Module 2: Housing Prices Assignment

Introduction

Understanding the factors that influence home prices is crucial for various stakeholders, including buyers, sellers, and professionals in the real estate industry. We can gain insights into these factors through statistical methods like regression analysis. Choosing the right approach to fit a regression model depends on prior research, data characteristics, and avoiding violations of model assumptions. Developing a well-tuned predictive model helps provide accurate and precise estimates of home prices, which can be highly valuable to stakeholders. Therefore, we built a predictive model to identify key drivers of home prices.

Method

We analyzed home sale prices in Ames, Iowa, using data from Kaggle and performed the analysis with Jupyter Notebooks. Our initial step was exploratory data analysis to understand the characteristics of the data, including associations, distributions, missing values, and outliers. We used both bivariate (e.g., Pearson correlations, t-tests, ANOVAs, simple linear regressions) and multivariate analyses (e.g., OLS and Ridge regression) to examine the data. We also implemented k-fold cross-validation, training/validation sets, and considered various model components like polynomial, indicator, and dichotomous predictors.

Results and Insights

The sale price data were first visually inspected using histograms and boxplots and summarized with descriptive statistics. The sale prices ranged from \$34,900 to \$755,000, with a right-skewed distribution. We performed a log transformation to improve normality.

After addressing missing data and outliers, we identified features significantly associated with sale prices, including central air, exterior quality, and total square footage. Simple linear regressions indicated a strong relationship between these features and sale prices.

We further explored polynomial and multiple linear regression models. Adding the years since the last remodel as a predictor improved the model's explanatory power, accounting for over three-quarters of the variance. Piecewise regression models also suggested accurate predictions, with an RMSE of 0.22 and an R^2 of 0.82.

We utilized principal component analysis (PCA) to reduce multicollinearity and included scaled log sale prices in our models. Our polynomial PCA regression achieved an R^2 of 0.90 with cross-validation, indicating a strong fit.

Regularization techniques like Ridge and Elastic Net regression identified important predictors, such as overall quality, total basement square footage, and garage characteristics. Our final models incorporated these predictors and encoded categorical variables, resulting in robust models.

We tested our models on the test dataset and submitted predictions to Kaggle, achieving RMSE scores of 0.16430 and 0.17007 for polynomial PCA and Ridge PCA regression models, respectively.

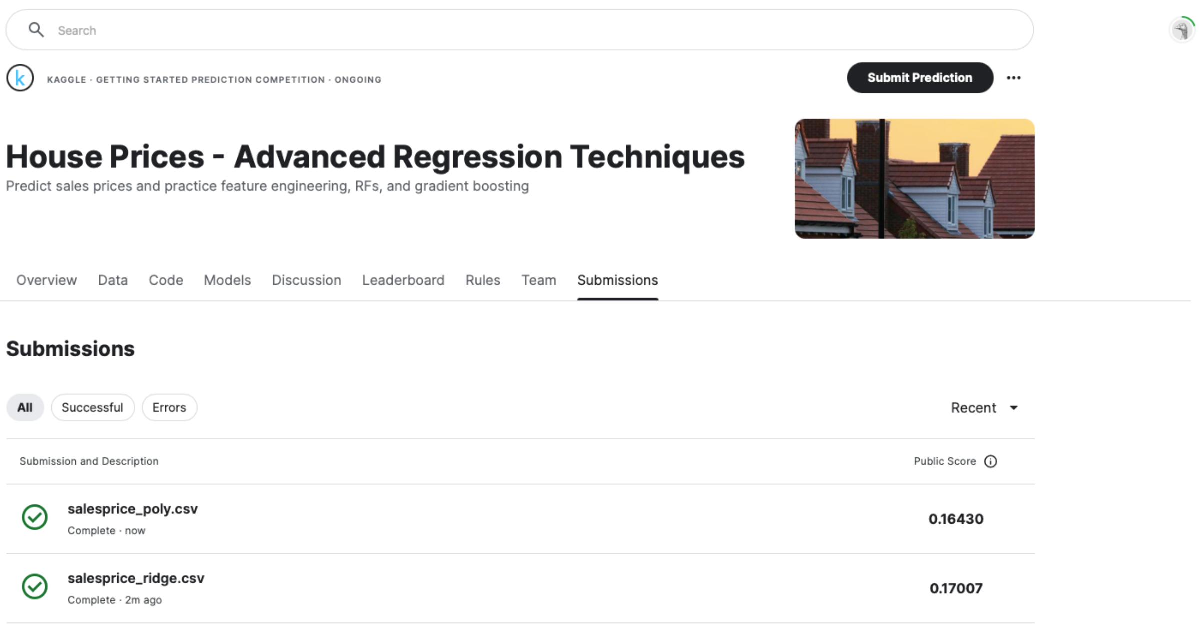
Exposition, Problem Description, and Management Recommendations

Understanding the myriad factors influencing home prices in Ames, Iowa, is essential for making informed decisions in the real estate market. This study identified key predictors like total square footage, exterior quality, and the presence of central air that significantly affect home prices. The primary challenge lies in accurately quantifying these relationships while addressing issues like multicollinearity, non-linearity, and outliers. Advanced techniques such as PCA, polynomial regression, and regularization were employed to build robust models. For practical application, stakeholders are advised to leverage these predictive models, focus on enhancing key property features, and regularly update the models with new data to ensure continued accuracy. This approach will provide a competitive edge in the market, allowing for precise and informed pricing strategies.

Kaggle Username:

sachinsharma03

Kaggle Submission Screenshots and Scores:



Appendix – Python code and outputs:

Module 2 Assignment 1: House Prices (Kaggle)

Sachin Sharma

MSDS 422

30 June 2024

Import Modules

```
import os
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import norm, kurtosis, ttest_ind, f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import statsmodels.api as sm
import pwlf
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from IPython.core.interactiveshell import InteractiveShell
from sklearn.model_selection import train_test_split, KFold,
cross_val_score, GridSearchCV
from numpy import mean, absolute, sqrt
from sklearn.linear_model import Ridge, ElasticNet, LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale, StandardScaler,
PolynomialFeatures, LabelEncoder, PolynomialFeatures
import warnings
# Ignore all FutureWarnings
warnings.filterwarnings("ignore", category=FutureWarning)
InteractiveShell.ast_node_interactivity = "all"
```

Data Preparation

```
housing_training_data = pd.read_csv('train.csv')
housing_training_data.shape
housing_training_data.head()
```

```
(1460, 81)
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
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


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

Details on Dependent Variable

We can start analyzing the distribution of the dataset's dependent variable, sale price, by generating summary statistics.

```
housing_training_data['SalePrice'].describe()

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

plt.figure(figsize=(10, 6))
sns.boxplot(x=housing_training_data["SalePrice"], palette="pastel")
plt.title('Distribution of Sale Prices', fontsize=16)
```

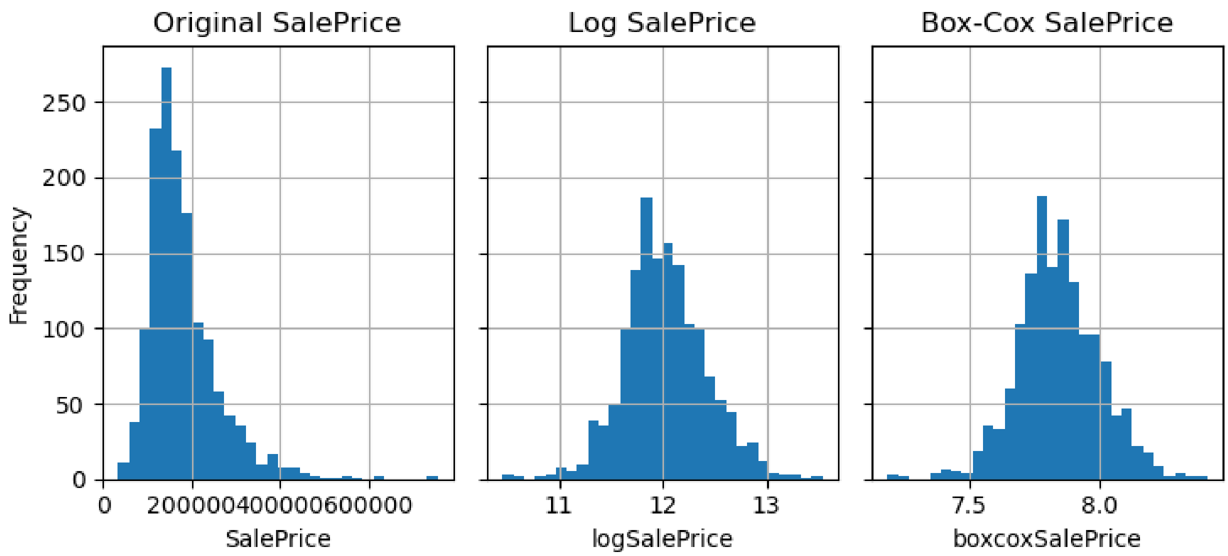
```
plt.xlabel('Sale Price', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.xticks(rotation=45)
plt.grid(True)
plt.show();
```

Below code extracts the sale price data and applies log and Box-Cox transformations. It then calculates the kurtosis for each distribution and plots histograms of the original, log-transformed, and Box-Cox transformed sale prices, ensuring the columns being plotted are numerical.

```
# Extract sale price data
raw_data = housing_training_data['SalePrice']
log_transformed_data = np.log(raw_data)
boxcox_transformed_data, best_lambda = stats.boxcox(raw_data)

# Calculate and print kurtosis
print("Home price kurtosis:", kurtosis(raw_data))
print("Log-transformed home price kurtosis:",
kurtosis(log_transformed_data))
print("Box-Cox transformed home price kurtosis:",
kurtosis(boxcox_transformed_data))

# Set plot parameters
```

```python
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True

# Create dataframes for plotting
s1 = pd.DataFrame({'SalePrice': raw_data})
s2 = pd.DataFrame({'logSalePrice': log_transformed_data})
s3 = pd.DataFrame({'boxcoxSalePrice': boxcox_transformed_data})

# Plot histograms
fig, axes = plt.subplots(1, 3, sharey=True);
s1['SalePrice'].hist(ax=axes[0], bins=30, range=(raw_data.min(),
raw_data.max()));
s2['logSalePrice'].hist(ax=axes[1], bins=30,
range=(log_transformed_data.min(), log_transformed_data.max()));
s3['boxcoxSalePrice'].hist(ax=axes[2], bins=30,
range=(boxcox_transformed_data.min(), boxcox_transformed_data.max()));

# Set titles and labels
axes[0].set_title('Original SalePrice');
axes[0].set_xlabel('SalePrice');
axes[0].set_ylabel('Frequency');

axes[1].set_title('Log SalePrice');
axes[1].set_xlabel('logSalePrice');

axes[2].set_title('Box-Cox SalePrice');
axes[2].set_xlabel('boxcoxSalePrice');

plt.show();
```

```
Home price kurtosis: 6.509812011089439
Log-transformed home price kurtosis: 0.8026555069117713
Box-Cox transformed home price kurtosis: 0.870759906431624
```



Check for Missing Data and Outliers

```
# Calculate null counts, percentage of null values, and column types
null_count = housing_training_data.isnull().sum()
null_percentage = (null_count * 100) / len(housing_training_data)
column_type = housing_training_data.dtypes

# Combine the null counts, percentage, and column types into a summary
DataFrame
null_summary = pd.concat([null_count, null_percentage, column_type],
axis=1, keys=['Missing Count', 'Percentage Missing', 'Column Type'])

# Filter the summary to show only columns with missing values, sorted
by percentage of missing values
null_summary_with_missing = null_summary[null_count >
0].sort_values('Percentage Missing', ascending=False)

# Display the summary of columns with missing values
null_summary_with_missing
```

	Missing Count	Percentage Missing	Column Type
PoolQC	1453	99.520548	object
MiscFeature	1406	96.301370	object
Alley	1369	93.767123	object
Fence	1179	80.753425	object
MasVnrType	872	59.726027	object
FireplaceQu	690	47.260274	object
LotFrontage	259	17.739726	float64
GarageType	81	5.547945	object
GarageYrBlt	81	5.547945	float64
GarageFinish	81	5.547945	object
GarageQual	81	5.547945	object
GarageCond	81	5.547945	object

```
BsmtFinType2                  38              2.602740      object
BsmtExposure                  38              2.602740      object
BsmtFinType1                  37              2.534247      object
BsmtCond                      37              2.534247      object
BsmtQual                      37              2.534247      object
MasVnrArea                     8              0.547945     float64
Electrical                     1              0.068493      object
```

We will address columns containing missing values in our exploratory data analysis by leveraging the percentage of null values, column types, and other available data columns that may offer insights useful for imputation.

```
# Drop columns with over 50% missing values: Alley, PoolQC, Fence,
MiscFeature
housing_training_data.drop(['Alley', 'PoolQC', 'Fence',
'MiscFeature'], axis=1, inplace=True)
housing_training_data.shape

(1460, 77)
```

We will assign null values

```
# List of non-numeric columns with more than one null value
columns_None = ['BsmtQual', 'BsmtCond', 'BsmtExposure',
'BsmtFinType1',
                'BsmtFinType2', 'GarageType', 'GarageFinish',
'GarageQual',
                'FireplaceQu', 'GarageCond', 'MasVnrType',
'Electrical']

# Fill null values in non-numeric columns with 'None'
housing_training_data[columns_None] =
housing_training_data[columns_None].fillna('None')

# Display the modified DataFrame or perform further operations
housing_training_data.head()

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

```
  Utilities LotConfig  ... EnclosedPorch 3SsnPorch ScreenPorch
PoolArea  \
0    AllPub    Inside  ...             0         0           0
0
1    AllPub       FR2  ...             0         0           0
0
2    AllPub    Inside  ...             0         0           0
0
3    AllPub    Corner  ...           272         0           0
0
4    AllPub       FR2  ...             0         0           0
0

  MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
0       0      2    2008        WD         Normal     208500
1       0      5    2007        WD         Normal     181500
2       0      9    2008        WD         Normal     223500
3       0      2    2006        WD        Abnorml     140000
4       0     12    2008        WD         Normal     250000

[5 rows x 77 columns]
```

We establish the optimal null-handling approach for each numeric column. Null values in the Masonry veneer area are replaced with 0, nulls in Lot Frontage with the median, and nulls in Year Garage was built with the average of the year the garage was built and the year the house was built.

```
housing_training_data['MasVnrArea'].fillna(0, inplace=True)
lot_frontage_median = housing_training_data['LotFrontage'].median()
housing_training_data['LotFrontage'].fillna(lot_frontage_median,
inplace=True)
housing_training_data['GarageYrBlt'].fillna((housing_training_data['Ye
arBuilt'] + housing_training_data['YearRemodAdd']) / 2, inplace=True)
```

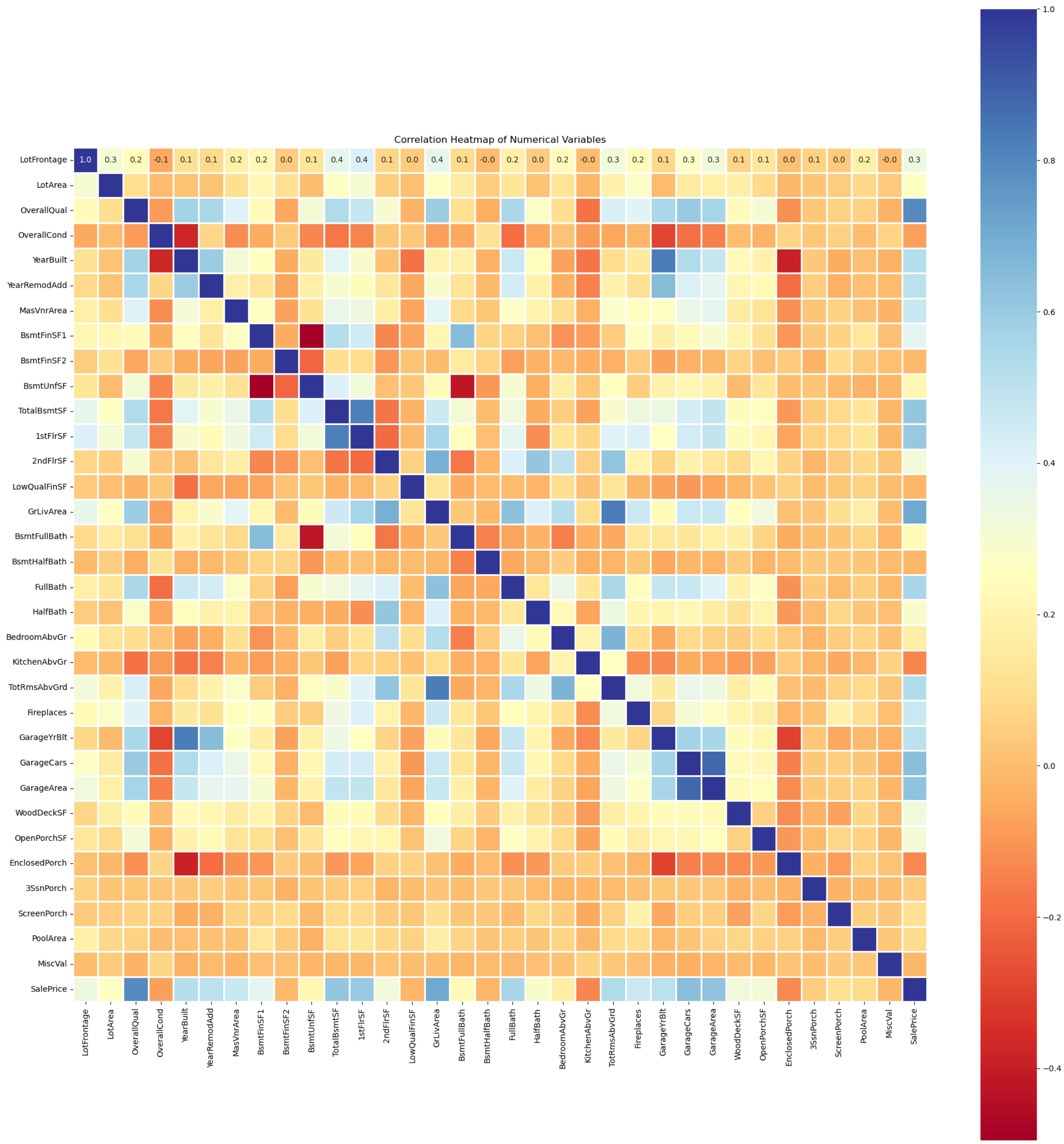
Let's check if there are any null values available

```
# check if there are no more missing values in the dataframe
null_count = housing_training_data.isnull().sum()
null_count[null_count != 0]

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

Heat Map between dependent and potential Predictor

```
# Select numerical variables
numerical_vars = ['LotFrontage', 'LotArea', 'OverallQual',
'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
                  'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
```

```
'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
                  'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                  'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
                  'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
'PoolArea', 'MiscVal', 'SalePrice']

# Create correlation matrix
df_corr_housing_training = housing_training_data[numerical_vars]
corrmat_housing_training = df_corr_housing_training.corr()

# Plot heatmap
plt.figure(figsize=(20, 20))
sns.heatmap(corrmat_housing_training, vmax=1, square=True, annot=True,
cmap='RdYlBu', linewidths=0.8, fmt=".1f")
plt.title('Correlation Heatmap of Numerical Variables')
plt.show();
```



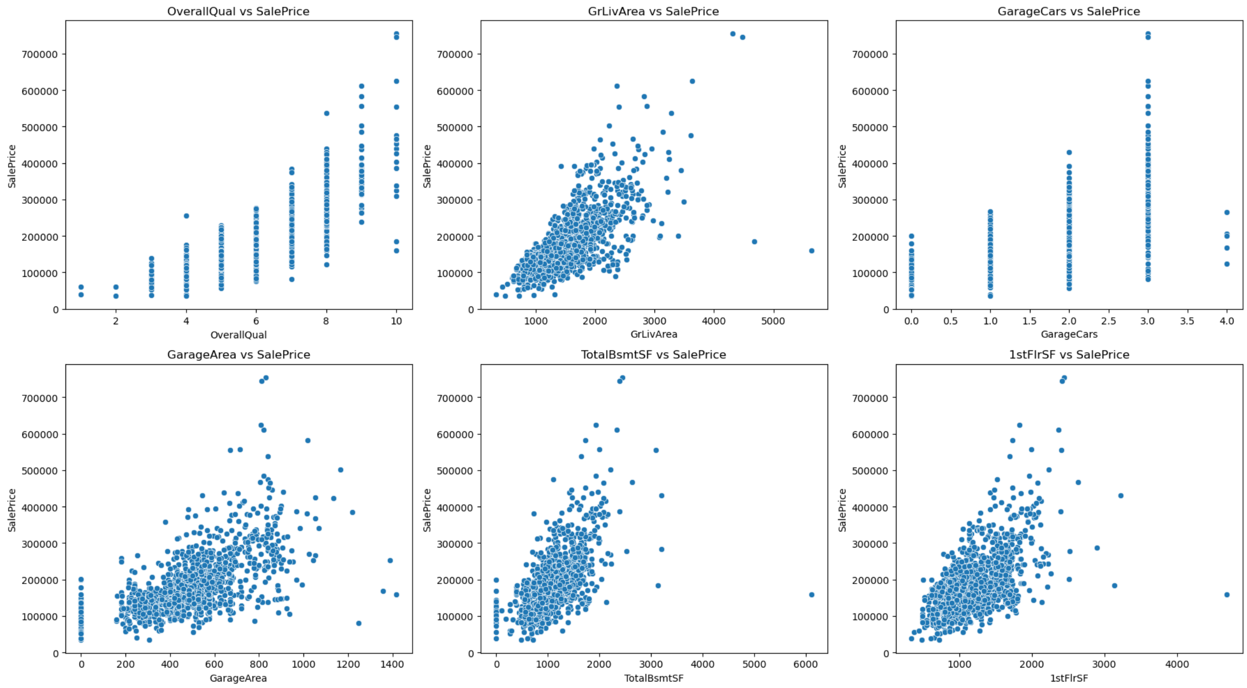

```
# Calculate correlation with SalePrice
cor_target = abs(cormat_housing_training["SalePrice"])

# Select features with correlation greater than 0.5
relevant_features = cor_target[cor_target > 0.5]

# Sort relevant features by correlation coefficient in descending
order
relevant_features = relevant_features.sort_values(ascending=False)
```

```python
# Display the relevant features
relevant_features
```

```
SalePrice       1.000000
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
GarageYrBlt     0.504317
Name: SalePrice, dtype: float64
```

Below are plots that examine the relationship between variables of interest and sale price

```python
variables_of_interest = ['OverallQual', 'GrLivArea', 'GarageCars',
'GarageArea', 'TotalBsmtSF', '1stFlrSF']

# Plotting relationships with SalePrice
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

for ax, var in zip(axes.flatten(), variables_of_interest):
    sns.scatterplot(x=housing_training_data[var],
y=housing_training_data['SalePrice'], ax=ax)
    ax.set_title(f'{var} vs SalePrice')

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



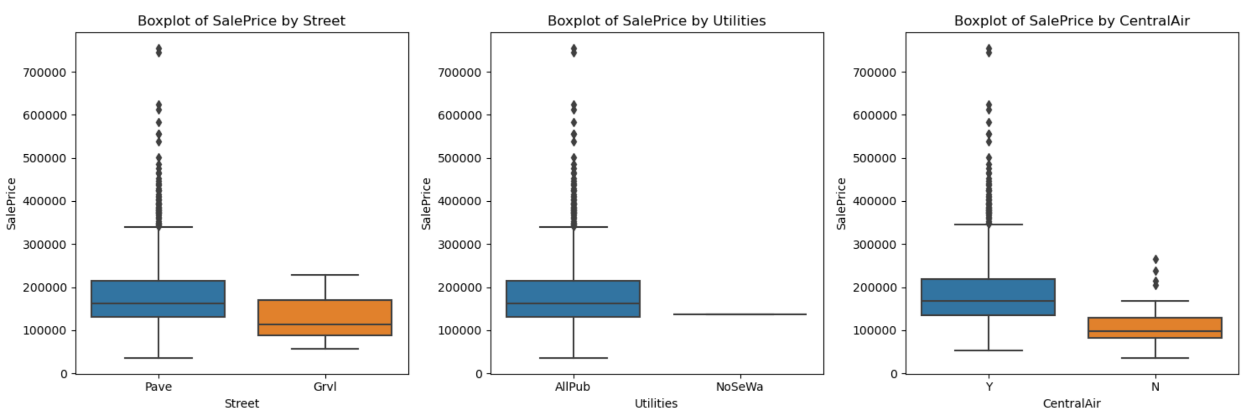
To identify the most predictive binary categorical variables for a regression model, we will utilize boxplots and conduct t-tests to assess which binary indicators exhibit the strongest correlation with home sale prices.

```
# Define list of categorical variables
categorical_variables = ['MSZoning', 'Street', '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',
                        'GarageQual', 'GarageCond', 'PavedDrive',
'SaleType', 'SaleCondition']

# Calculate number of unique categories for each variable
category_counts = [len(housing_training_data[var].unique()) for var in
categorical_variables]

# Create a DataFrame to summarize categorical variables and their
category counts
categorical_variable_dictionary = {'Categorical Predictor':
categorical_variables, 'Number of Categories': category_counts}
categorical_var_df = pd.DataFrame(categorical_variable_dictionary)

# Identify indicator variables (binary categorical variables)
```

```
indicator_variables_df = categorical_var_df[categorical_var_df['Number
of Categories'] == 2]

# Identify non-indicator categorical variables (more than two
categories)
non_indicator_categorial_vars_df =
categorical_var_df[categorical_var_df['Number of Categories'] > 2]

# Display the results
print("Summary of Categorical Variables:")
print(", ".join(f"('{var}':
{len(housing_training_data[var].unique())})" for var in
categorical_variables))
print("\nIndicator Variables (Binary):")
print(", ".join(f"('{var}': {num})" for var, num in
zip(indicator_variables_df['Categorical Predictor'],
indicator_variables_df['Number of Categories'])))
# Print results for non_indicator_categorial_vars_df
print("\nNon-Indicator Categorical Variables (More than Two
Categories):")
print(", ".join(f"('{var}': {num})" for var, num in
zip(non_indicator_categorial_vars_df['Categorical Predictor'],
non_indicator_categorial_vars_df['Number of Categories'])))
```

```
Summary of Categorical Variables:
('MSZoning': 5), ('Street': 2), ('LotShape': 4), ('LandContour': 4),
('Utilities': 2), ('LotConfig': 5), ('LandSlope': 3), ('Neighborhood':
25), ('Condition1': 9), ('Condition2': 8), ('BldgType': 5),
('HouseStyle': 8), ('RoofStyle': 6), ('RoofMatl': 8), ('Exterior1st':
15), ('Exterior2nd': 16), ('MasVnrType': 4), ('ExterQual': 4),
('ExterCond': 5), ('Foundation': 6), ('BsmtQual': 5), ('BsmtCond': 5),
('BsmtExposure': 5), ('BsmtFinType1': 7), ('BsmtFinType2': 7),
('Heating': 6), ('HeatingQC': 5), ('CentralAir': 2), ('Electrical':
6), ('KitchenQual': 4), ('Functional': 7), ('FireplaceQu': 6),
('GarageType': 7), ('GarageQual': 6), ('GarageCond': 6),
('PavedDrive': 3), ('SaleType': 9), ('SaleCondition': 6)

Indicator Variables (Binary):
('Street': 2), ('Utilities': 2), ('CentralAir': 2)

Non-Indicator Categorical Variables (More than Two Categories):
('MSZoning': 5), ('LotShape': 4), ('LandContour': 4), ('LotConfig':
5), ('LandSlope': 3), ('Neighborhood': 25), ('Condition1': 9),
('Condition2': 8), ('BldgType': 5), ('HouseStyle': 8), ('RoofStyle':
6), ('RoofMatl': 8), ('Exterior1st': 15), ('Exterior2nd': 16),
('MasVnrType': 4), ('ExterQual': 4), ('ExterCond': 5), ('Foundation':
6), ('BsmtQual': 5), ('BsmtCond': 5), ('BsmtExposure': 5),
('BsmtFinType1': 7), ('BsmtFinType2': 7), ('Heating': 6),
('HeatingQC': 5), ('Electrical': 6), ('KitchenQual': 4),
('Functional': 7), ('FireplaceQu': 6), ('GarageType': 7),
```

```
('GarageQual': 6), ('GarageCond': 6), ('PavedDrive': 3), ('SaleType':
9), ('SaleCondition': 6)

# Define the indicator variables of interest
indicator_vars = ['Street', 'Utilities', 'CentralAir']

# Create subplots for each indicator variable
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

# Iterate through each indicator variable and create boxplots
for var, subplot in zip(indicator_vars, ax.flatten()):
    sns.boxplot(x=var, y='SalePrice', data=housing_training_data,
ax=subplot)
    subplot.set_title(f'Boxplot of SalePrice by {var}')
    subplot.set_xlabel(var)
    subplot.set_ylabel('SalePrice')

# Adjust layout and display the plots
fig.tight_layout()
plt.show();
```




```
# Define the indicator variables
indicator_vars = ['Street', 'Utilities', 'CentralAir']

# Run T-tests for each indicator variable
Street_t_test = ttest_ind(
    housing_training_data['SalePrice'][housing_training_data['Street']
== 'Pave'],
    housing_training_data['SalePrice'][housing_training_data['Street']
== 'Grvl'],
    equal_var=False
)

Utilities_t_test = ttest_ind(
    housing_training_data['SalePrice']
[housing_training_data['Utilities'] == 'AllPub'],
    housing_training_data['SalePrice']
```

```
[housing_training_data['Utilities'] == 'NoSeWa'],
    equal_var=False
)

CentralAir_t_test = ttest_ind(
    housing_training_data['SalePrice']
[housing_training_data['CentralAir'] == 'Y'],
    housing_training_data['SalePrice']
[housing_training_data['CentralAir'] == 'N'],
    equal_var=False
)

# Compile T-test statistics and p-values into lists
Indicator_Variable_t_test_statistics = [Street_t_test[0],
Utilities_t_test[0], CentralAir_t_test[0]]
Indicator_Variable_t_test_p_values = [Street_t_test[1],
Utilities_t_test[1], CentralAir_t_test[1]]

# Create a dictionary for the T-test results
indicator_var_t_tests = {
    'Indicator Variable': indicator_vars,
    'T-Test Statistic': Indicator_Variable_t_test_statistics,
    'P-Values': Indicator_Variable_t_test_p_values
}

# Create a DataFrame from the dictionary
Indicator_var_t_test_df = pd.DataFrame(indicator_var_t_tests)

# Apply background gradient to highlight values in the DataFrame
styled_df =
Indicator_var_t_test_df.style.background_gradient(cmap='Greens')

# Display the styled DataFrame
styled_df
```

```
/opt/anaconda3/lib/python3.11/site-packages/scipy/stats/
_stats_py.py:1103: RuntimeWarning: divide by zero encountered in
divide
  var *= np.divide(n, n-ddof)  # to avoid error on division by zero
/opt/anaconda3/lib/python3.11/site-packages/scipy/stats/_stats_py.py:1
103: RuntimeWarning: invalid value encountered in scalar multiply
  var *= np.divide(n, n-ddof)  # to avoid error on division by zero
```

```
<pandas.io.formats.style.Styler at 0x17abae190>
```

Correlation with SalePrice: Calculate correlation coefficients (such as Pearson correlation) between encoded categorical variables and SalePrice. Higher absolute correlation coefficients suggest stronger relationships, which will guide our selection for ANOVA testing.

```
encoded_categorical_vars =
housing_training_data[categorical_variables].apply(LabelEncoder().fit_
transform)

# Calculate correlation with SalePrice
corr_with_saleprice =
encoded_categorical_vars.corrwith(housing_training_data['SalePrice'])

# Identify variables with high correlation (absolute value)
relevant_categorical_vars =
corr_with_saleprice[abs(corr_with_saleprice) > 0.5].index.tolist()

print("Categorical variables with high correlation with SalePrice:")
print(relevant_categorical_vars)
```

```
Categorical variables with high correlation with SalePrice:
['ExterQual', 'BsmtQual', 'KitchenQual']
```

It looks like we've identified `ExterQual`, `BsmtQual`, and `KitchenQual` as categorical variables that show high correlation with `SalePrice`. These variables are likely to be significant predictors in your analysis. If you need further assistance with analyzing these variables or any other aspect of your project, feel free to ask!

```
# Define ANOVA variables
ANOVA_variables = ['ExterQual', 'BsmtQual', 'KitchenQual']

# ExterQual ANOVA
ExterQual_Gd = housing_training_data['SalePrice']
[housing_training_data['ExterQual'] == 'Gd']
ExterQual_TA = housing_training_data['SalePrice']
[housing_training_data['ExterQual'] == 'TA']
ExterQual_Ex = housing_training_data['SalePrice']
[housing_training_data['ExterQual'] == 'Ex']
ExterQual_Fa = housing_training_data['SalePrice']
[housing_training_data['ExterQual'] == 'Fa']

ANOVA_ExterQual = f_oneway(ExterQual_Gd, ExterQual_TA, ExterQual_Ex,
ExterQual_Fa)

# BsmtQual ANOVA
BsmtQual_Gd = housing_training_data['SalePrice']
[housing_training_data['BsmtQual'] == 'Gd']
BsmtQual_TA = housing_training_data['SalePrice']
[housing_training_data['BsmtQual'] == 'TA']
BsmtQual_Ex = housing_training_data['SalePrice']
[housing_training_data['BsmtQual'] == 'Ex']
BsmtQual_None = housing_training_data['SalePrice']
[housing_training_data['BsmtQual'] == 'None']
BsmtQual_Fa = housing_training_data['SalePrice']
[housing_training_data['BsmtQual'] == 'Fa']
```

```
ANOVA_BsmtQual = f_oneway(BsmtQual_Gd, BsmtQual_TA, BsmtQual_Ex,
BsmtQual_None, BsmtQual_Fa)

# KitchenQual ANOVA
KitchenQual_Gd = housing_training_data['SalePrice']
[housing_training_data['KitchenQual'] == 'Gd']
KitchenQual_TA = housing_training_data['SalePrice']
[housing_training_data['KitchenQual'] == 'TA']
KitchenQual_Ex = housing_training_data['SalePrice']
[housing_training_data['KitchenQual'] == 'Ex']
KitchenQual_Fa = housing_training_data['SalePrice']
[housing_training_data['KitchenQual'] == 'Fa']

ANOVA_KitchenQual = f_oneway(KitchenQual_Gd, KitchenQual_TA,
KitchenQual_Ex, KitchenQual_Fa)

# Compile Outputs
ANOVA_statistics = [ANOVA_ExterQual[0], ANOVA_BsmtQual[0],
ANOVA_KitchenQual[0]]
ANOVA_p_values = [ANOVA_ExterQual[1], ANOVA_BsmtQual[1],
ANOVA_KitchenQual[1]]

ANOVA_outputs = {'Categorical Variable': ANOVA_variables, 'Test
Statistic': ANOVA_statistics, 'P-Values': ANOVA_p_values}
ANOVA_df = pd.DataFrame(ANOVA_outputs)
ANOVA_df.style.background_gradient(cmap='Greens')
```

```
<pandas.io.formats.style.Styler at 0x17b0e0d10>
```

Based on the analysis above, the variable for exterior quality shows significant promise for creating a dichotomous variable in our regression model. We'll apply the Tukey-Cramer Multiple Comparison Test to confirm statistically significant mean differences through pairwise comparisons of categorical variable values.

```
tukey_cramer_result =
pairwise_tukeyhsd(endog=housing_training_data['SalePrice'],
groups=housing_training_data['ExterQual'],alpha=0.05)
print(tukey_cramer_result)
```

```
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
==================================================================
group1 group2   meandiff   p-adj     lower        upper     reject
------------------------------------------------------------------
    Ex     Fa -279375.7473    0.0 -323896.9633 -234854.5313   True
    Ex     Gd -135727.4513    0.0 -157297.2472 -114157.6553   True
    Ex     TA -223019.6481    0.0  -244104.848 -201934.4481   True
    Fa     Gd   143648.296    0.0  103567.2848  183729.3071   True
    Fa     TA   56356.0992 0.0016   16533.7811   96178.4172   True
```

```
    Gd     TA  -87292.1968    0.0  -95594.8733  -78989.5203   True
-------------------------------------------------------------------
```

Based on the results from the ANOVA and Tukey-Cramer tests, which suggest that excellent home exterior quality could be a strong predictor of sale price, we will create a dichotomous variable to indicate whether a home has excellent exterior quality.

```
housing_training_data['Excellent_Exterior_Quality'] =
np.where(housing_training_data['ExterQual'] == 'Ex', True, False)
```

Encode important categorical variables

```
important_categorical = ['ExterQual', 'BsmtQual',
'KitchenQual','CentralAir']

# process columns, apply LabelEncoder to categorical features
for i in important_categorical:
    lbl = LabelEncoder()
    lbl.fit(list(housing_training_data[i].values))
    housing_training_data[i] =
lbl.transform(list(housing_training_data[i].values))

# shape
print('Shape all_data: {}'.format(housing_training_data.shape))

LabelEncoder()

LabelEncoder()

LabelEncoder()

LabelEncoder()

Shape all_data: (1460, 78)
```

Feature Creation

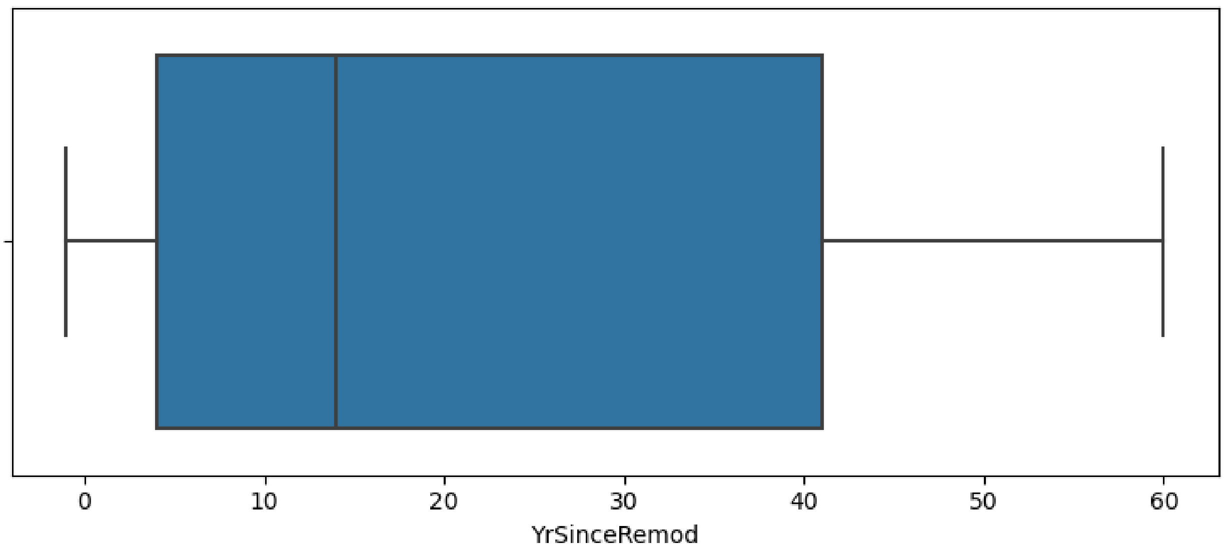
We will create a new feature to represent the number of years since a home was last remodeled, which may enhance the accuracy of our home sale price prediction models.

```
# Create a new variable: years since the house was remodeled from
selling date (use construction date if no remodeling or additions)
housing_training_data['YrSinceRemod'] =
housing_training_data['YrSold'] -
housing_training_data['YearRemodAdd']

# Create a boxplot for YrSinceRemod
sns.boxplot(x='YrSinceRemod', data=housing_training_data)

# Compute Pearson correlation coefficient and p-value for SalePrice
```

```
and YrSinceRemod
correlation_coefficient, p_value =
stats.pearsonr(housing_training_data['YrSinceRemod'],
housing_training_data['SalePrice'])
print("Pearson correlation coefficient and p-value for SalePrice and
Years since House was remodeled/built:")
print("Correlation coefficient:", correlation_coefficient)
print("P-value:", p_value)
```

```
<Axes: xlabel='YrSinceRemod'>
```

```
Pearson correlation coefficient and p-value for SalePrice and Years
since House was remodeled/built:
Correlation coefficient: -0.5090787380156276
P-value: 4.374855446379975e-97
```



Additionally, we'll create a feature to represent the total square footage of the home, which can further contribute to our home sale price prediction models.

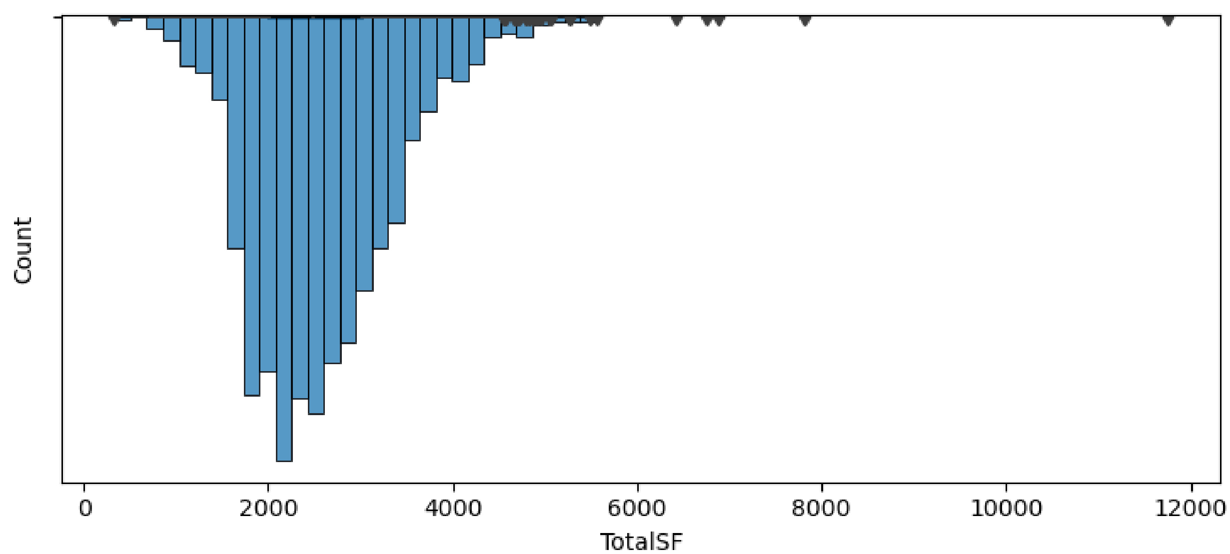
```
# Create a new variable: Total square feet (TotalSF)
housing_training_data['TotalSF'] =
housing_training_data['TotalBsmtSF'] +
housing_training_data['GrLivArea']

# Create a boxplot for TotalSF
sns.boxplot(x='TotalSF', data=housing_training_data);

# Drop large outliers from the dataframe
housing_training_data.drop(housing_training_data[housing_training_data
['TotalSF'] > 6000].index, inplace=True)

# Visualize distribution without extreme outliers
```

```python
sns.histplot(data=housing_training_data, x="TotalSF");

# Compute Pearson correlation coefficient and p-value for SalePrice
and TotalSF
correlation_coefficient, p_value =
stats.pearsonr(housing_training_data['TotalSF'],
housing_training_data['SalePrice'])
print("Pearson correlation coefficient and p-value for SalePrice and
TotalSF (Total square feet - includes basement):")
print("Correlation coefficient:", correlation_coefficient)
print("P-value:", p_value)
```

```
<Axes: xlabel='TotalSF'>
```

```
<Axes: xlabel='TotalSF', ylabel='Count'>
```

```
Pearson correlation coefficient and p-value for SalePrice and TotalSF
(Total square feet - includes basement):
Correlation coefficient: 0.8199962912972798
P-value: 0.0
```



Model Assumptions

1. Linearity
2. Homoscedasticity
3. Independence of Errors
4. Multivariate Normality
5. No or little Multicollinearity

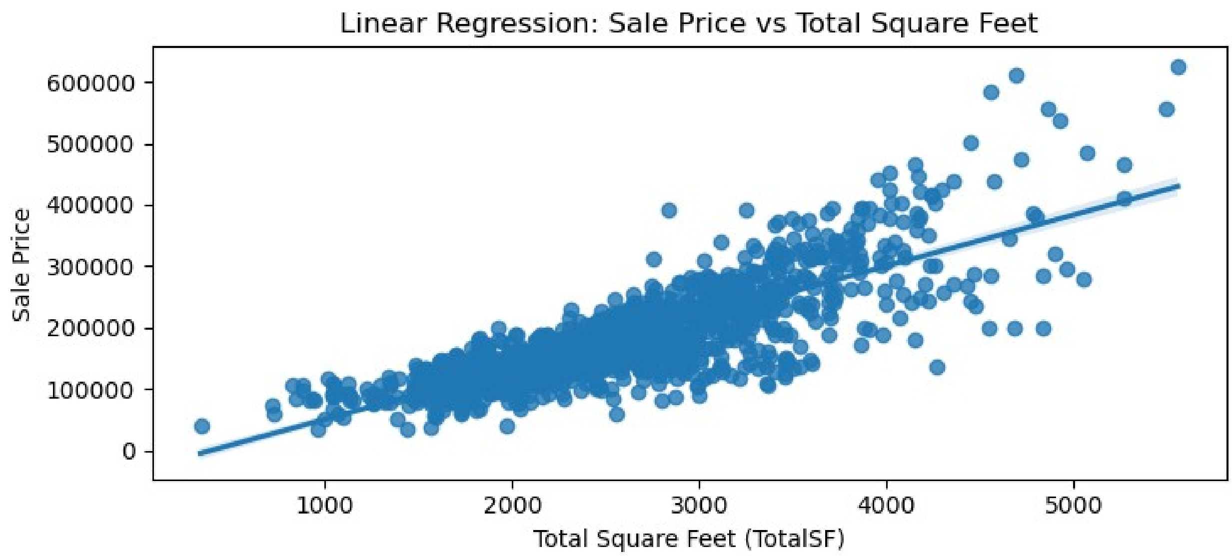
Constructing Models to Predict Home Prices

Below are simple and multiple regression analyses that explore the relationships between variables of interest and sale prices.

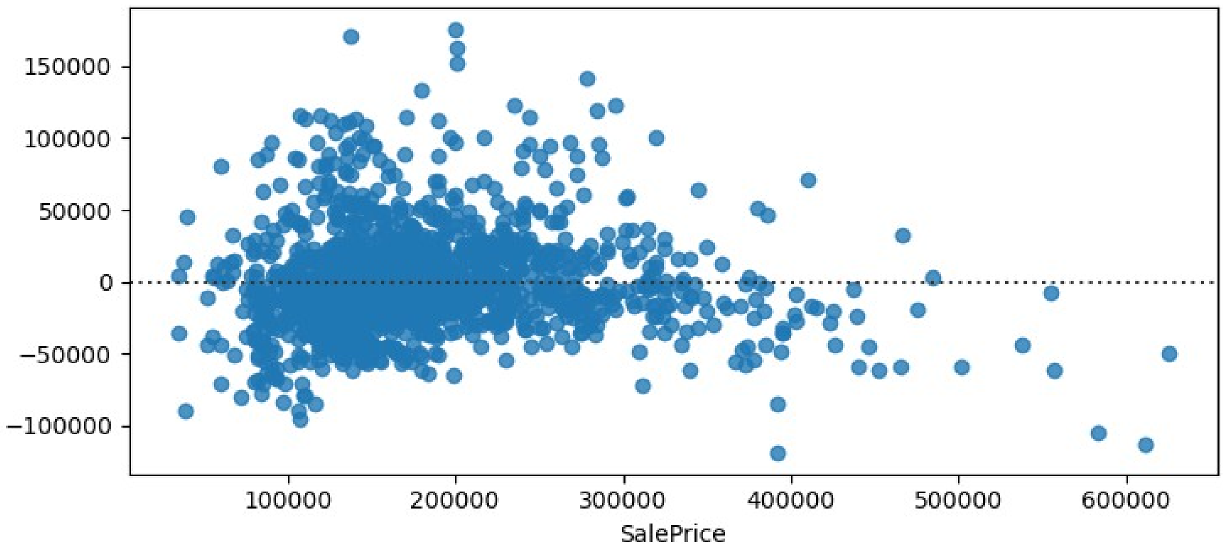
```
# New feature is highly correlated, let's try a simple linear
regression
x = housing_training_data['TotalSF']
y = housing_training_data['SalePrice']

# Add constant to predictor variables
X = sm.add_constant(x)

# Fit linear regression model
model = sm.OLS(y, X).fit()

# View model summary
print(model.summary())

# Plot the regression model
sns.regplot(x=x, y=y)
plt.xlabel('Total Square Feet (TotalSF)')
plt.ylabel('Sale Price')
plt.title('Linear Regression: Sale Price vs Total Square Feet')
plt.show();
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:              SalePrice   R-squared:
0.672
Model:                            OLS   Adj. R-squared:
0.672
Method:                 Least Squares   F-statistic:
2982.
Date:                Sun, 30 Jun 2024   Prob (F-statistic):
0.00
Time:                        21:32:42   Log-Likelihood:
-17613.
No. Observations:                1455   AIC:
3.523e+04
Df Residuals:                    1453   BIC:
3.524e+04
Df Model:                           1

Covariance Type:            nonrobust

==============================================================================
========
```

```
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------
--------
const      -3.239e+04   4054.647     -7.989      0.000   -4.03e+04   -
2.44e+04
TotalSF       83.1361      1.522     54.610      0.000      80.150
86.122
========================================================================
========
Omnibus:                      125.366   Durbin-Watson:
1.967
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
649.680
Skew:                           0.197   Prob(JB):
8.39e-142
Kurtosis:                       6.250   Cond. No.
9.41e+03
========================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 9.41e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```




```
# plot the residuals
y_pred=model.predict(X)
sns.residplot(x=y, y=y_pred);
```



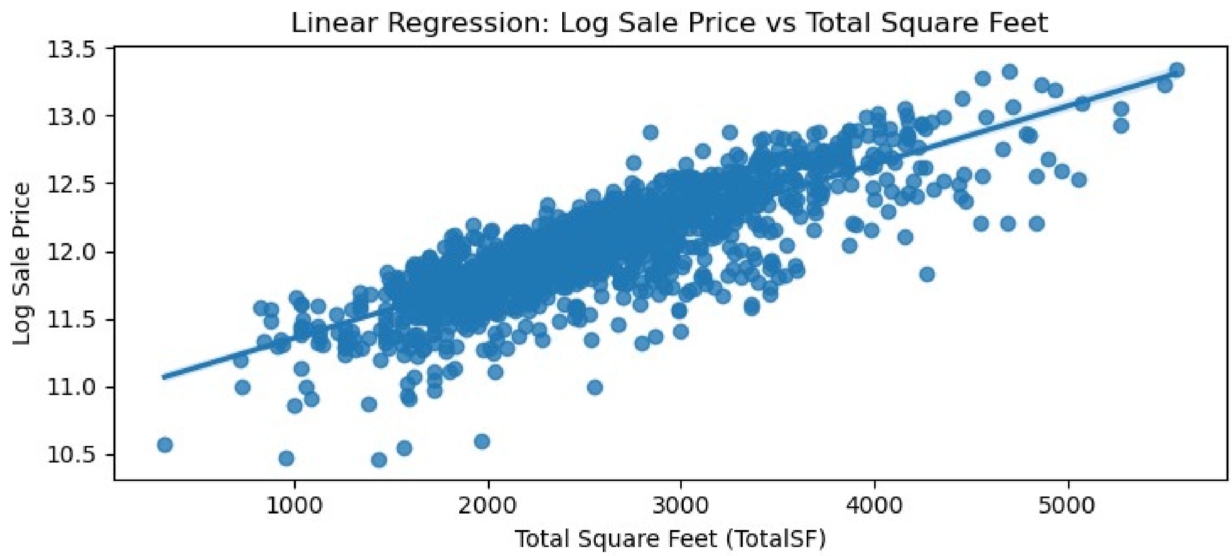
The residual plot indicates evidence of heteroscedasticity, as the residuals are not evenly scattered. Specifically, for higher sale prices, the residuals are negative, suggesting that the model tends to overestimate homes with higher sale prices. These observations indicate potential violations of the linearity, homoscedasticity, and independence of errors assumptions in the model.

Let's attempt to transform the dependent variable, SalePrice, as earlier in the analysis we observed that this transformation helped normalize its distribution.

```
# Log transform Sales Price variable
y_log = np.log(housing_training_data['SalePrice'])

# Add constant to predictor variables
X = sm.add_constant(x)

# Fit linear regression model
model = sm.OLS(y_log, X).fit()

# View model summary
print(model.summary())

# Plot the regression model
sns.regplot(x=x, y=y_log)
plt.xlabel('Total Square Feet (TotalSF)')
plt.ylabel('Log Sale Price')
plt.title('Linear Regression: Log Sale Price vs Total Square Feet')
plt.show();
```

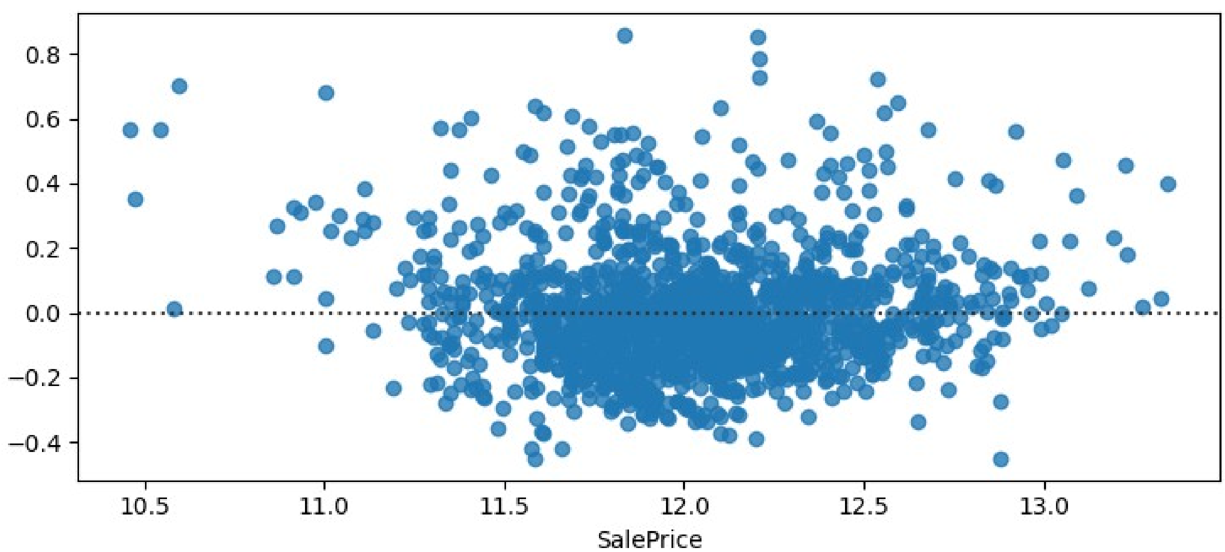
```
                            OLS Regression Results

======================================================================
========
Dep. Variable:              SalePrice   R-squared:
```

```
0.669
Model:                            OLS   Adj. R-squared:
0.669
Method:                 Least Squares   F-statistic:
2934.
Date:                Sun, 30 Jun 2024   Prob (F-statistic):
0.00
Time:                        21:32:42   Log-Likelihood:
89.766
No. Observations:                1455   AIC:
-175.5
Df Residuals:                    1453   BIC:
-165.0
Df Model:                           1

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
const         10.9256      0.021    518.068      0.000      10.884
10.967
TotalSF        0.0004   7.92e-06     54.167      0.000       0.000
0.000
======================================================================
========
Omnibus:                      274.748   Durbin-Watson:
1.937
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
567.536
Skew:                          -1.090   Prob(JB):
5.77e-124
Kurtosis:                       5.147   Cond. No.
9.41e+03
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 9.41e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```



The relationship between log(Sale Price) and TotalSF appears to be linear, satisfying the Linearity assumption.

```
# plot the residuals
y_pred=model.predict(X)
sns.residplot(x=y_log, y=y_pred);
```



The residuals appear to be more randomly scattered across values of Sales Price. This suggests that the model with the transformed Sales Price better meets the assumptions of Homoscedasticity and Independence of Errors compared to the model with the untransformed Sales Price.

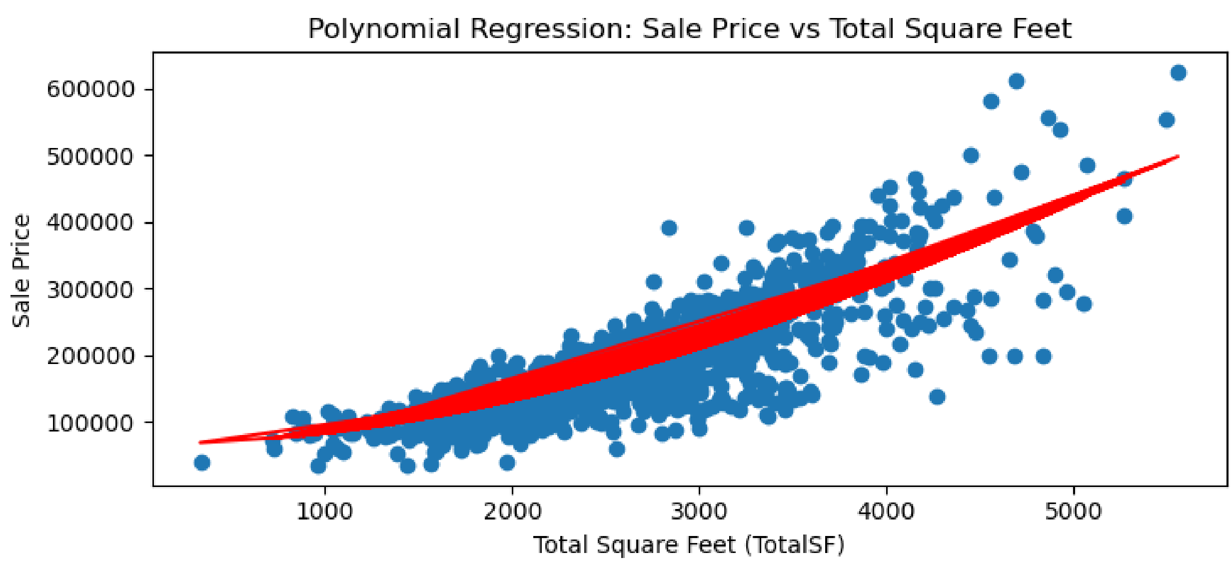
Polynomial Regression

Third-order polynomial regression was performed with Total SF as a predictor of Sales Price.

```python
x = housing_training_data[['TotalSF']]
y = housing_training_data['SalePrice']

polynomial_features = PolynomialFeatures(degree=3)
xp = polynomial_features.fit_transform(x)
xp.shape

model = sm.OLS(y, xp).fit()

# View model summary
print(model.summary())

# Predicted sales price
y_pred = model.predict(xp)

# Plot model against data
plt.scatter(x, y)
plt.plot(x, y_pred, color='red')
plt.xlabel('Total Square Feet (TotalSF)')
plt.ylabel('Sale Price')
plt.title('Polynomial Regression: Sale Price vs Total Square Feet')
plt.show();
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:              SalePrice   R-squared:
0.689
Model:                            OLS   Adj. R-squared:
0.688
Method:                 Least Squares   F-statistic:
1070.
Date:                Sun, 30 Jun 2024   Prob (F-statistic):
0.00
Time:                        21:32:42   Log-Likelihood:
-17576.
No. Observations:                1455   AIC:
3.516e+04
Df Residuals:                    1451   BIC:
3.518e+04
Df Model:                           3

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
```

```
--------
const        6.831e+04   2.17e+04      3.146      0.002    2.57e+04
1.11e+05
x1             -4.1538     24.479     -0.170      0.865     -52.171
43.864
x2              0.0203      0.009      2.311      0.021       0.003
0.037
x3          -1.007e-06   9.93e-07     -1.015      0.310   -2.95e-06
9.4e-07
======================================================================
========
Omnibus:                      126.279   Durbin-Watson:
1.954
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
633.593
Skew:                          -0.222   Prob(JB):
2.61e-138
Kurtosis:                       6.202   Cond. No.
5.70e+11
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 5.7e+11. This might indicate that
there are
strong multicollinearity or other numerical problems.
```



Multiple Linear Regression

Check the correlation between the two new variables. If they are highly correlated, we will avoid constructing a multiple linear regression model with both variables as predictors.

```
df_corr_mult_lreg = housing_training_data[['TotalSF','YrSinceRemod']]
df_corr_mult_lreg.corr()
```

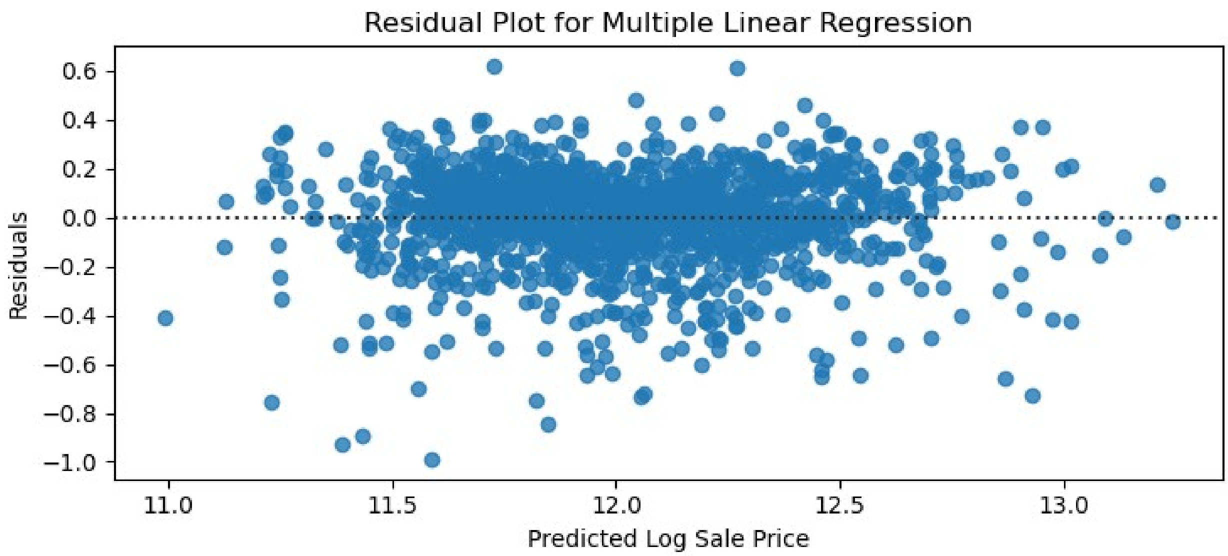
	TotalSF	YrSinceRemod
TotalSF	1.000000	-0.352279
YrSinceRemod	-0.352279	1.000000

TotalSF and YrSinceRemod show a low correlation, allowing us to proceed with constructing a multiple linear regression model using these variables as predictors of the log-transformed Sales Price.

```
x = housing_training_data[['TotalSF', 'YrSinceRemod']]
y_log = np.log(housing_training_data['SalePrice'])

# Add constant to predictor variables
X = sm.add_constant(x)

# Fit linear regression model
model = sm.OLS(y_log, X).fit()

# View model summary
print(model.summary())

# Plot the residuals
y_pred = model.predict(X)
sns.residplot(x=y_pred, y=y_log)
plt.xlabel('Predicted Log Sale Price')
plt.ylabel('Residuals')
plt.title('Residual Plot for Multiple Linear Regression')
plt.show();
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:              SalePrice   R-squared:
0.761
Model:                            OLS   Adj. R-squared:
0.760
Method:                 Least Squares   F-statistic:
2308.
Date:                Sun, 30 Jun 2024   Prob (F-statistic):
0.00
Time:                        21:32:42   Log-Likelihood:
326.20
```

```
No. Observations:                1455   AIC:
-646.4
Df Residuals:                    1452   BIC:
-630.6
Df Model:                           2

Covariance Type:            nonrobust

======================================================================
==========
                   coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
----------
const           11.2212      0.022    513.127      0.000      11.178
11.264
TotalSF          0.0004   7.19e-06     51.300      0.000       0.000
0.000
YrSinceRemod    -0.0062      0.000    -23.614      0.000      -0.007
-0.006
======================================================================
========
Omnibus:                      279.765   Durbin-Watson:
1.927
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
639.242
Skew:                          -1.065   Prob(JB):
1.55e-139
Kurtosis:                       5.450   Cond. No.
1.15e+04
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.15e+04. This might indicate that
there are
strong multicollinearity or other numerical problems.
```



An (R^2) value of 0.761 indicates that the model explains 76.1% of the variance in the dependent variable. The adjusted (R^2) value, which is similar to (R^2), suggests that adding multiple variables hasn't led to overfitting. However, the omnibus test indicates non-normally distributed residuals, with high kurtosis indicating peakedness compared to a normal distribution. The condition number of 1.15e+04 suggests potential multicollinearity in the model.

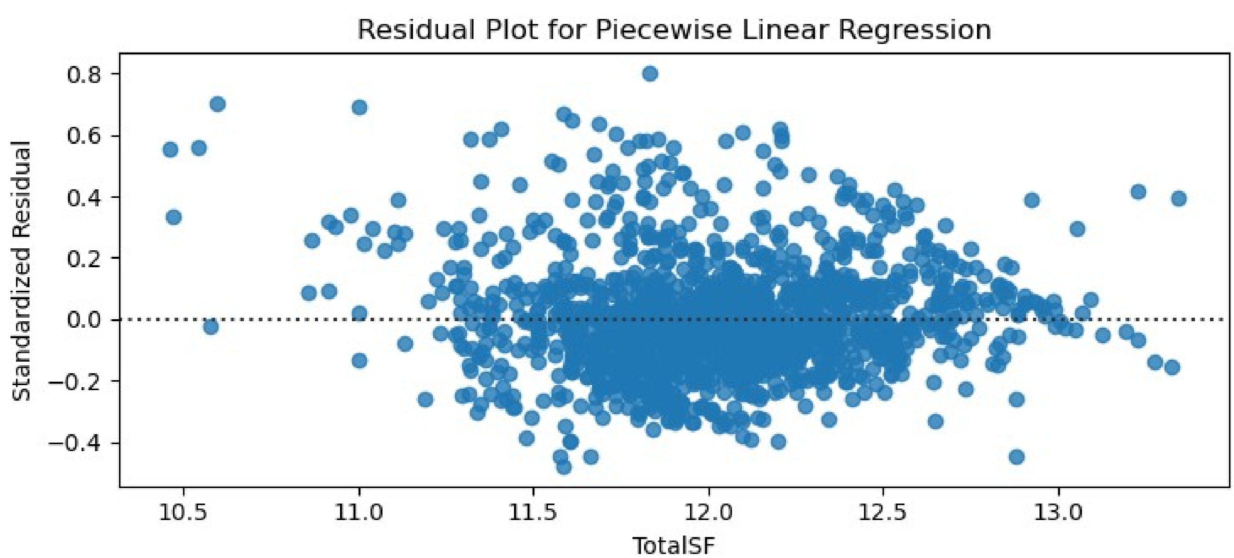
Piecewise Regression

It looks like you want to fit a piecewise regression model to predict sale price.

```
x = np.array(housing_training_data['TotalSF'])
y = np.array(np.log(housing_training_data['SalePrice']))

# Initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(x, y)

# Fit the data for four line segments
res = my_pwlf.fit(3)

# Predict for the determined points
xHat = np.array(housing_training_data['TotalSF'])
yHat = my_pwlf.predict(xHat)

# Prepare output dataframe
piecewise_regression_output = pd.DataFrame({"TotalSF": xHat,
                                            "Log Sale Price":
y.tolist(),
                                            "Predicted Log Sale
Price": yHat.tolist()})

# Calculate residuals
```

```
piecewise_regression_output['residual'] =
piecewise_regression_output['Log Sale Price'] -
piecewise_regression_output['Predicted Log Sale Price']
piecewise_regression_output['squared residuals'] =
piecewise_regression_output['residual'] ** 2

# Calculate RMSE
RMSE = np.sqrt(piecewise_regression_output['squared
residuals'].mean())
print(f"The Root Mean Squared Error of this piecewise regression model
is {RMSE}.")

# Calculate correlation
correlation = piecewise_regression_output['Log Sale
Price'].corr(piecewise_regression_output['Predicted Log Sale Price'])
print(f"The correlation of this piecewise regression model is
{correlation}.")

# Plot residuals
sns.residplot(x=y, y=yHat)
plt.ylabel('Standardized Residual')
plt.xlabel('TotalSF')
plt.title('Residual Plot for Piecewise Linear Regression')
plt.show();
```

```
The Root Mean Squared Error of this piecewise regression model is
0.2253833897981796.
The correlation of this piecewise regression model is
0.8215313536893254.
```



Inspection of multicollinearity: VIF, correlations

It appears that the correlation between the number of garage cars and total square feet is moderately high.

```
# The independent variables set
x = housing_training_data[['TotalSF', 'YrSinceRemod', 'GarageCars',
'ExterQual', 'CentralAir']]

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = x.columns

# Calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(x.values, i)
                   for i in range(len(x.columns))]

print(vif_data)
print(x.corr())
```

```
        feature        VIF
0       TotalSF  14.662165
1  YrSinceRemod   3.243161
2    GarageCars  10.247820
3     ExterQual  12.812931
4    CentralAir  14.699583
               TotalSF  YrSinceRemod  GarageCars  ExterQual
CentralAir
TotalSF       1.000000     -0.352279    0.556693  -0.483450
0.180174
YrSinceRemod -0.352279      1.000000   -0.422033   0.480107   -
0.299245
GarageCars    0.556693     -0.422033    1.000000  -0.447523
0.233414
ExterQual    -0.483450      0.480107   -0.447523   1.000000   -
0.085933
CentralAir    0.180174     -0.299245    0.233414  -0.085933
1.000000
```

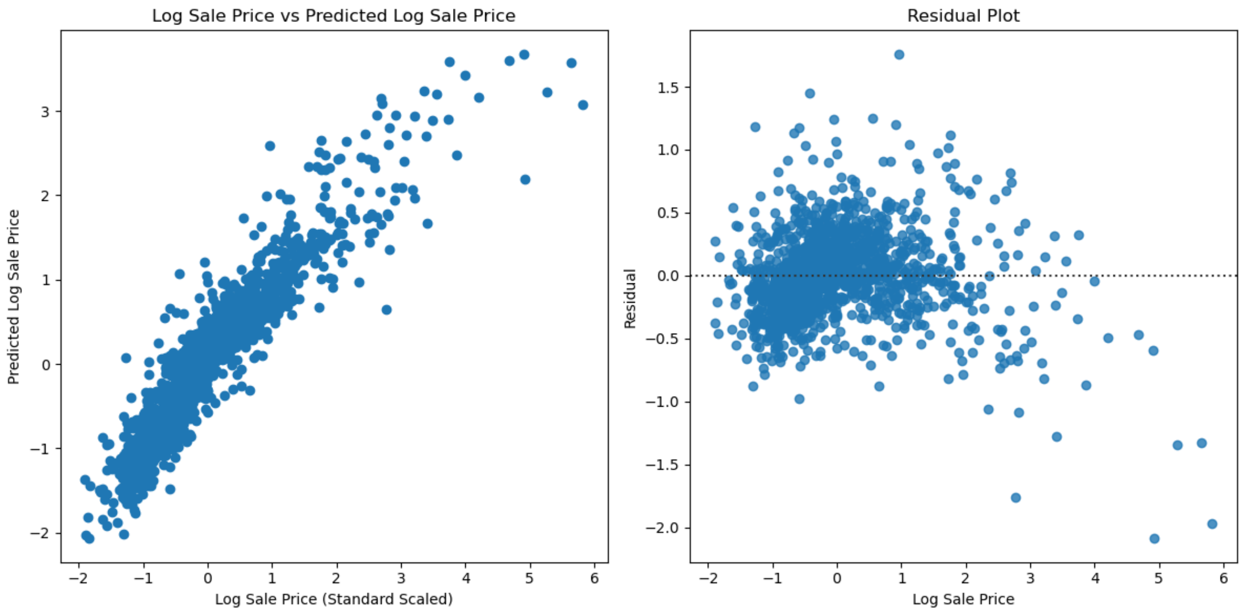
The VIF values suggest that our model exhibits multicollinearity across all features. Different sources propose various thresholds for identifying multicollinearity, with some suggesting values greater than 10, while others use thresholds like 5.0 or 1.0. While this may not significantly impact a predictive model, it could have implications for an inferential model.

Regression on principal components

```
# Initialize scaler and PCA
scaler = StandardScaler()
pca = PCA(n_components=8)
```

```python
# Independent variables
x_raw =
housing_training_data.select_dtypes(exclude=['object']).drop(columns=[
'SalePrice'])
x_scale = scaler.fit_transform(x_raw)
x_pca_raw = pca.fit_transform(x_raw)
x_pca_scale = pca.fit_transform(x_scale)  # PCA is affected by scale.
We use the scaled values.

# Importance scores of the principal components
loadings = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8'], index=x_raw.columns)

# Dependent variable
y_trans = scaler.fit_transform(housing_training_data[['SalePrice']])
y_raw = np.array(housing_training_data[['SalePrice']]).reshape(-1, 1)
y_scale = np.array(y_trans).reshape(-1, 1)

# Train linear model
regr = LinearRegression()
regr.fit(x_pca_scale, y_scale)
y_pred = regr.predict(x_pca_scale)

# Plotting
plt.figure(figsize=(12, 6))

# Scatter plot of actual vs predicted
plt.subplot(1, 2, 1)
plt.scatter(y_scale, y_pred)
plt.xlabel('Log Sale Price (Standard Scaled)')
plt.ylabel('Predicted Log Sale Price')
plt.title('Log Sale Price vs Predicted Log Sale Price')

# Residual plot
plt.subplot(1, 2, 2)
sns.residplot(x=y_scale.flatten(), y=y_pred.flatten())
plt.xlabel('Log Sale Price')
plt.ylabel('Residual')
plt.title('Residual Plot')

# Calculate RMSE
MSE = mean_squared_error(y_scale, y_pred)
print("MSE:", MSE)

r2 = r2_score(y_scale, y_pred)
print("R_squared:", r2)

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

```
MSE: 0.1321781506388294
R_squared: 0.8678218493611706
```




```
# Define cross-validation method to use
cv = KFold(n_splits=10, random_state=1, shuffle=True)

# Define polynomial model, degree 2
degree = 2

# PolynomialFeatures will create a new matrix consisting of all
polynomial combinations
# of the features with a degree less than or equal to the degree
specified (2)
poly_model = PolynomialFeatures(degree=degree)

# Transform polynomial features
poly_x_values = poly_model.fit_transform(x_pca_scale)

# Build multiple linear regression model
model = LinearRegression()

# Use k-fold CV to evaluate model
scores_mse = cross_val_score(model, poly_x_values, y_scale,
scoring='neg_mean_absolute_error',
                             cv=cv, n_jobs=-1)
print("Over 10 folds: %0.2f MSE with a standard deviation of %0.2f" %
(scores_mse.mean(), scores_mse.std()))

# Use k-fold CV to evaluate model R2
scores_r2 = cross_val_score(model, poly_x_values, y_scale,
```

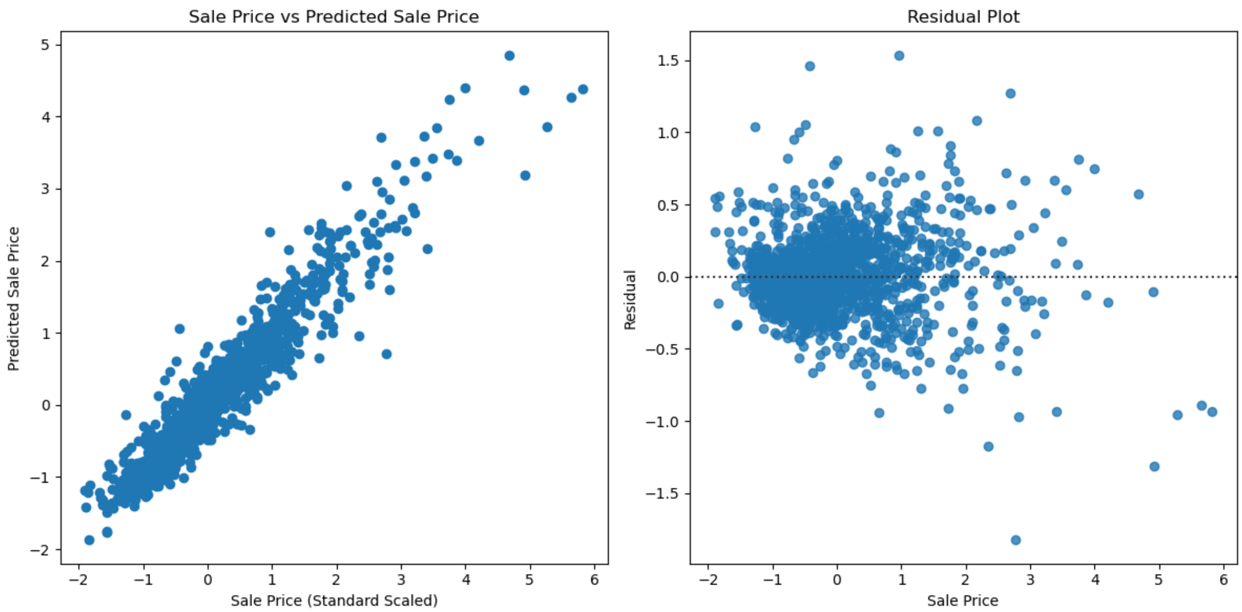
```
scoring='r2',
                            cv=cv, n_jobs=-1)
print("Over 10 folds: %0.2f r2 with a standard deviation of %0.2f" %
(scores_r2.mean(), scores_r2.std()))
```

```
Over 10 folds: -0.22 MSE with a standard deviation of 0.03
Over 10 folds: 0.90 r2 with a standard deviation of 0.02
```

Polynomial regression using predictors from PCA

```
# Scale the target variable 'SalePrice'
ypolyscaler = StandardScaler()
poly_y_scale =
ypolyscaler.fit_transform(np.array(housing_training_data['SalePrice'])
.reshape(-1, 1))

# Initialize a linear regression model
poly_regression_model = LinearRegression()

# Fit the polynomial regression model
poly_regression_model.fit(poly_x_values, poly_y_scale)

# Predictions
y_pred = poly_regression_model.predict(poly_x_values)

# Plotting
plt.figure(figsize=(12, 6))

# Scatter plot of actual vs predicted
plt.subplot(1, 2, 1)
plt.scatter(poly_y_scale, y_pred)
plt.xlabel('Sale Price (Standard Scaled)')
plt.ylabel('Predicted Sale Price')
plt.title('Sale Price vs Predicted Sale Price')

# Residual plot
plt.subplot(1, 2, 2)
sns.residplot(x=poly_y_scale.flatten(), y=y_pred.flatten())
plt.xlabel('Sale Price')
plt.ylabel('Residual')
plt.title('Residual Plot')

# Calculate MSE
MSE = mean_squared_error(poly_y_scale, y_pred)
print("MSE:", MSE)

# Calculate R-squared
r2 = r2_score(poly_y_scale, y_pred)
print("R_squared:", r2)
```

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

MSE: 0.0876962411967407
R_squared: 0.9123037588032593
```



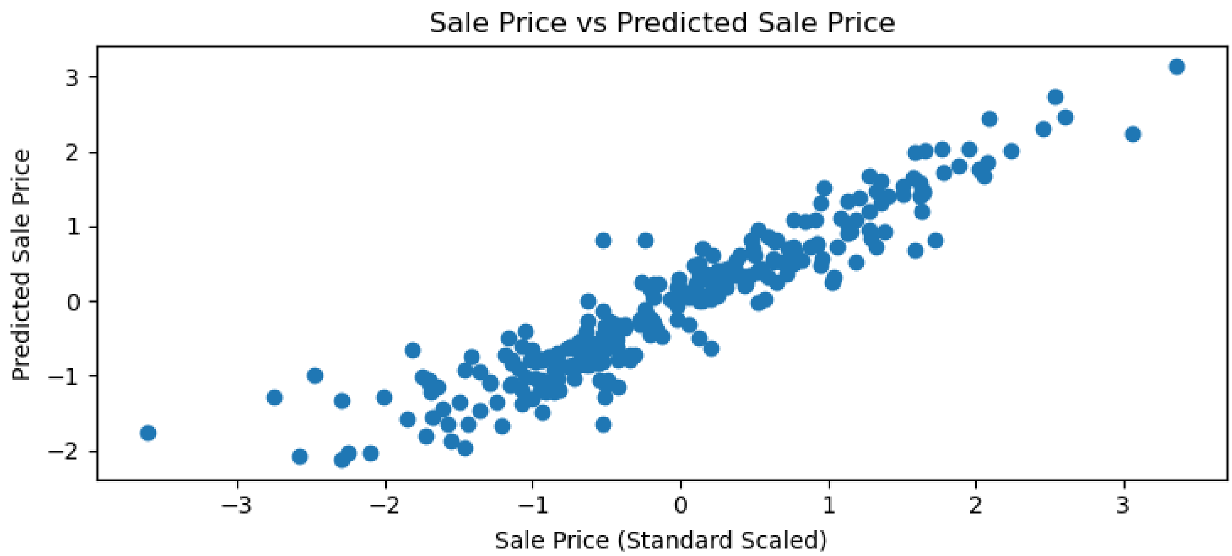
Ridge regression with PCA predictors and Cross-Validation to determine the optimal alpha

Principal components inherently do not exhibit collinearity, as each component represents a linear combination of the original variables with orthogonal directions. However, applying Ridge regularization to PCA can enhance its robustness and mitigate overfitting. Below are insights into the loadings within our principal components.

```
# Scale the target variable 'SalePrice'
yscaler = StandardScaler()
y_log_scale =
yscaler.fit_transform(np.array(np.log(housing_training_data['SalePrice
'])).reshape(-1, 1))

# PCA for independent variables
pca = PCA(n_components=8)

# Scale and transform independent variables with PCA
xscaler = StandardScaler()
x_raw =
housing_training_data.select_dtypes(exclude=['object']).drop(columns=[
'SalePrice'])
x_scale = xscaler.fit_transform(x_raw)
x_pca_scale = pca.fit_transform(x_scale)
```

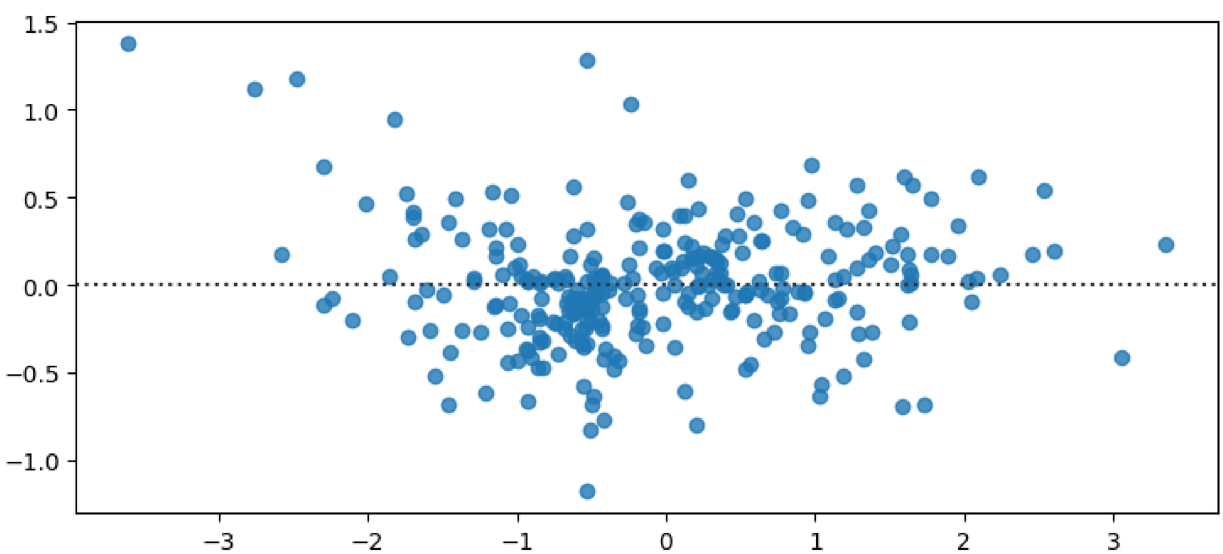
```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_pca_scale,
y_log_scale, test_size=0.2, random_state=42)

# Set alpha values to test
alpha_values = np.logspace(-4, 4, num=50)

# Create Ridge Regression model
ridge = Ridge()

# Set up Grid Search with cross-validation
param_grid = {'alpha': alpha_values}
grid_search = GridSearchCV(ridge, param_grid, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1)

# Fit Grid Search to training data
grid_search.fit(X_train, y_train)

# Get best alpha value
best_alpha = grid_search.best_params_['alpha']
print("Best alpha value:", best_alpha)

# Create and fit Ridge Regression model with best alpha value
ridge_best = Ridge(alpha=best_alpha)
ridgemodel = ridge_best.fit(X_train, y_train)

# Predict
y_pred = ridge_best.predict(X_test)

# Plot predicted vs actual sale prices
plt.scatter(y_test, y_pred)
plt.xlabel('Sale Price (Standard Scaled)')
plt.ylabel('Predicted Sale Price')
plt.title('Sale Price vs Predicted Sale Price')

# Calculate mean squared error (MSE) of predictions
MSE = mean_squared_error(y_test, y_pred)
print("MSE:", MSE)

# Calculate R-squared (R²) of predictions
r2 = r2_score(y_test, y_pred)
print("R_squared:", r2)

plt.show();
```

```
Best alpha value: 16.768329368110066
MSE: 0.13575130687968592
R_squared: 0.8806567366704461
```




```
# Residuals

sns.residplot(x=y_test, y=y_pred)
```

```
<Axes: >
```



Prepare Test CSV Data

```
# load test data
housing_testing_data = pd.read_csv('test.csv')

# Process columns, apply LabelEncoder to categorical features
for col in important_categorical:
    lbl = LabelEncoder()
    lbl.fit(housing_testing_data[col].values.astype(str))
```

```
    housing_testing_data[col] =
lbl.transform(housing_testing_data[col].values.astype(str))
```

```
LabelEncoder()
```

```
LabelEncoder()
```

```
LabelEncoder()
```

```
LabelEncoder()
```

Handle Null values just like train dataset

```
# Find null counts, percentage of null values, and column type
null_count = housing_testing_data.isnull().sum()
null_percentage = housing_testing_data.isnull().sum() * 100 /
len(housing_testing_data)
column_type = housing_testing_data.dtypes

# Create a summary DataFrame for columns with missing values
null_summary = pd.concat([null_count, null_percentage, column_type],
axis=1, keys=['Missing Count', 'Percentage Missing', 'Column Type'])

# Filter and sort columns with more than one null value
null_summary_only_missing = null_summary[null_count !=
0].sort_values('Percentage Missing', ascending=False)

# PoolQC, MiscFeature, Alley, Fence all have over 50% of missing
values, we will remove those from our dataframe
housing_testing_data.drop(['Alley','PoolQC','Fence','MiscFeature',
'MasVnrType' ],axis=1,inplace=True)

columns_None =
['SaleType','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','G
arageType','GarageFinish','GarageQual',

'MSZoning','FireplaceQu','Functional','Utilities','GarageCond',
'Exterior2nd','Exterior1st']
# set Nulls in non-numeric columns to 'None'
housing_testing_data[columns_None] =
housing_testing_data[columns_None].fillna('None')

# change Null values to 0 for the following variables
columns_zero = ['MasVnrArea',
'GarageArea','GarageCars','TotalBsmtSF','BsmtUnfSF','BsmtFinSF2','Bsmt
FinSF1','BsmtHalfBath','BsmtFullBath']
housing_testing_data[columns_zero] =
housing_testing_data[columns_zero].fillna(0)
housing_testing_data['LotFrontage'].fillna(housing_testing_data['LotFr
ontage'].median(), inplace=True)
housing_testing_data['GarageYrBlt'].fillna(housing_testing_data['Garag
```

```
eYrBlt'].median(), inplace=True)
housing_testing_data['TotalSF'] = housing_testing_data['TotalBsmtSF']
+ housing_testing_data['GrLivArea']
housing_testing_data['YrSinceRemod'] = housing_testing_data['YrSold']
- housing_testing_data['YearRemodAdd']
housing_testing_data['Excellent_Exterior_Quality'] =
np.where(housing_testing_data['ExterQual'] == 'Ex', True, False)
```

Predicting Home Sale Prices using the Ridge Model

```
x_test_raw = housing_testing_data.select_dtypes(exclude=['object'])
x_test_scale = xscaler.fit_transform(x_test_raw)
x_test_pca_scale = pca.transform(x_test_scale)
scaled_y_predtest = ridgemodel.predict(x_test_pca_scale)
y_predtest = yscaler.inverse_transform(scaled_y_predtest)
y_pred_final = np.exp(y_predtest)
predictiondf=pd.DataFrame(y_pred_final, columns=['SalePrice'])
predictiondf.insert(0, 'Id', housing_testing_data['Id'])

predictiondf.to_csv('salesprice_ridge.csv', index=False)
```

Upon submitting our home price predictions from the ridge regression model on Kaggle under the username sachinsharma03, we obtained a RMSE of 0.17007 for the testing dataset.

Predicting Home Sale Prices using the Polynomial Model

Let's predict Sales Price using our polynomial model

```
poly_model = PolynomialFeatures(degree=degree)
poly_x_test_values = poly_model.fit_transform(x_test_pca_scale)
y_poly_pred = poly_regression_model.predict(poly_x_test_values)
y_poly_predtest = ypolyscaler.inverse_transform(y_poly_pred)
predictionpolydf=pd.DataFrame(y_poly_predtest, columns=['SalePrice'])
predictionpolydf.insert(0, 'Id', housing_testing_data['Id'])

predictionpolydf.to_csv('salesprice_poly.csv', index=False)
```

Upon submitting our home price predictions from the polynomial model on Kaggle using the username **sachinsharma03**, we achieved a RMSE of 0.16430 for the testing dataset.

Discuss what your models tell you in layman's terms

Our models help us understand how different factors influence the sale prices of homes. By analyzing various features such as total square footage, years since the last remodel, garage size, and others, we can make predictions about a home's sale price. Here's a simplified explanation of what our models tell us:

1. **Total Square Footage**: Homes with larger total square footage generally have higher sale prices. This makes sense because bigger homes offer more living space and are typically more valuable.

2. **Years Since Remodel**: Homes that have been recently remodeled or built tend to have higher sale prices. Newer or updated homes are often more appealing to buyers, which can drive up their prices.

3. **Exterior Quality**: The quality of a home's exterior plays a significant role in its sale price. Homes with excellent exterior quality tend to sell for more, indicating that buyers value the condition and appearance of the outside of a house.

4. **Garage Size**: The number of cars a garage can accommodate is also an important factor. Larger garages, which can hold more cars, generally add to the home's value.

5. **Central Air**: Whether a home has central air conditioning impacts its sale price. Homes with central air are more comfortable and desirable, especially in warmer climates, which can increase their value.

6. **Predictive Power and Accuracy**: Our models use statistical techniques to fit the data and make predictions. We ensure these models are accurate by testing them against actual sale prices and adjusting them to improve their predictive power. For example, we use cross-validation to select the best parameters (like the alpha value in ridge regression) that minimize prediction errors.

7. **Handling Complex Relationships**: Some models, like polynomial regression, allow us to capture more complex relationships between features and sale prices. For instance, the relationship between square footage and sale price might not be perfectly linear, and polynomial regression can better model such nuances.

8. **Robustness and Overfitting**: We also take steps to ensure our models are robust and not overfitting the data. Overfitting happens when a model performs well on training data but poorly on new, unseen data. By using techniques like regularization and principal component analysis (PCA), we make our models more generalizable and reliable for making predictions on new data.

In summary, our models use various features of homes to predict their sale prices accurately. They help us understand which factors are most influential and ensure our predictions are reliable and robust.