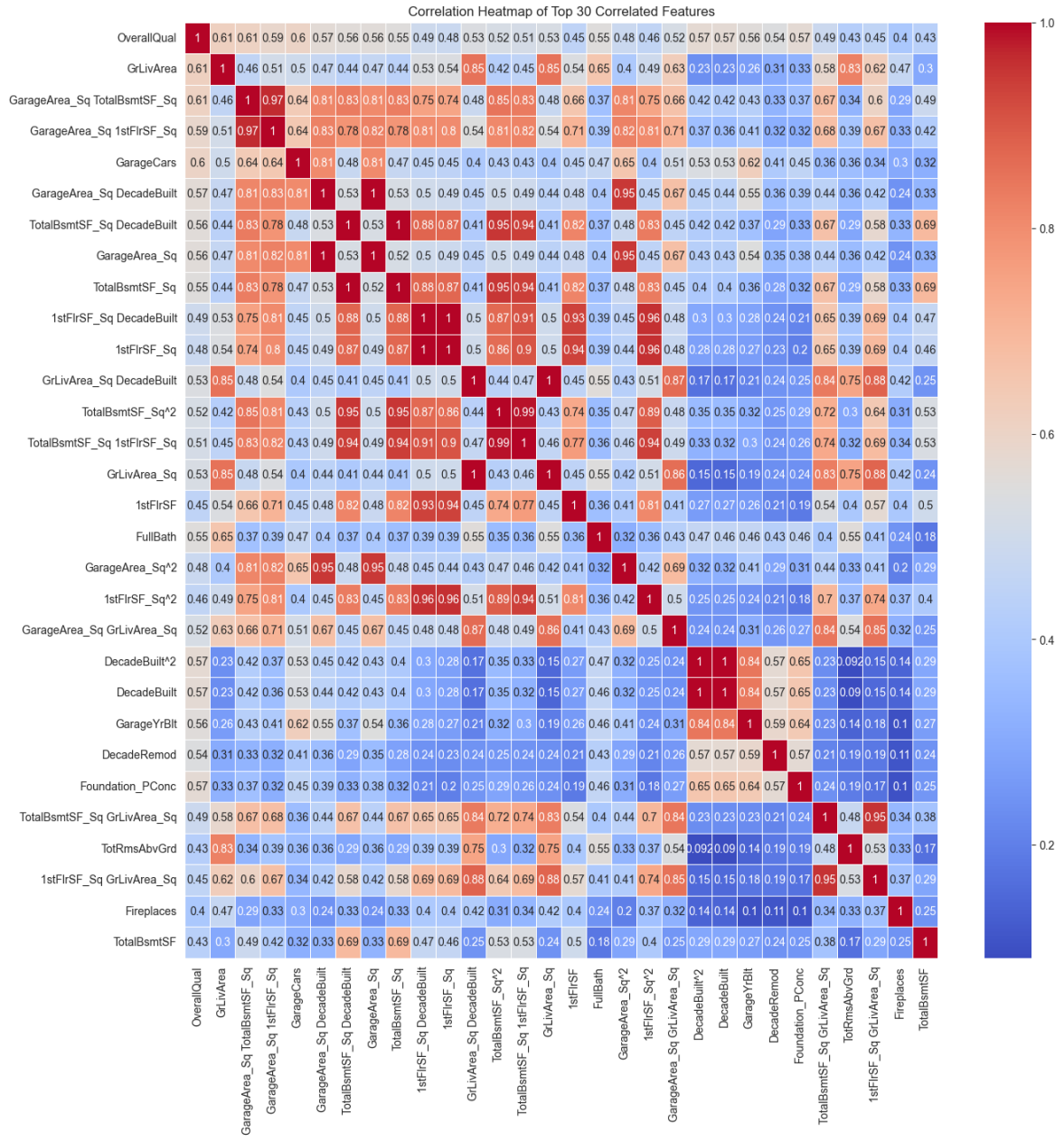
In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures

# Selecting top 5 numerical features most correlated with SalePrice
top_features = correlation_with_saleprice.abs().sort_values(ascending=False).hea

# Creating polynomial and interaction features for the top features
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=False)
poly_features = poly.fit_transform(train[top_features])
poly_feature_names = poly.get_feature_names_out(input_features=top_features)


# Creating a DataFrame for the polynomial features
poly_df = pd.DataFrame(poly_features, columns=poly_feature_names)

# Dropping the original features from the polynomial DataFrame as they already e
poly_df = poly_df.drop(columns=top_features)

# Concatenating the polynomial features to the original dataset
```

```
train_poly = pd.concat([train, poly_df], axis=1)

# Displaying the first few rows of the dataset with the polynomial features
train_poly.head()

train = train_poly
```

In [ ]:

```
# Correlation of each feature with the target variable 'SalePrice'
correlation_with_target = train.corr()['SalePrice'].sort_values(ascending=False)

# Dropping 'SalePrice' from the correlation results
correlation_with_target = correlation_with_target.drop('SalePrice')

# Visualizing the correlation using a bar plot
plt.figure(figsize=(15, 20))
correlation_with_target.plot(kind='barh')
plt.title('Correlation of Features with SalePrice')
plt.xlabel('Correlation Coefficient')
plt.ylabel('Features')
plt.show()

# Visualizing the correlation matrix using a heatmap for the top correlated feat
top_correlated_features = correlation_with_target.index[:30]  # Top 30 features
correlation_matrix = train[top_correlated_features].corr()

plt.figure(figsize=(15, 15))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Top 30 Correlated Features')
plt.show()

correlation_with_target
```

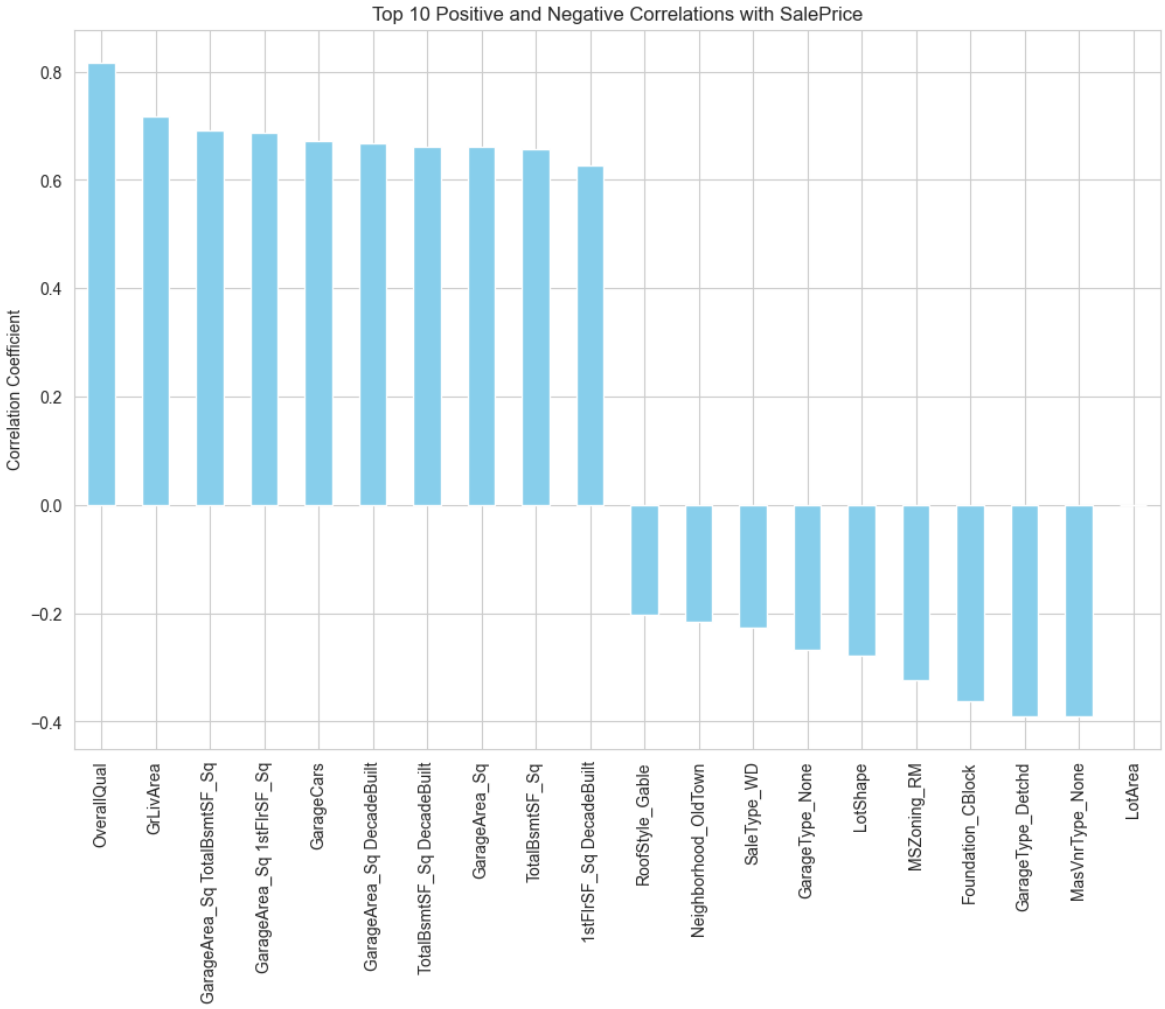

Correlation of Features with SalePrice
Features
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Correlation Coefficient



Out[ ]:
```
OverallQual                  0.816856
GrLivArea                    0.717018
GarageArea_Sq TotalBsmtSF_Sq 0.691855
GarageArea_Sq 1stFlrSF_Sq    0.686409
GarageCars                   0.672293
                                  ...
MSZoning_RM                 -0.323491
Foundation_CBlock           -0.362833
GarageType_Detchd           -0.389651
MasVnrType_None             -0.390085
LotArea                           NaN
Name: SalePrice, Length: 211, dtype: float64
```

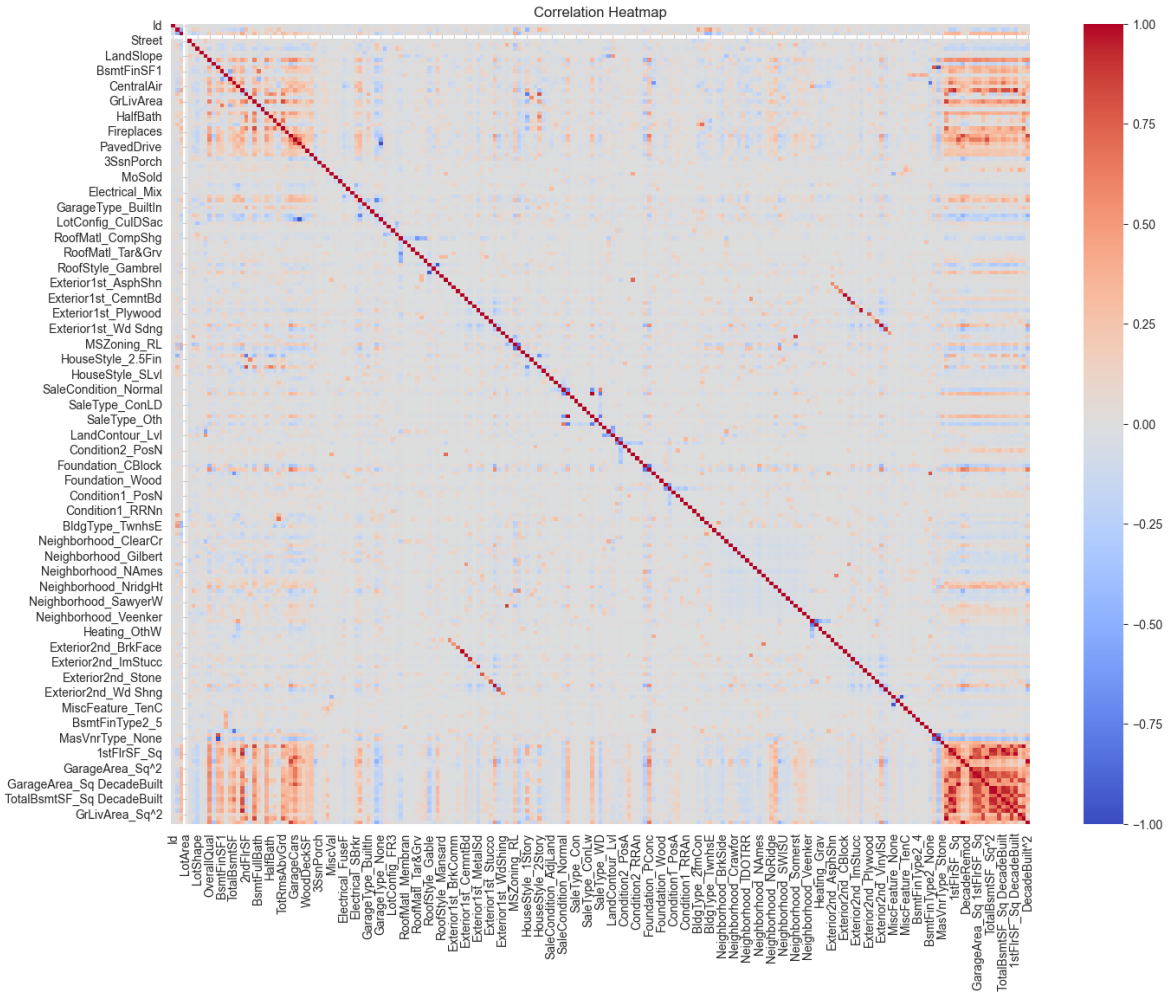
In [ ]:
```
# Calculate the correlation of each feature with 'SalePrice'
correlation = train.corr()['SalePrice'].sort_values(ascending=False)

# Drop 'SalePrice' correlation with itself
correlation = correlation.drop('SalePrice')

# Display the features with the highest correlation (positive and negative) with
top_correlation = pd.concat([correlation.head(10), correlation.tail(10)])
```

```
# Visualize the top correlated features
plt.figure(figsize=(12, 8))
top_correlation.plot(kind='bar', color='skyblue')
plt.title('Top 10 Positive and Negative Correlations with SalePrice')
plt.ylabel('Correlation Coefficient')
plt.show()

top_correlation
```



Out[ ]:
```
OverallQual                    0.816856
GrLivArea                      0.717018
GarageArea_Sq TotalBsmtSF_Sq   0.691855
GarageArea_Sq 1stFlrSF_Sq      0.686409
GarageCars                     0.672293
GarageArea_Sq DecadeBuilt      0.666924
TotalBsmtSF_Sq DecadeBuilt     0.662138
GarageArea_Sq                  0.660190
TotalBsmtSF_Sq                 0.656559
1stFlrSF_Sq DecadeBuilt        0.627642
RoofStyle_Gable               -0.203719
Neighborhood_OldTown          -0.216855
SaleType_WD                   -0.227458
GarageType_None               -0.267004
LotShape                      -0.279291
MSZoning_RM                   -0.323491
Foundation_CBlock             -0.362833
GarageType_Detchd             -0.389651
MasVnrType_None               -0.390085
LotArea                             NaN
Name: SalePrice, dtype: float64
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
corr_matrix = train.drop('SalePrice', axis=1).corr()

# Plot the heatmap
plt.figure(figsize=(16, 12))
sns.heatmap(corr_matrix, cmap='coolwarm', vmax=1, vmin=-1)
plt.title('Correlation Heatmap')
plt.show()
```



In [ ]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

# Splitting the data into features and target variable
X = train.drop(columns=['SalePrice'])
y = train['SalePrice']

# Initialize a Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)

# Get feature importances
feature_importances = rf.feature_importances_

# Creating a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# Sorting the features based on importance
sorted_features = features_df.sort_values(by='Importance', ascending=False)

sorted_features.head(20)
```

Out[ ]:

	Feature	Importance
9	OverallQual	0.404037
203	TotalBsmtSF_Sq GrLivArea_Sq	0.327353
199	GarageArea_Sq GrLivArea_Sq	0.062836
206	1stFlrSF_Sq GrLivArea_Sq	0.027414
12	BsmtFinSF1	0.010712
14	BsmtUnfSF	0.008824
209	GrLivArea_Sq DecadeBuilt	0.008817
198	GarageArea_Sq 1stFlrSF_Sq	0.007611
10	OverallCond	0.007416
197	GarageArea_Sq TotalBsmtSF_Sq	0.006632
29	GarageYrBlt	0.006407
194	DecadeBuilt	0.006103
195	DecadeRemod	0.005363
18	2ndFlrSF	0.005312
30	GarageCars	0.004719
210	DecadeBuilt^2	0.004479
0	Id	0.004343
28	Fireplaces	0.004243
2	LotFrontage	0.004088
33	WoodDeckSF	0.003713

In [ ]:
```
# Selecting the top 30 features
top_30_features = sorted_features['Feature'].head(30).tolist()

# Keeping only the top 30 features in the train dataset
train = train[top_30_features + ['SalePrice']]
```

In [ ]:
```
from sklearn.model_selection import train_test_split

# Define features and target variable
X = train.drop(columns=['SalePrice'])
y = train['SalePrice']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[ ]:
```
((1168, 30), (292, 30), (1168,), (292,))
```

In [ ]:
```
X_train.to_csv("X_train.csv")
X_test.to_csv("X_test.csv")
y_train.to_csv("y_train.csv")
y_test.to_csv("y_test.csv")
```

In [ ]:
```
import xgboost as xgb
import lightgbm as lgb
from sklearn.ensemble import AdaBoostRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Load the data
X_train = pd.read_csv("X_train.csv", index_col=0)
X_test = pd.read_csv("X_test.csv", index_col=0)
y_train = pd.read_csv("y_train.csv", index_col=0).squeeze()
y_test = pd.read_csv("y_test.csv", index_col=0).squeeze()

# Update the models dictionary to include new models
models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso": Lasso(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "XGBoost": xgb.XGBRegressor(objective='reg:squarederror'),
    "LightGBM": lgb.LGBMRegressor(),
    "AdaBoost": AdaBoostRegressor()
}

# Evaluate each model
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    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)

    results[name] = [mae, mse, rmse, r2]

# Update the results DataFrame
results_df = pd.DataFrame(results, index=["MAE", "MSE", "RMSE", "R2"]).T
results_df
```

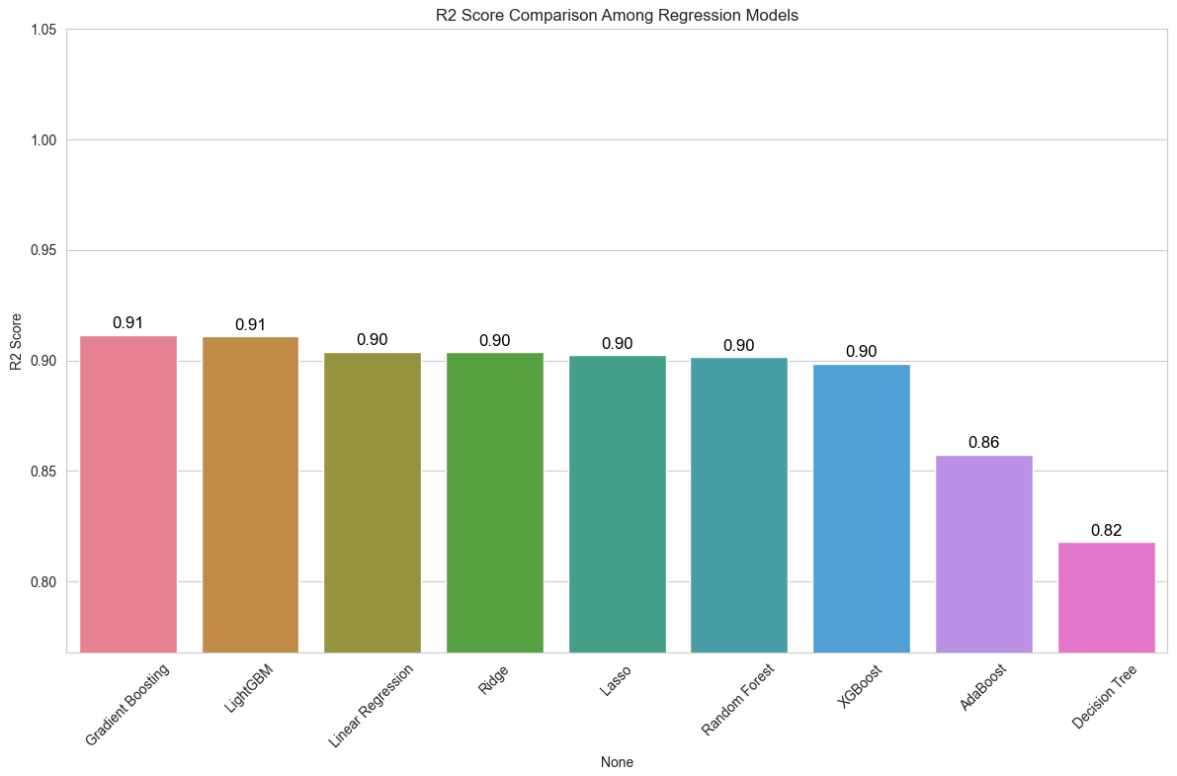
Out[ ]:

	MAE	MSE	RMSE	R2
Linear Regression	16679.046674	4.696910e+08	21672.356177	0.903934
Ridge	16577.881845	4.712328e+08	21707.896862	0.903619
Lasso	16639.285195	4.772759e+08	21846.645675	0.902383
Decision Tree	21397.592466	8.909252e+08	29848.370912	0.817779
Random Forest	15138.122346	4.817995e+08	21949.931275	0.901457
Gradient Boosting	15026.483570	4.324508e+08	20795.450401	0.911551
XGBoost	15508.458958	4.954350e+08	22258.368825	0.898669
LightGBM	14365.461180	4.362609e+08	20886.858800	0.910771
AdaBoost	20324.493233	6.974005e+08	26408.340994	0.857360

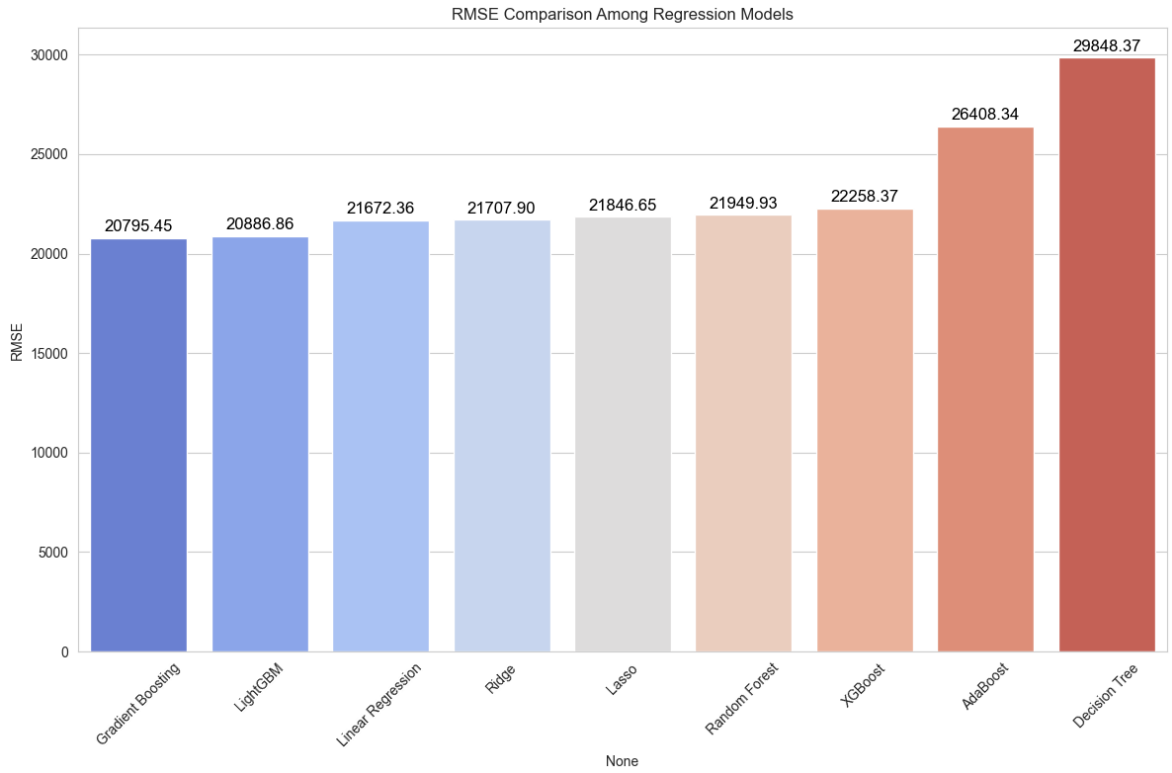
In [ ]:

```
# Sorting the models based on their R2 scores for better visualization
sorted_results = results_df.sort_values(by="R2", ascending=False)

# Function to plot the metrics
def plot_metric(dataframe, metric, title, y_label, color_palette, ylim=None):
    sorted_df = dataframe.sort_values(by=metric, ascending=(metric != "R2"))  #
    plt.figure(figsize=(12, 8))
    colors = sns.color_palette(color_palette, len(sorted_df))  # Generate a list
    bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)

    # Annotate bars
    for p in bars.patches:
        bars.annotate(f"{p.get_height():.2f}",
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      fontsize=12, color='black',
                      xytext=(0, 8),
                      textcoords='offset points')

    plt.ylabel(y_label)
    plt.title(title)
    plt.xticks(rotation=45)
    if ylim:
        plt.ylim(ylim)
    plt.tight_layout()
    plt.show()

# Plotting R2 scores
plot_metric(sorted_results, "R2", "R2 Score Comparison Among Regression Models",
            "R2 Score", "husl", [min(sorted_results['R2']) - 0.05, 1.05])

# Plotting RMSE
plot_metric(sorted_results, "RMSE", "RMSE Comparison Among Regression Models",
            "RMSE", "coolwarm")
```

```
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\1006554028.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)
```




```
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\1006554028.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)
```



In [ ]:
```
from sklearn.ensemble import StackingRegressor
from sklearn.model_selection import RandomizedSearchCV

# 1. Stacking

# Defining the base models
base_models = [
    ('Random Forest', RandomForestRegressor()),
    ('Gradient Boosting', GradientBoostingRegressor()),
    ('XGBoost', xgb.XGBRegressor(objective='reg:squarederror')),
    ('LightGBM', lgb.LGBMRegressor())
]

# Initializing the Stacking Regressor
stacked_model = StackingRegressor(estimators=base_models, final_estimator=Linear

# Training the Stacking Regressor
stacked_model.fit(X_train, y_train)

# Predicting with Stacking Regressor
y_pred_stacked = stacked_model.predict(X_test)

# Evaluation Metrics for Stacking Regressor
mae_stacked = mean_absolute_error(y_test, y_pred_stacked)
mse_stacked = mean_squared_error(y_test, y_pred_stacked)
rmse_stacked = np.sqrt(mse_stacked)
r2_stacked = r2_score(y_test, y_pred_stacked)

stacked_results = {"Stacking": [mae_stacked, mse_stacked, rmse_stacked, r2_stack

# 2. Blending (Using Linear Regression as the meta-model)

# Splitting the training data further into two subsets for blending
X_train_blend, X_val_blend, y_train_blend, y_val_blend = train_test_split(X_trai

# Creating an empty DataFrame for storing blending features
```

```
blend_features = pd.DataFrame()

# Training base models and collecting their predictions on validation set
for name, model in base_models:
    model.fit(X_train_blend, y_train_blend)
    blend_features[name] = model.predict(X_val_blend)

# Training the meta-model on top of the base model predictions
meta_model = LinearRegression().fit(blend_features, y_val_blend)

# Preparing test data for blending (using predictions from base models as featur
blend_test_features = pd.DataFrame()
for name, model in base_models:
    blend_test_features[name] = model.predict(X_test)

# Predicting with the meta-model
y_pred_blend = meta_model.predict(blend_test_features)

# Evaluation Metrics for Blending
mae_blend = mean_absolute_error(y_test, y_pred_blend)
mse_blend = mean_squared_error(y_test, y_pred_blend)
rmse_blend = np.sqrt(mse_blend)
r2_blend = r2_score(y_test, y_pred_blend)

blend_results = {"Blending": [mae_blend, mse_blend, rmse_blend, r2_blend]}

# 3. Hyperparameter Tuning

# Parameters for Randomized Search
xgb_params = {
    'learning_rate': [0.01, 0.05, 0.1, 0.5],
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_child_weight': [1, 2, 3, 4],
    'n_estimators': [100, 200, 300, 400, 500],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

gb_params = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.5],
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3],
    'subsample': [0.8, 0.9, 1.0]
}

lgb_params = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.5],
    'max_depth': [3, 4, 5, 6, 7, 8],
    'num_leaves': [31, 62, 93, 124, 155],
    'min_child_samples': [10, 20, 30],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

# XGBoost
xgb_search = RandomizedSearchCV(xgb.XGBRegressor(objective='reg:squarederror'),
```

```
xgb_search.fit(X_train, y_train)

# Gradient Boosting
gb_search = RandomizedSearchCV(GradientBoostingRegressor(), gb_params, n_iter=10
gb_search.fit(X_train, y_train)

# LightGBM
lgb_search = RandomizedSearchCV(lgb.LGBMRegressor(), lgb_params, n_iter=10, scor
lgb_search.fit(X_train, y_train)

# Best models from Randomized Search
best_xgb = xgb_search.best_estimator_
best_gb = gb_search.best_estimator_
best_lgb = lgb_search.best_estimator_

# Training and Predicting with the best models
best_xgb.fit(X_train, y_train)
y_pred_xgb_tuned = best_xgb.predict(X_test)

best_gb.fit(X_train, y_train)
y_pred_gb_tuned = best_gb.predict(X_test)

best_lgb.fit(X_train, y_train)
y_pred_lgb_tuned = best_lgb.predict(X_test)

# Evaluation Metrics for the best models
mae_xgb_tuned = mean_absolute_error(y_test, y_pred_xgb_tuned)
mse_xgb_tuned = mean_squared_error(y_test, y_pred_xgb_tuned)
rmse_xgb_tuned = np.sqrt(mse_xgb_tuned)
r2_xgb_tuned = r2_score(y_test, y_pred_xgb_tuned)

mae_gb_tuned = mean_absolute_error(y_test, y_pred_gb_tuned)
mse_gb_tuned = mean_squared_error(y_test, y_pred_gb_tuned)
rmse_gb_tuned = np.sqrt(mse_gb_tuned)
r2_gb_tuned = r2_score(y_test, y_pred_gb_tuned)

mae_lgb_tuned = mean_absolute_error(y_test, y_pred_lgb_tuned)
mse_lgb_tuned = mean_squared_error(y_test, y_pred_lgb_tuned)
rmse_lgb_tuned = np.sqrt(mse_lgb_tuned)
r2_lgb_tuned = r2_score(y_test, y_pred_lgb_tuned)

# Results
xgb_tuned_results = {"XGBoost Tuned": [mae_xgb_tuned, mse_xgb_tuned, rmse_xgb_tu
gb_tuned_results = {"Gradient Boosting Tuned": [mae_gb_tuned, mse_gb_tuned, rmse
lgb_tuned_results = {"LightGBM Tuned": [mae_lgb_tuned, mse_lgb_tuned, rmse_lgb_t

# Combining all results
all_results = {**results, **stacked_results, **blend_results, **xgb_tuned_result

# Updating the results DataFrame
results_df_updated = pd.DataFrame(all_results, index=["MAE", "MSE", "RMSE", "R2"
results_df_updated
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[ ]:

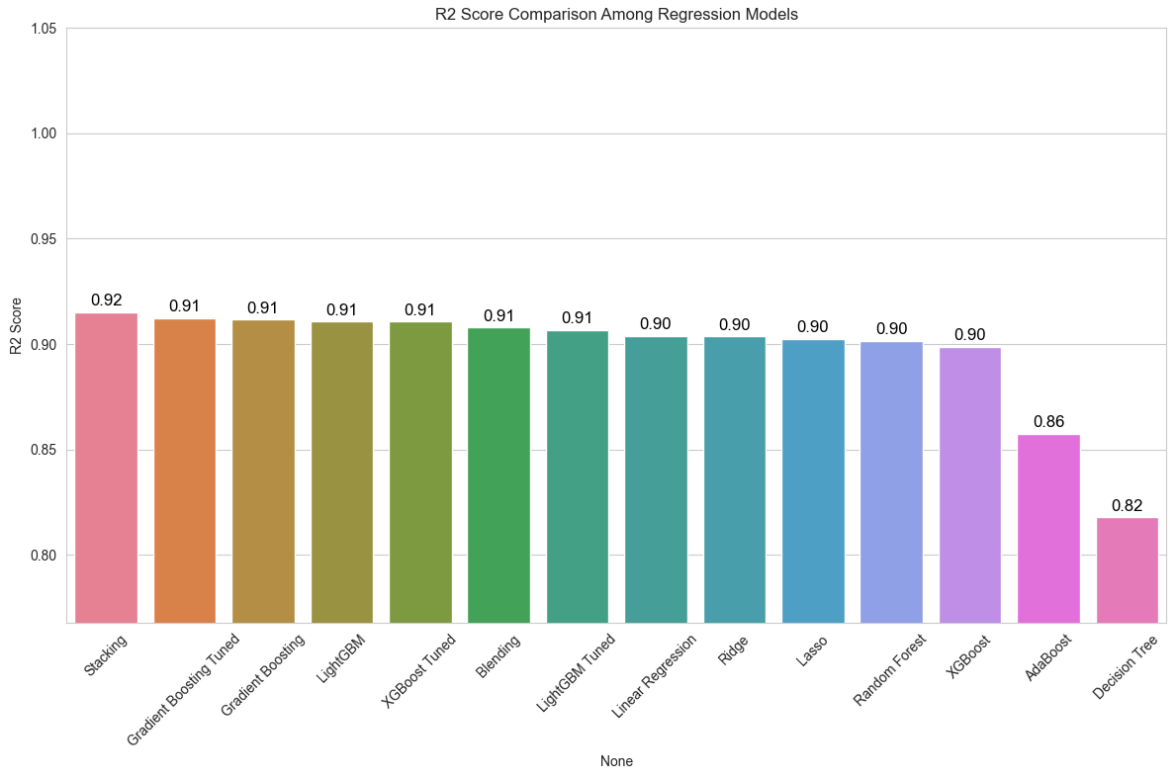
	MAE	MSE	RMSE	R2
Linear Regression	16679.046674	4.696910e+08	21672.356177	0.903934
Ridge	16577.881845	4.712328e+08	21707.896862	0.903619
Lasso	16639.285195	4.772759e+08	21846.645675	0.902383
Decision Tree	21397.592466	8.909252e+08	29848.370912	0.817779
Random Forest	15138.122346	4.817995e+08	21949.931275	0.901457
Gradient Boosting	15026.483570	4.324508e+08	20795.450401	0.911551
XGBoost	15508.458958	4.954350e+08	22258.368825	0.898669
LightGBM	14365.461180	4.362609e+08	20886.858800	0.910771
AdaBoost	20324.493233	6.974005e+08	26408.340994	0.857360
Stacking	14517.123290	4.155215e+08	20384.344254	0.915013
Blending	15055.665205	4.504844e+08	21224.618111	0.907862
XGBoost Tuned	14598.349957	4.362749e+08	20887.193923	0.910769
Gradient Boosting Tuned	14146.283873	4.291380e+08	20715.646766	0.912228
LightGBM Tuned	14657.862140	4.558525e+08	21350.702587	0.906764

In [ ]:

```
# Sorting the models based on their R2 scores for better visualization
sorted_results = results_df_updated.sort_values(by="R2", ascending=False)

# Function to plot the metrics
```

```
def plot_metric(dataframe, metric, title, y_label, color_palette, ylim=None):
    sorted_df = dataframe.sort_values(by=metric, ascending=(metric != "R2"))  #
    plt.figure(figsize=(12, 8))
    colors = sns.color_palette(color_palette, len(sorted_df))  # Generate a list
    bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)

    # Annotate bars
    for p in bars.patches:
        bars.annotate(f"{p.get_height():.2f}",
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      fontsize=12, color='black',
                      xytext=(0, 8),
                      textcoords='offset points')

    plt.ylabel(y_label)
    plt.title(title)
    plt.xticks(rotation=45)
    if ylim:
        plt.ylim(ylim)
    plt.tight_layout()
    plt.show()

# Plotting R2 scores
plot_metric(sorted_results, "R2", "R2 Score Comparison Among Regression Models",
            "R2 Score", "husl", [min(sorted_results['R2']) - 0.05, 1.05])

# Plotting RMSE
plot_metric(sorted_results, "RMSE", "RMSE Comparison Among Regression Models",
            "RMSE", "coolwarm")
```

```
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\739825395.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)
```




```
C:\Users\asad1\AppData\Local\Temp\ipykernel_28404\739825395.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  bars = sns.barplot(x=sorted_df.index, y=sorted_df[metric], palette=colors)
```