MODULE 3: HOUSING PRICES ASSIGNMENT

Claire Markey, Julia Granito, Manny Hurtado, and Steve Desilets

MSDS 422: Practical Machine Learning

April 16th, 2023

Introduction

Developing a well-tuned predictive model (i.e, Ridge, Lasso, ElasticNet) can provide accurate, precise estimates of home prices, which may be helpful to real-estate industry stakeholders. Conceptually, a Lasso or ElasticNet approach may perform better if there are many good predictors due to cost functions. All of these methods offer advantages such as regularization, reducing collinearities, and improving performance on unseen data. Conversely, there are negative consequences such as not being the best options for variable selection due to the algorithms arbitrarily selecting which variable(s) to allocate to in the presence of multicollinearity. To that end, predictive models (i.e., Ridge, Lasso, ElasticNet) were evaluated to identify predictors of home prices and to determine which approach may be best suited to predict home prices.

Method

Data about home sale prices in Ames, Iowa, were downloaded from Kaggle and analyzed using Jupyter Notebooks (Montoya, 2016). An exploratory data analysis explored feature characteristics (associations, distributions, missingness, and outliers), as well as bivariate (Pearson correlations, t-tests, ANOVAs, simple linear regressions) and multivariate analyses (OLS, Ridge, Lasso, and ElasticNet regression). Multivariate analyses were conducted using a *k*-fold cross-validation design (James et al. 2021) and Kaggle training/validation sets. Underlying model assumptions and metrics were examined in analyses.

Results and Insights

First, descriptive statistics about the outcome of interest (Sale Price) were examined by constructing histograms and boxplots, then examined by calculating summary statistics. Sale price was skewed right, ranged from $34,900 to $755,000 ($M$= $180,921), and included outliers. A log transformation of the home sale price variable improved the normality of the outcome variable ahead of subsequent analyses. After addressing missing data and outliers, categorical variables were separated into dichotomous indicator, ordinal indicator, and non-indicator variables in order to be encoded. An alternative, larger dataset was created with these new variables. We used this dataset later on in the analysis that explored an ElasticNet regression model.

To determine which features should be included in OLS models, we examined the associations between SalePrice and features. Bivariate analyses suggested the following variables had strong relationships with Sale Price: central air ($t(1458) = 17.26$, $p< .001$), exterior quality ($F(3, 1456)= 443.33$, $p< .001$), and total square footage (TotalSF, a linear combination of GrLivArea and TotalBsmtSF; $r= .82$, $p< .001$). Simple linear regressions were conducted between Sale Price and its key predictors. A simple linear regression, piecewise regression model, and parsimonious five-factor multiple regression model exhibited R^2 values of .76, .82, and .84, respectively. Then a second-degree polynomial regression on principal components and a regression using L2 regularization on principal components were considered, which returned R^2 values of .91.

To further this analysis, regularization techniques were explored (Ridge, Lasso, ElasticNet) on the full dataset; these analyses included all encoded variables. We observe empirically that non-zero levels of alpha resulted in more generalizable models.

Hyperparameter tuning using different methods, including a five-fold cross-validation design, were then conducted to identify the alpha that minimizes the error on the testing data. First, using R^2 as the performance metric, we plotted training and testing data for alphas at even increments from 0 to .02. By inspection, there was an inflection point for the Lasso regularization technique. We then assessed that the optimal value for alpha was about .004, with an R^2 of .90.

Next, we explored a Lasso regression model to model the log sale prices. To do so, we standardized each of the numeric predictor variables and fit a Lasso regression model to the training data using the optimal tuning parameter. The R squared of this lasso model when applied to a validation set was 0.863. After applying the model to the Kaggle test dataset, the resulting root mean squared error was 0.16194.

For Ridge regression analyses, the optimal alpha was approximately .015 for the initial data; the alpha increased to .81 and the R^2 was .89 after including categorical variables. We found that our ElasticNet model performed the best out of the three regularization methods when all the encoded variables (full data set) were included, with an R^2 for the final model of .94 and a Kaggle RMSE score of .1356. Regarding ElasticNet, the best alpha values were .015 and .009 for the limited and full data sets, respectively. This indicates that a combination of L1 and L2 regularization is optimal for our dataset. The implications of an alpha close to zero for ElasticNet regression means that the importance of the predictors is likely to be distributed among the features. This results in the coefficients of important variables to be close, despite any high correlations. If the alpha value were closer to 1, therefore suggesting higher bias, then the coefficient for one of the correlated variables would be high, while the others would have shrunk to zero. In comparing the coefficients, we observe that the Lasso model has more non-zero coefficients with lower magnitude, while the ElasticNet coefficients are larger but the matrix is somewhat more sparse. By incorporating both L1 and L2 regularization, ElasticNet optimizes the cost function to produce the best fit.

Models such as ElasticNet are not ideal for feature selection as the penalty term may cause the model to skew the weights of certain predictors, making some good predictors appear unimportant. However, we do consider the variables in our model in the analysis. Regarding the Elastic Net model that achieved the lowest Kaggle RMSE score (0.1356), the largest coefficients included square footage variables (Total Square Ft), Overall Quality and Condition variables (overall material, finish, and condition of a home), and the Year Built variable. These had positive coefficients indicating Sale Price increased as the predictors increased. The coefficient of one of the encoded variables, indicating whether the home is in the Crawford neighborhood, was large, indicating that living in this neighborhood was associated with higher Sale Prices. Some variables have negative coefficients, like the engineered feature, Year Since Last remodel, and like some of the of the encoded variables related to the zoning classification of properties, specifically for commercial and medium-density residential properties. This indicates that a greater number of years since a remodel or selling a commercial or medium-density residential property can negatively impact the sale price of a home.

Exploratory data analysis provided insights about factors that impact the sale price of homes in Ames. This work successfully explored methods to accurately predict Sale Price using regression techniques (simple linear, MLR, Piecewise, Polynomial, Ridge, Lasso, and Elastic Net). Our highest performing model, built using Elastic Net regression, has practical implications for Ames real estate market stakeholders. Given the variability of the housing market over time, along with the fact that the housing market in Ames may not be representative of other cities or regions, extrapolating the results beyond the timeframe and city for the data used to train the model will likely lead to inaccurate predictions.

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2021. *An Introduction to Statistical Learning*. Springer. https://www.statlearning.com/

Montoya, Anna. 2016. "House Prices - Advanced Regression Techniques." *Kaggle*. https://www.kaggle.com/c/house-prices-advanced-regression-techniques

Appendix - Python Code and Outputs

Data Preparation

First, we will set up this notebook so that it will display multiple outputs for each cell if needed.

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Second, we will import the data. We will view the first five rows of data and the shape of the dataframe to confirm that the data imported correctly.

In [2]:
```
import pandas as pd
housing_training_data = pd.read_csv('train.csv')

# show first five rows of the data
housing_training_data.head()
# show number of columns and rows
housing_training_data.shape
```

Out[2]:

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

5 rows × 81 columns

Out[2]:
```
(1460, 81)
```

Distribution of the Dependent Variable

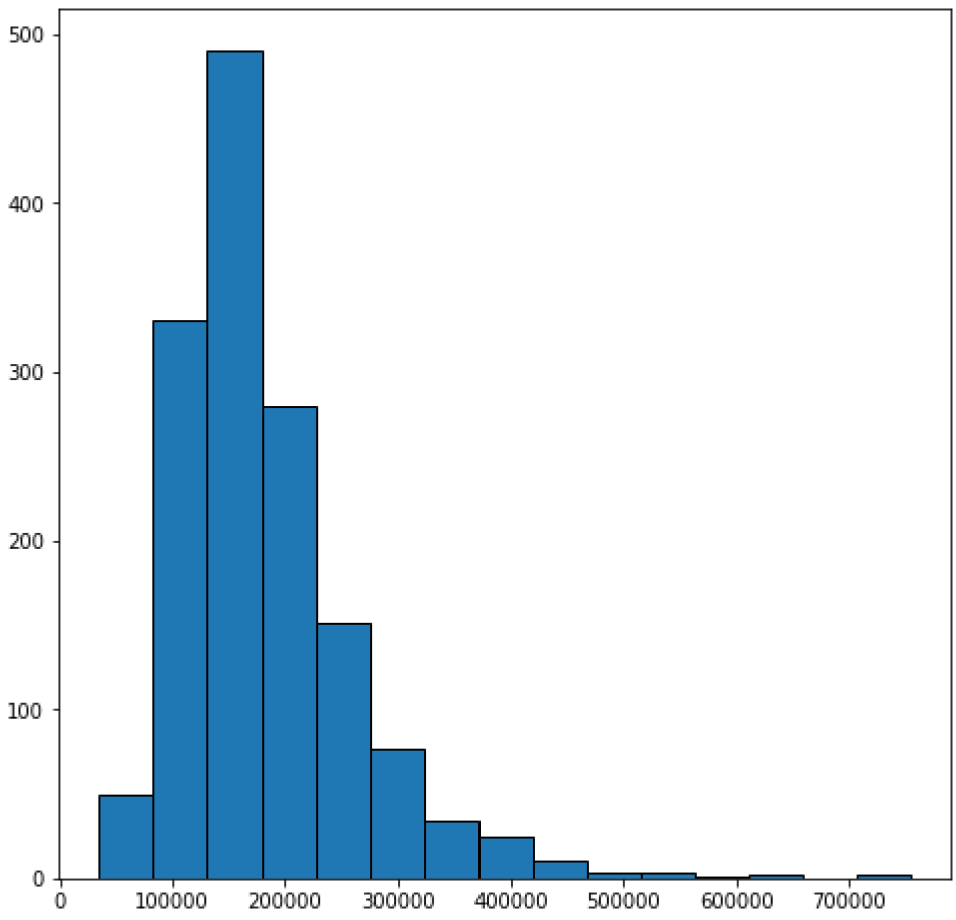
We can begin examining the distribution of this dataset's dependent variable, sale price, by generating summary statistics for this variable.

In [3]:
```
housing_training_data['SalePrice'].describe()
```

Out[3]:
```
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
```

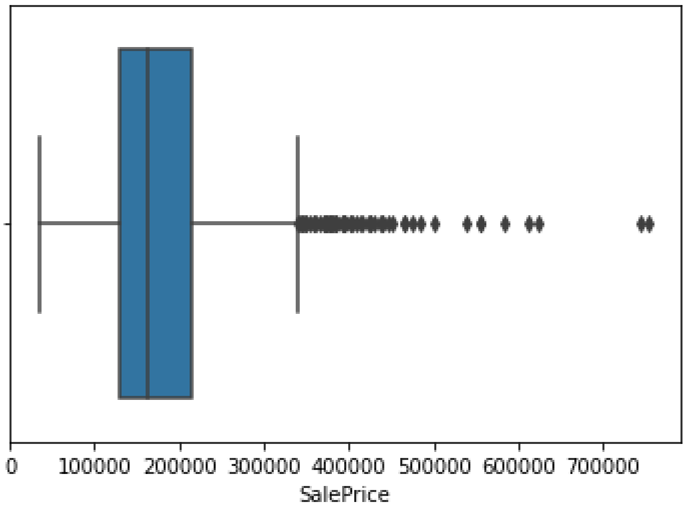
We can also construct a histogram and a boxplot to visualize the distribution of the sale price variable in this dataframe.

In [4]:
```
import seaborn as sns
import matplotlib.pyplot as plt

histogram = housing_training_data['SalePrice'].hist(edgecolor = 'black', bins = 15, fi
```



In [5]:
```
sns.boxplot(x=housing_training_data["SalePrice"])
```

Out[5]:
```
<AxesSubplot:xlabel='SalePrice'>
```



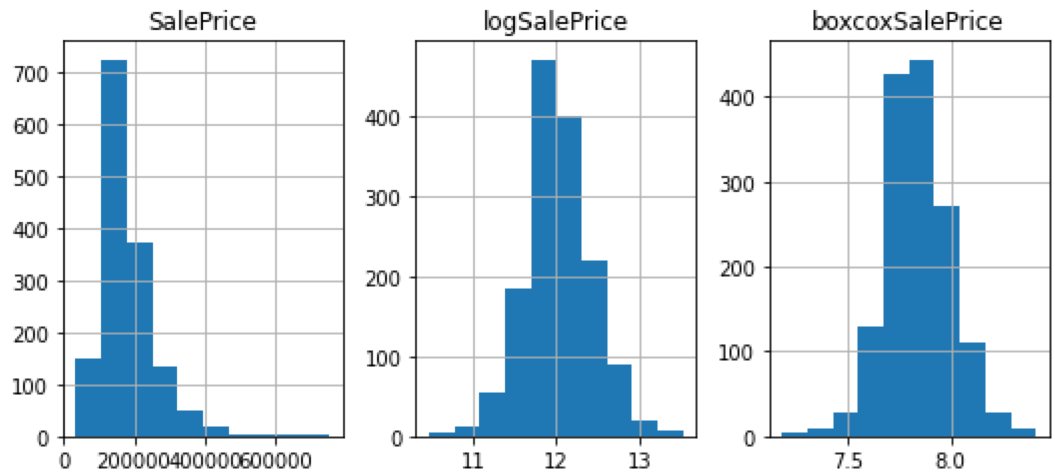
In [6]:
```
import numpy as np
from scipy import stats
from scipy.stats import norm, kurtosis
df = []
raw_data = housing_training_data['SalePrice']
transform_data = np.log(housing_training_data['SalePrice'])
transform_data2, best_lambda = stats.boxcox(housing_training_data['SalePrice'])


print("homeprice kurtosis:", kurtosis(raw_data))
print("log of homeprice kurtosis:", kurtosis(transform_data))
print("boxcox transform of homeprice kurtosis:", kurtosis(transform_data2))


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

s1 = pd.DataFrame(raw_data)
s2 = pd.DataFrame(np.array(transform_data).tolist(), columns = ['logSalePrice'])
s3 = pd.DataFrame(np.array(transform_data2).tolist(), columns = ['boxcoxSalePrice'])

fig, axes = plt.subplots(1, 3)

s1.hist('SalePrice', ax=axes[0])
s2.hist('logSalePrice', ax=axes[1])
s3.hist('boxcoxSalePrice', ax=axes[2])
plt.show()
```

```
homeprice kurtosis: 6.509812011089439
log of homeprice kurtosis: 0.8026555069117713
boxcox transform of homeprice kurtosis: 0.870759906431624
```

Out[6]:
```
array([<AxesSubplot:title={'center':'SalePrice'}>], dtype=object)
```

Out[6]:
```
array([<AxesSubplot:title={'center':'logSalePrice'}>], dtype=object)
```

Out[6]:
```
array([<AxesSubplot:title={'center':'boxcoxSalePrice'}>], dtype=object)
```



Shapiro Wilk test for normality

In [7]:
```
print("Shapiro Wilk test for normality: ", stats.shapiro(raw_data))
print("Shapiro Wilk test for normality: ", stats.shapiro(transform_data))
print("Shapiro Wilk test for normality: ", stats.shapiro(transform_data2))
```

```
Shapiro Wilk test for normality:  (0.869671642780304, 3.206247534576162e-33)
Shapiro Wilk test for normality:  (0.9912067651748657, 1.1490678986092462e-07)
Shapiro Wilk test for normality:  (0.9915341138839722, 1.906367685933219e-07)
```

The Shapiro Wilk test for normality (H0: normal, Ha: not-normal) suggests a departure from normality for both the raw and transformed data.

Investigation of Missing Data and Outliers

We can take a look at the counts of reported values in each column to determine the number of missing values for each variable in the dataframe.

In [8]:
```
# find null counts, percentage of null values, and column type
null_count = housing_training_data.isnull().sum()
null_percentage = housing_training_data.isnull().sum() * 100 / len(housing_training_da
column_type = housing_training_data.dtypes

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[8]:

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

We will deal with columns that contain missing values. For the purpose of this exploratory data analysis, we will use the percentage of nulls missing, the column type, and the other columns present in the data that may provide information that can be used to fill in the missing values.

We will remove columns with over 50% Null values.

In [9]:

```
# PoolQC, MiscFeature, Alley, Fence all have over 50% of missing values, we will remov
housing_training_data.drop(['Alley','PoolQC','Fence','MiscFeature'],axis=1,inplace=Tru

# show new shape
housing_training_data.shape
```

Out[9]:

```
(1460, 77)
```

We will set Null values in columns that are non-numeric to None.

In [10]:

```
# select non-numeric columns that contain more than 1 Null value
columns_None = ['BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','Ga
# set Nulls in non-numeric columns to 'None'
housing_training_data[columns_None] = housing_training_data[columns_None].fillna('None
```

We determine the best way to handle nulls for each numeric column. We replace nulls in Masonry veneer area with 0, nulls in Lot Frontage with the median, and nulls in Year Garage was built with the average between the year the garage was built and year house was built .

In [11]:
```
# change Null values to 0 for Masonry veneer area
housing_training_data['MasVnrArea'].fillna(0, inplace=True)

# show distribution stats for Lot Frontage
housing_training_data['LotFrontage'].describe()
# fill Nulls for Lot Frontage with median value
housing_training_data['LotFrontage'].fillna(housing_training_data['LotFrontage'].media

# average years between garage being built and years built
avg_years = round((housing_training_data['GarageYrBlt'] - housing_training_data['YearB
# fill Nulls with avg bet year garage was built and year house was built
housing_training_data['GarageYrBlt'].fillna(housing_training_data['YearBuilt']+avg_yea
```

Out[11]:
```
count    1201.000000
mean       70.049958
std        24.284752
min        21.000000
25%        59.000000
50%        69.000000
75%        80.000000
max       313.000000
Name: LotFrontage, dtype: float64
```

We can see there are no more missing values in our original dataframe.

In [12]:
```
# check that there are no more missing values in the dataframe
null_count = housing_training_data.isnull().sum()
null_count[null_count != 0]
```

Out[12]:
```
Series([], dtype: int64)
```

We can also create boxplots for each of the continuous variables in the dataframe to analyze whether outliers exist for each of those variables.

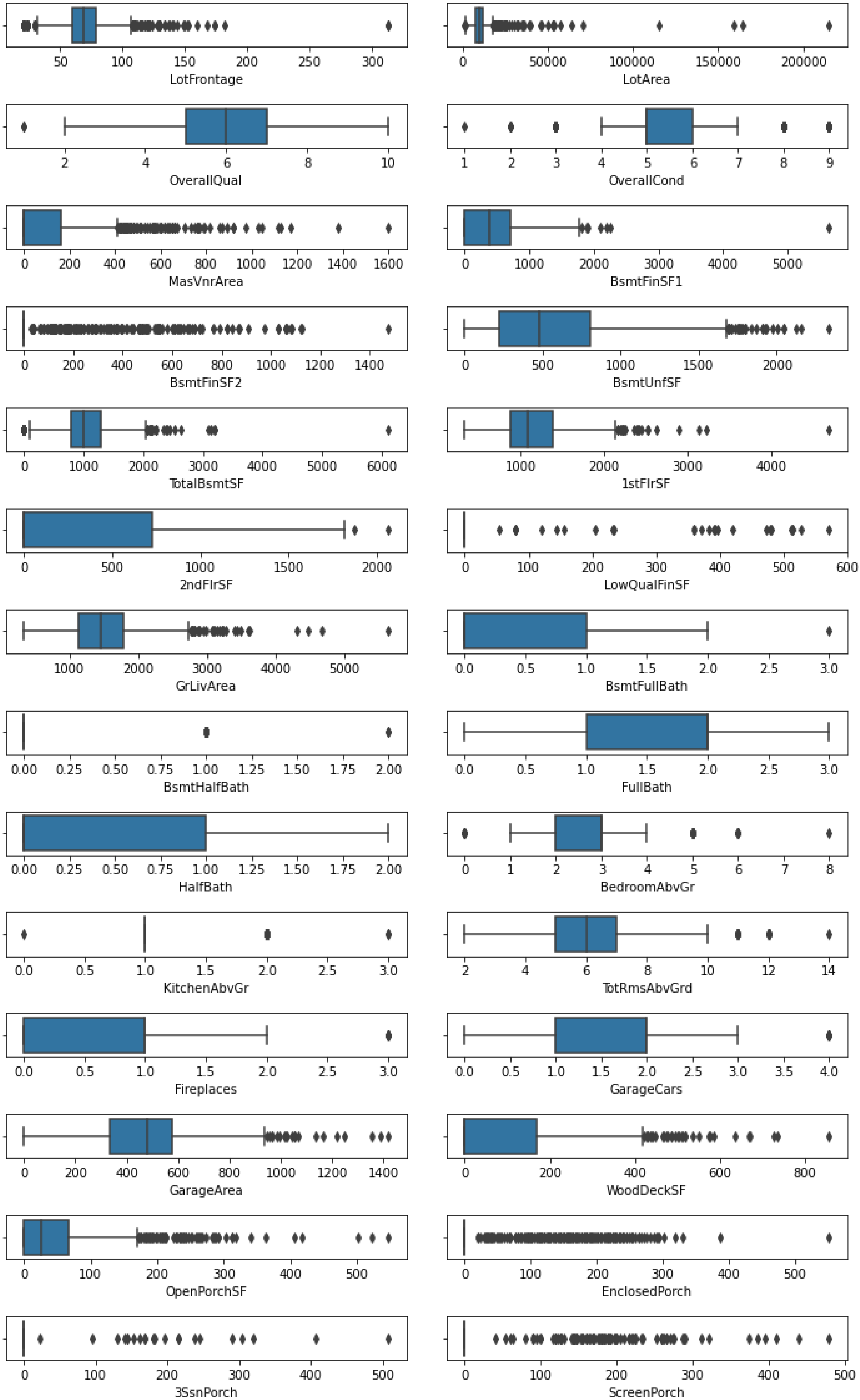
In [13]:
```
numerical_vars = ['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'MasVnrArea'
                  'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '
                  'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 
                  'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF
                  'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', '

fig, ax = plt.subplots(17, 2, figsize = (10, 19))

for var, subplot in zip(numerical_vars, ax.flatten()):
    sns.boxplot(x=housing_training_data[var], ax = subplot)

fig.tight_layout()
```

Out[13]:
```
<AxesSubplot:xlabel='LotFrontage'>
```

Out[13]:
```
<AxesSubplot:xlabel='LotArea'>
```

Out[13]:
```
<AxesSubplot:xlabel='OverallQual'>
```

```
Out[13]: <AxesSubplot:xlabel='OverallCond'>
Out[13]: <AxesSubplot:xlabel='MasVnrArea'>
Out[13]: <AxesSubplot:xlabel='BsmtFinSF1'>
Out[13]: <AxesSubplot:xlabel='BsmtFinSF2'>
Out[13]: <AxesSubplot:xlabel='BsmtUnfSF'>
Out[13]: <AxesSubplot:xlabel='TotalBsmtSF'>
Out[13]: <AxesSubplot:xlabel='1stFlrSF'>
Out[13]: <AxesSubplot:xlabel='2ndFlrSF'>
Out[13]: <AxesSubplot:xlabel='LowQualFinSF'>
Out[13]: <AxesSubplot:xlabel='GrLivArea'>
Out[13]: <AxesSubplot:xlabel='BsmtFullBath'>
Out[13]: <AxesSubplot:xlabel='BsmtHalfBath'>
Out[13]: <AxesSubplot:xlabel='FullBath'>
Out[13]: <AxesSubplot:xlabel='HalfBath'>
Out[13]: <AxesSubplot:xlabel='BedroomAbvGr'>
Out[13]: <AxesSubplot:xlabel='KitchenAbvGr'>
Out[13]: <AxesSubplot:xlabel='TotRmsAbvGrd'>
Out[13]: <AxesSubplot:xlabel='Fireplaces'>
Out[13]: <AxesSubplot:xlabel='GarageCars'>
Out[13]: <AxesSubplot:xlabel='GarageArea'>
Out[13]: <AxesSubplot:xlabel='WoodDeckSF'>
Out[13]: <AxesSubplot:xlabel='OpenPorchSF'>
Out[13]: <AxesSubplot:xlabel='EnclosedPorch'>
Out[13]: <AxesSubplot:xlabel='3SsnPorch'>
Out[13]: <AxesSubplot:xlabel='ScreenPorch'>
Out[13]: <AxesSubplot:xlabel='PoolArea'>
Out[13]: <AxesSubplot:xlabel='MiscVal'>
Out[13]: <AxesSubplot:xlabel='SalePrice'>
```

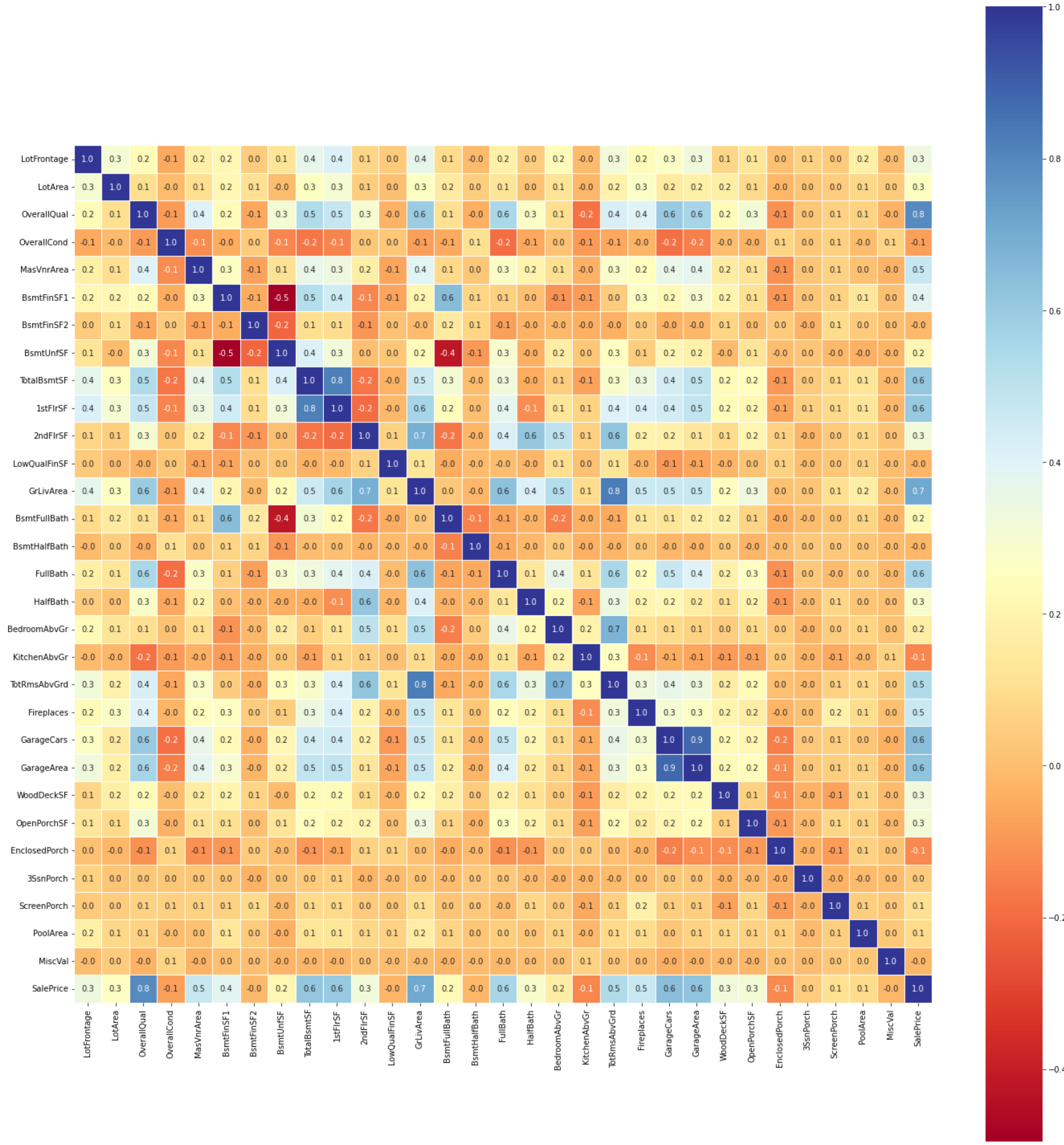

50 100 150 200 250 300
LotFrontage
0 50000 100000 150000 200000
LotArea
2 4 6 8 10
OverallQual
1 2 3 4 5 6 7 8 9
OverallCond
0 200 400 600 800 1000 1200 1400 1600
MasVnrArea
0 1000 2000 3000 4000 5000
BsmtFinSF1
0 200 400 600 800 1000 1200 1400
BsmtFinSF2
0 500 1000 1500 2000
BsmtUnfSF
0 1000 2000 3000 4000 5000 6000
TotalBsmtSF
1000 2000 3000 4000
1stFlrSF
0 500 1000 1500 2000
2ndFlrSF
0 100 200 300 400 500 600
LowQualFinSF
1000 2000 3000 4000 5000
GrLivArea
0.0 0.5 1.0 1.5 2.0 2.5 3.0
BsmtFullBath
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
BsmtHalfBath
0.0 0.5 1.0 1.5 2.0 2.5 3.0
FullBath
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
HalfBath
0 1 2 3 4 5 6 7 8
BedroomAbvGr
0.0 0.5 1.0 1.5 2.0 2.5 3.0
KitchenAbvGr
2 4 6 8 10 12 14
TotRmsAbvGrd
0.0 0.5 1.0 1.5 2.0 2.5 3.0
Fireplaces
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
GarageCars
0 200 400 600 800 1000 1200 1400
GarageArea
0 200 400 600 800
WoodDeckSF
0 100 200 300 400 500
OpenPorchSF
0 100 200 300 400 500
EnclosedPorch
0 100 200 300 400 500
3SsnPorch
0 100 200 300 400 500
ScreenPorch

Examination of the Relationship between the Dependent Variable and Potential Predictors

We can use a correlation heatmap to quantify the correlation between the dependent variable, sale price, and the potential continuous predictor variables.

In [14]:
```
df_corr_housing_training = housing_training_data[numerical_vars]
corrmat_housing_training = df_corr_housing_training.corr()

f, ax = plt.subplots(figsize = (20, 20))
sns.heatmap(corrmat_housing_training, vmax = 1, square = True, annot = True,
            cmap = 'RdYlBu', linewidths = 0.5, fmt=".1f")
```

Out[14]:
```
<AxesSubplot:>
```



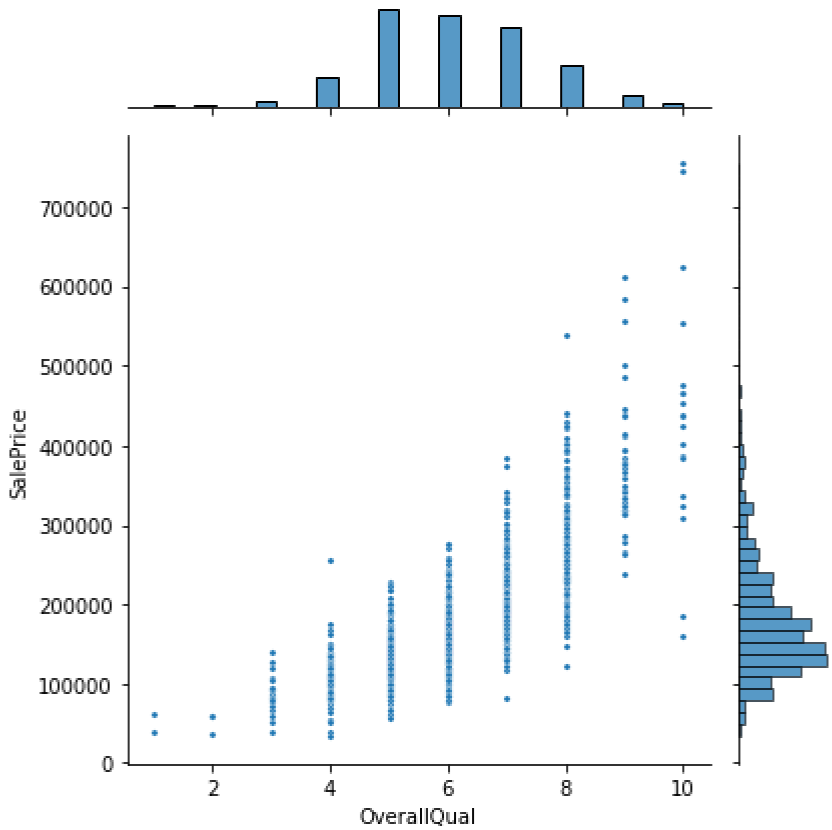
In [15]:
```python
#Correlation with output variable
cor_target = abs(cormat_housing_training["SalePrice"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.5]
relevant_features.sort_values(ascending=False)
```

Out[15]:
```
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
Name: SalePrice, dtype: float64
```

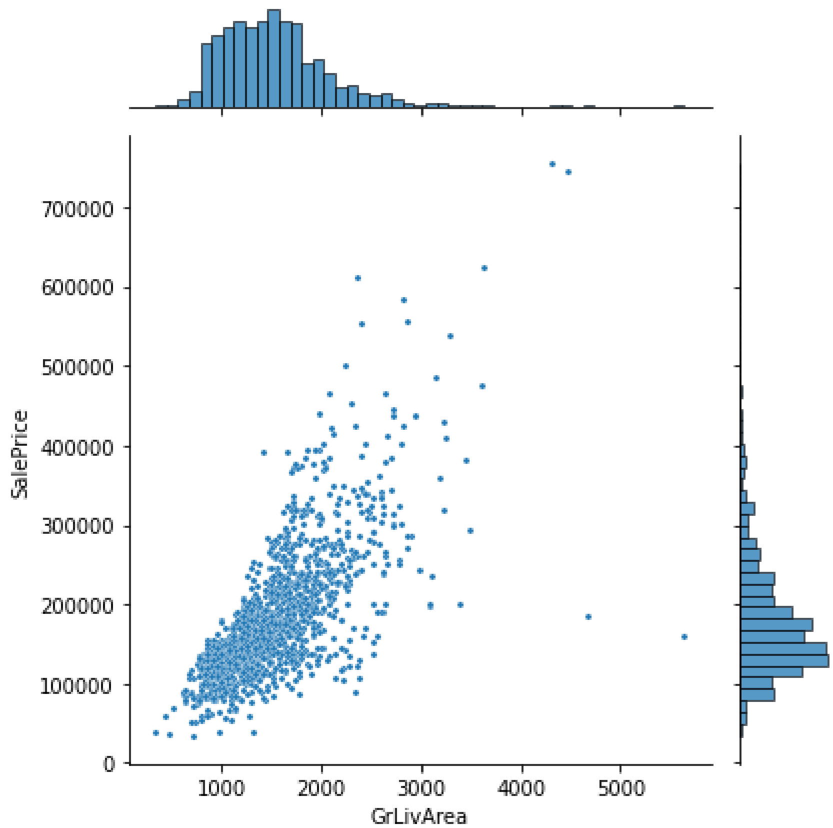
We can use jointplots to take a closer look at the relationship between sale price and five of the continuous variables with which sale price has a strong or moderate association: OverallQual, GrLivArea, GarageArea, Fullbath, and TotalBsmntSF.

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

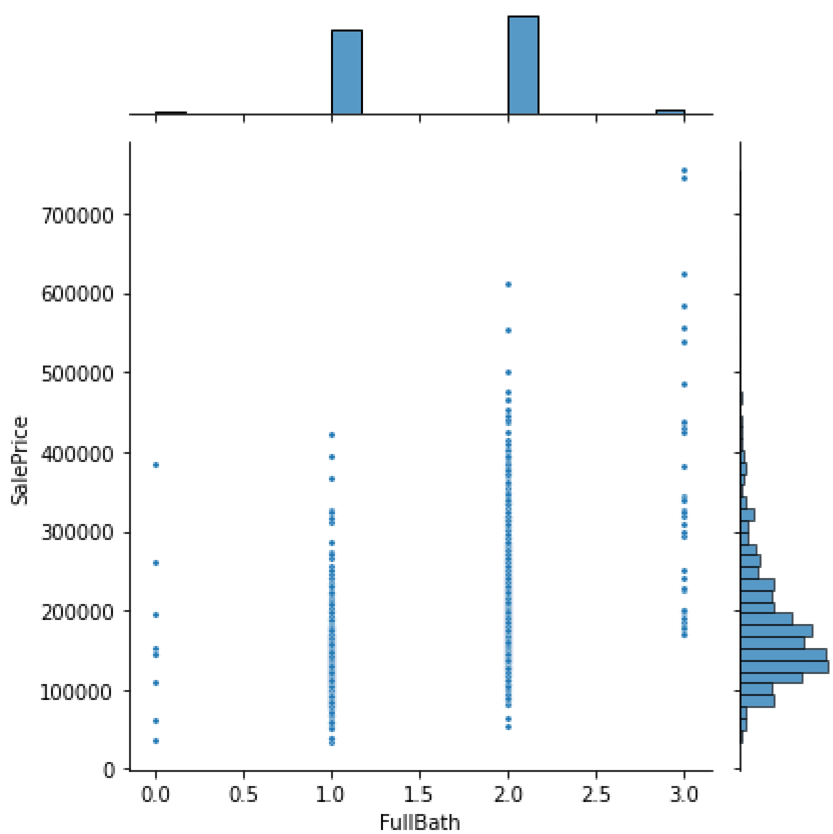
In [16]:
```
sns.jointplot(x='OverallQual', y='SalePrice', data = housing_training_data, joint_kws=
```



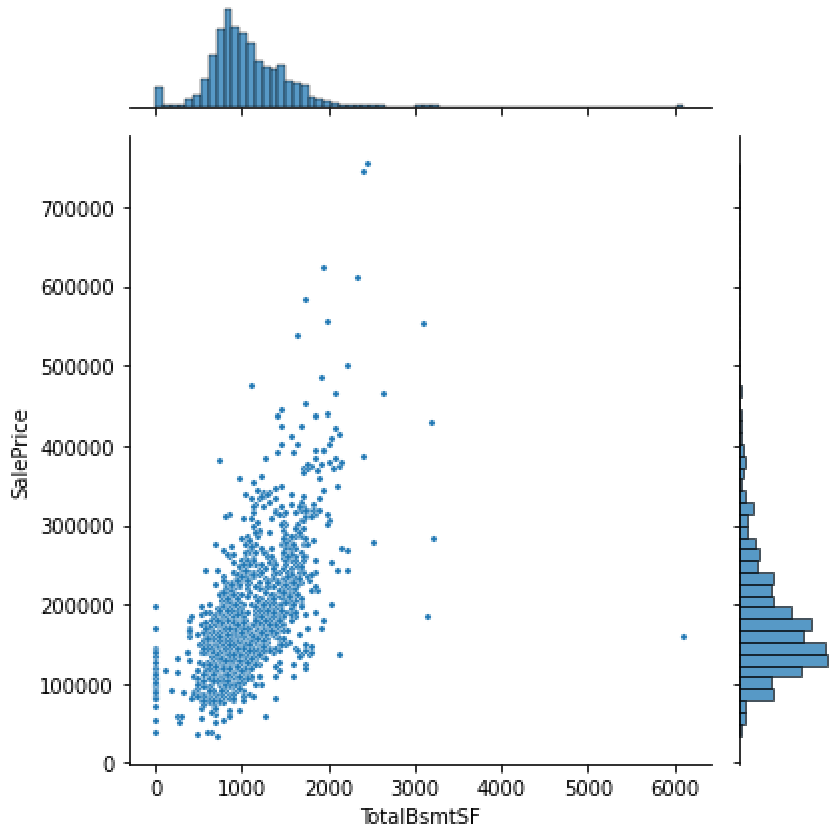
In [17]:
```
sns.jointplot(x='GrLivArea', y='SalePrice', data = housing_training_data, joint_kws={'
```



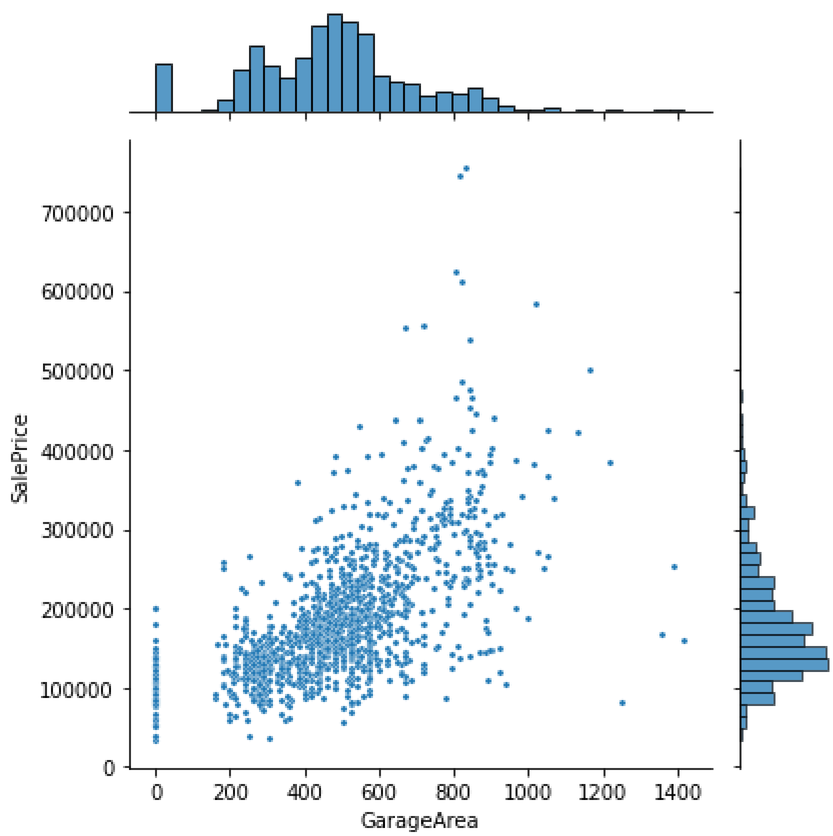
In [18]:
```
sns.jointplot(x='FullBath', y='SalePrice', data = housing_training_data, joint_kws={"s
```



In [19]:
```
sns.jointplot(x='TotalBsmtSF', y='SalePrice', data = housing_training_data, joint_kws=
```



In [20]:
```
sns.jointplot(x='GarageArea', y='SalePrice', data = housing_training_data, joint_kws=
```



To determine which binary categorical variables might serve as the best predictors in a regression model, we can create boxplots and run t-tests to help decipher which binary indicator variables may have the strongest relationship with home sale prices.

In [21]:
```
categorical_variables = ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
                         'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'House
                         'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'Exte
                         'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtF
                         'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'Fire
                         'GarageQual', 'GarageCond', 'PavedDrive',  'SaleType', 'SaleCo
                          'YrSold', 'MoSold']

category_counts = []

for var in categorical_variables:
    category_counts.append(len(housing_training_data[var].unique()))



categorical_variable_dictionary = {'Categorical Predictor':categorical_variables,'Numb
categorical_var_df = pd.DataFrame(categorical_variable_dictionary)
categorical_var_df


# Identify the Indicator Variables
indicator_variables_df = categorical_var_df[categorical_var_df['Number of Categories']
indicator_variables_df

# Identify the Non-Indicator Categorical Variables
non_indicator_categorial_vars_df = categorical_var_df[categorical_var_df['Number of Ca
non_indicator_categorial_vars_df
```

Out[21]:

	Categorical Predictor	Number of Categories
0	MSZoning	5
1	Street	2
2	LotShape	4
3	LandContour	4
4	Utilities	2
5	LotConfig	5
6	LandSlope	3
7	Neighborhood	25
8	Condition1	9
9	Condition2	8
10	BldgType	5
11	HouseStyle	8
12	RoofStyle	6
13	RoofMatl	8
14	Exterior1st	15
15	Exterior2nd	16
16	MasVnrType	4
17	ExterQual	4
18	ExterCond	5
19	Foundation	6
20	BsmtQual	5
21	BsmtCond	5
22	BsmtExposure	5
23	BsmtFinType1	7
24	BsmtFinType2	7
25	Heating	6
26	HeatingQC	5
27	CentralAir	2
28	Electrical	6
29	KitchenQual	4
30	Functional	7
31	FireplaceQu	6
32	GarageType	7
33	GarageQual	6

	Categorical Predictor	Number of Categories
34	GarageCond	6
35	PavedDrive	3
36	SaleType	9
37	SaleCondition	6
38	YearBuilt	112
39	GarageYrBlt	102
40	YrSold	5
41	MoSold	12

Out[21]:

	Categorical Predictor	Number of Categories
1	Street	2
4	Utilities	2
27	CentralAir	2

Out[21]:

	Categorical Predictor	Number of Categories
0	MSZoning	5
2	LotShape	4
3	LandContour	4
5	LotConfig	5
6	LandSlope	3
7	Neighborhood	25
8	Condition1	9
9	Condition2	8
10	BldgType	5
11	HouseStyle	8
12	RoofStyle	6
13	RoofMatl	8
14	Exterior1st	15
15	Exterior2nd	16
16	MasVnrType	4
17	ExterQual	4
18	ExterCond	5
19	Foundation	6
20	BsmtQual	5
21	BsmtCond	5
22	BsmtExposure	5
23	BsmtFinType1	7
24	BsmtFinType2	7
25	Heating	6
26	HeatingQC	5
28	Electrical	6
29	KitchenQual	4
30	Functional	7
31	FireplaceQu	6
32	GarageType	7
33	GarageQual	6
34	GarageCond	6
35	PavedDrive	3
36	SaleType	9

	Categorical Predictor	Number of Categories
37	SaleCondition	6
38	YearBuilt	112
39	GarageYrBlt	102
40	YrSold	5
41	MoSold	12

In [22]:
```
# convert Paved Drive to dichotomous, indicator variable
housing_training_data['PavedDrive'] = np.where(housing_training_data['PavedDrive'] ==
housing_training_data['PavedDrive'].value_counts()

# view indicator variables
indicator_vars = ['Street', 'Utilities', 'CentralAir','PavedDrive']

fig, ax = plt.subplots(1, 4, figsize=(15, 5))

for var, subplot in zip(indicator_vars, ax.flatten()):
        sns.boxplot(x = var, y = 'SalePrice', data=housing_training_data, ax=subplot)

fig.tight_layout()
```

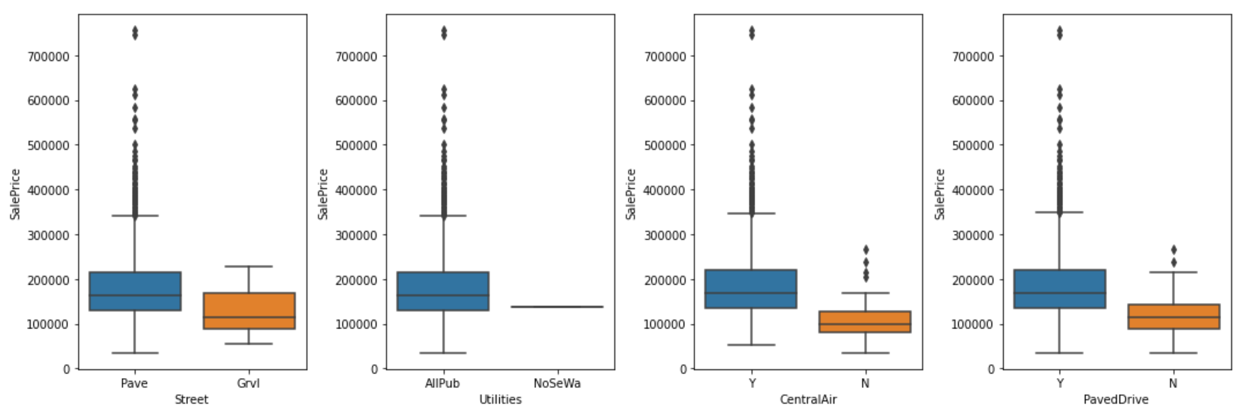
Out[22]:
```
Y    1340
N     120
Name: PavedDrive, dtype: int64
```

Out[22]: <AxesSubplot:xlabel='Street', ylabel='SalePrice'>

Out[22]: <AxesSubplot:xlabel='Utilities', ylabel='SalePrice'>

Out[22]: <AxesSubplot:xlabel='CentralAir', ylabel='SalePrice'>

Out[22]: <AxesSubplot:xlabel='PavedDrive', ylabel='SalePrice'>



In [23]:
```
# Run T-Tests To Determine Which Indicator Variables Might Have the Strongest Associat
from scipy.stats import ttest_ind

Street_t_test = ttest_ind(housing_training_data['SalePrice'][housing_training_data['St
         housing_training_data['SalePrice'][housing_training_data['Street'] == 'Grvl'
         equal_var=False)

Utilities_t_test = ttest_ind(housing_training_data['SalePrice'][housing_training_data[
         housing_training_data['SalePrice'][housing_training_data['Utilities'] == 'No
         equal_var=False)
```

```
Central_Air_t_test = ttest_ind(housing_training_data['SalePrice'][housing_training_dat
          housing_training_data['SalePrice'][housing_training_data['CentralAir'] == 'N
          equal_var=False)

Paved_Drive_t_test = ttest_ind(housing_training_data['SalePrice'][housing_training_dat
          housing_training_data['SalePrice'][housing_training_data['PavedDrive'] == 'N
          equal_var=False)



Indicator_Variable_t_test_statistics = [Street_t_test[0], Utilities_t_test[0], Central

Indicator_Variable_t_test_p_values = [Street_t_test[1], Utilities_t_test[1], Central_A

indicator_var_t_tests = {'Indicator Variable':indicator_vars,'T-Test Statistic':Indica
                         'P-Values':Indicator_Variable_t_test_p_values}
Indicator_var_t_test_df = pd.DataFrame(indicator_var_t_tests)
Indicator_var_t_test_df.style.background_gradient(cmap = 'Greens')
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/numpy/c
ore/fromnumeric.py:3724: RuntimeWarning: Degrees of freedom <= 0 for slice
  **kwargs)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/numpy/c
ore/_methods.py:254: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Out[23]:

	Indicator Variable	T-Test Statistic	P-Values
0	Street	1.900788	0.115048
1	Utilities	nan	nan
2	CentralAir	17.267773	0.000000
3	PavedDrive	15.093535	0.000000

We can see that Street, Central Air, and Paved Drive all have statistically significant t-test statistics. This tells us that there does appear to be statistically significant differences in Sale Prices between the categories of these variables. Let's dummy encode each one so that we can use them in our regression analysis.

In [ ]:

In [25]:

```
# dummy encode the Street, Central Air, Paved Drive indicator variables, we will exclu
housing_training_data = pd.get_dummies(housing_training_data, columns=['Street','Centr
```

To determine which categorical variables might be most useful for inclusion in a regression model (in the form of a dichotomous variable), we can create boxplots and run analyses of variance (ANOVA) to determine which non-binary categorical variables may have the strongest relationship with home sale prices.

We can create boxplots to visually display the distribution of sale prices disaggregated by the categories associated with each of the non-indicator categorical variables as well.

In [26]:
```
# redefine categorical_vars_df after transforming Paved Drive variable
non_indicator_categorial_vars_df = non_indicator_categorial_vars_df[non_indicator_cate

fig, ax = plt.subplots(33, 1, figsize=(15, 200))

for var, subplot in zip(non_indicator_categorial_vars_df['Categorical Predictor'], ax.
        sns.boxplot(x = var, y = 'SalePrice', data=housing_training_data, ax=subplot)

fig.tight_layout()
```

```
Out[26]: <AxesSubplot:xlabel='MSZoning', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='LotShape', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='LandContour', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='LotConfig', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='LandSlope', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Neighborhood', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Condition1', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Condition2', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BldgType', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='HouseStyle', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='RoofStyle', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='RoofMatl', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Exterior1st', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Exterior2nd', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='MasVnrType', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='ExterQual', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='ExterCond', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Foundation', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BsmtQual', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BsmtCond', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BsmtExposure', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BsmtFinType1', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='BsmtFinType2', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Heating', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='HeatingQC', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='Electrical', ylabel='SalePrice'>
Out[26]: <AxesSubplot:xlabel='KitchenQual', ylabel='SalePrice'>
```

Out[26]: <AxesSubplot:xlabel='Functional', ylabel='SalePrice'>

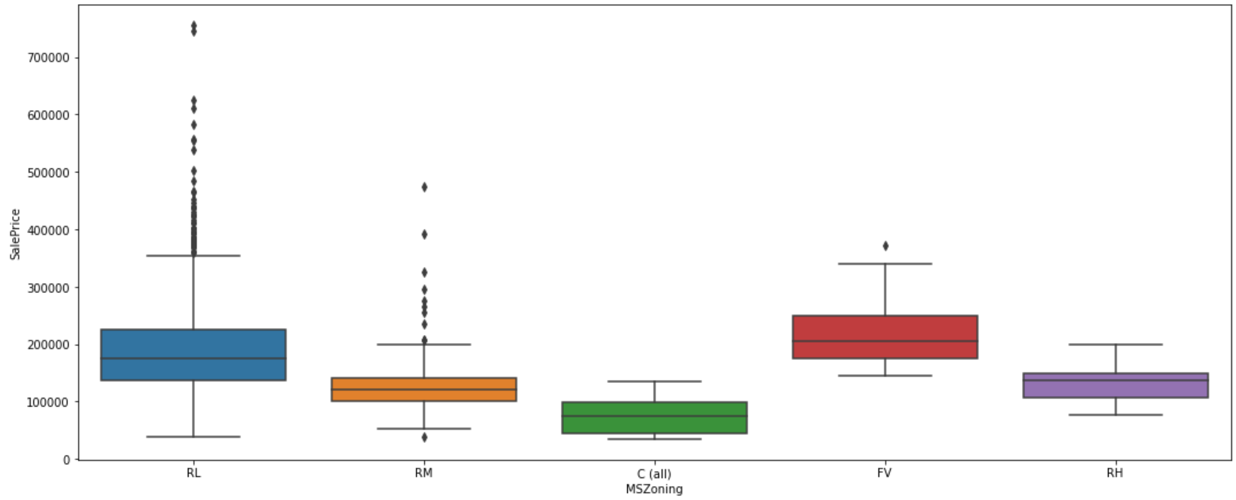
Out[26]: <AxesSubplot:xlabel='FireplaceQu', ylabel='SalePrice'>

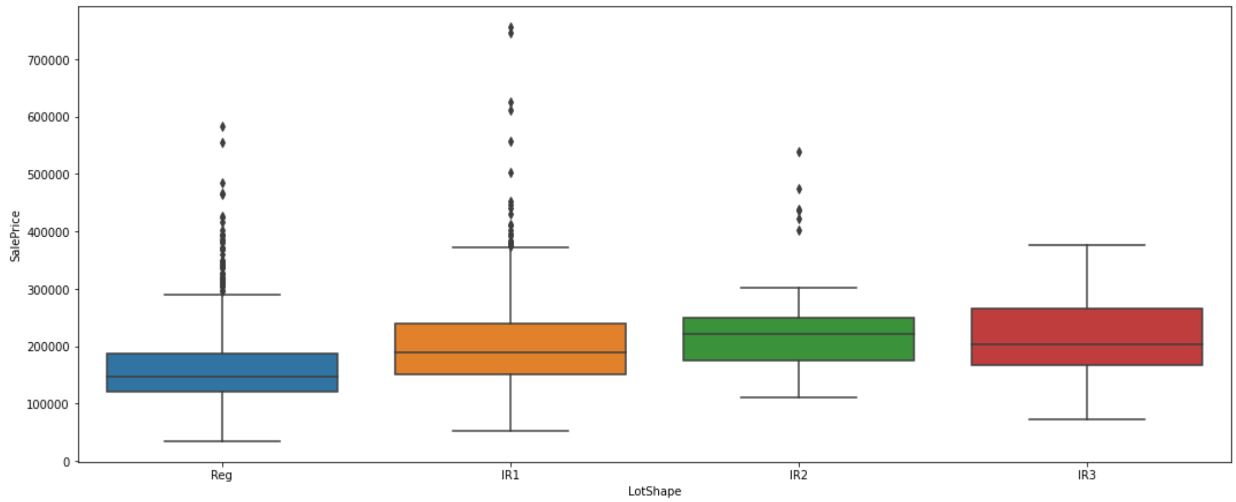
Out[26]: <AxesSubplot:xlabel='GarageType', ylabel='SalePrice'>

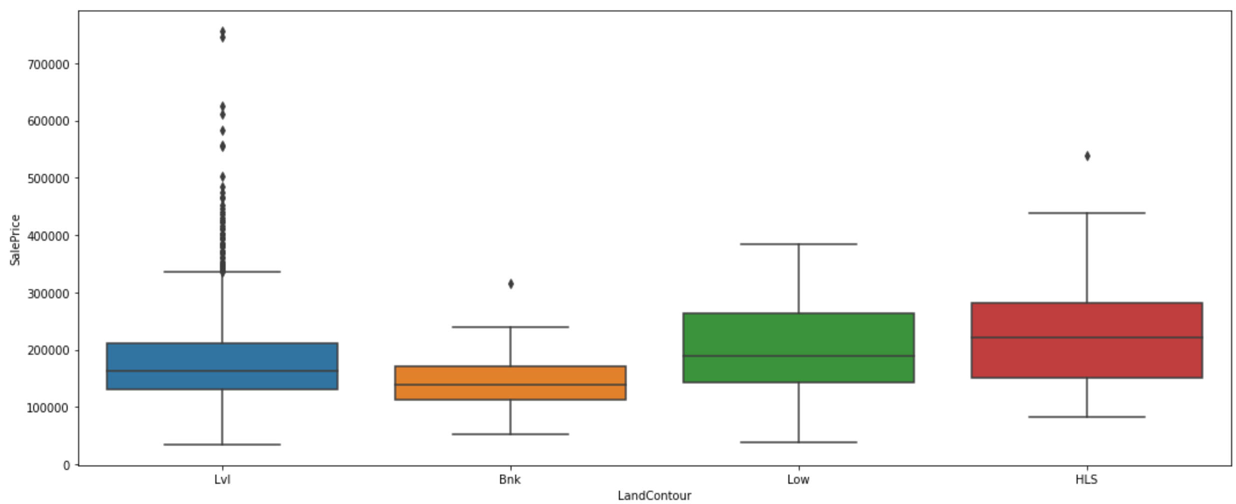
Out[26]: <AxesSubplot:xlabel='GarageQual', ylabel='SalePrice'>

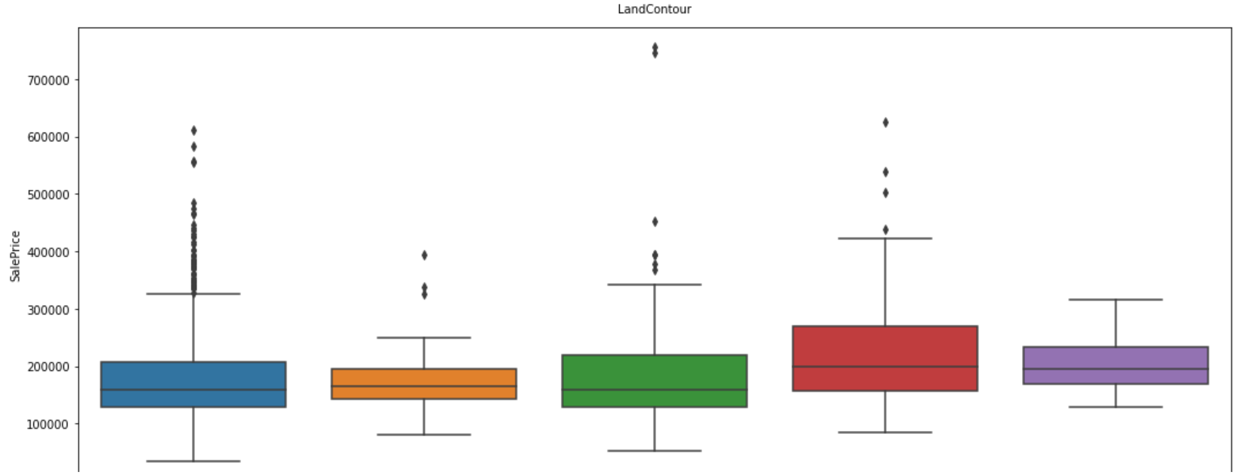
Out[26]: <AxesSubplot:xlabel='GarageCond', ylabel='SalePrice'>

Out[26]: <AxesSubplot:xlabel='SaleType', ylabel='SalePrice'>


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
RL
RM
C (all)
FV
RH
MSZoning


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
Reg
IR1
IR2
IR3
LotShape


SalePrice
700000
600000
500000
400000
300000
200000
100000
0
Lvl
Bnk
Low
HLS
LandContour


SalePrice
700000
600000
500000
400000
300000
200000
100000

Visual inspection of the boxplots above suggests that the variables for exterior quality, basement quality, fireplace quality, kitchen quality, garage quality and condition, andHeating quality may provide the most promise in our search for helpful categorical predictors that may be transformed into dichotomous variables. Conducting ANOVAs can shed more light on the relationship between these seven variables and home sale prices.

In [27]:
```
ANOVA_variables = ['ExterQual', 'BsmtQual', 'FireplaceQu', 'KitchenQual','GarageQual',

from scipy.stats import f_oneway

# ExterQual ANOVA
ExterQual_Gd = housing_training_data['SalePrice'][housing_training_data['ExterQual'] =
ExterQual_TA = housing_training_data['SalePrice'][housing_training_data['ExterQual'] =
ExterQual_Ex = housing_training_data['SalePrice'][housing_training_data['ExterQual'] =
ExterQual_Fa = housing_training_data['SalePrice'][housing_training_data['ExterQual'] =

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

# BsmtQual ANOVA
BsmtQual_Gd = housing_training_data['SalePrice'][housing_training_data['BsmtQual'] ==
BsmtQual_TA = housing_training_data['SalePrice'][housing_training_data['BsmtQual'] ==
BsmtQual_Ex = housing_training_data['SalePrice'][housing_training_data['BsmtQual'] ==
BsmtQual_None = housing_training_data['SalePrice'][housing_training_data['BsmtQual'] =
BsmtQual_Fa = housing_training_data['SalePrice'][housing_training_data['BsmtQual'] ==

ANOVA_BsmtQual = f_oneway(BsmtQual_Gd, BsmtQual_TA, BsmtQual_Ex, BsmtQual_None, BsmtQu

# FireplaceQu ANOVA
FireplaceQu_None = housing_training_data['SalePrice'][housing_training_data['Fireplace
```

```
FireplaceQu_TA = housing_training_data['SalePrice'][housing_training_data['FireplaceQu
FireplaceQu_Gd = housing_training_data['SalePrice'][housing_training_data['FireplaceQu
FireplaceQu_Fa = housing_training_data['SalePrice'][housing_training_data['FireplaceQu
FireplaceQu_Ex = housing_training_data['SalePrice'][housing_training_data['FireplaceQu
FireplaceQu_Po = housing_training_data['SalePrice'][housing_training_data['FireplaceQu

ANOVA_FireplaceQu = f_oneway(FireplaceQu_None, FireplaceQu_TA, FireplaceQu_Gd, Firepla
                            FireplaceQu_Ex, FireplaceQu_Po)

# KitchenQual ANOVA
KitchenQual_Gd = housing_training_data['SalePrice'][housing_training_data['KitchenQual
KitchenQual_TA = housing_training_data['SalePrice'][housing_training_data['KitchenQual
KitchenQual_Ex = housing_training_data['SalePrice'][housing_training_data['KitchenQual
KitchenQual_Fa = housing_training_data['SalePrice'][housing_training_data['KitchenQual

ANOVA_KitchenQual = f_oneway(KitchenQual_Gd, KitchenQual_TA, KitchenQual_Ex, KitchenQu

# GarageQual ANOVA
GarageQu_None = housing_training_data['SalePrice'][housing_training_data['GarageQual']
GarageQu_TA = housing_training_data['SalePrice'][housing_training_data['GarageQual'] =
GarageQu_Gd = housing_training_data['SalePrice'][housing_training_data['GarageQual'] =
GarageQu_Fa = housing_training_data['SalePrice'][housing_training_data['GarageQual'] =
GarageQu_Ex = housing_training_data['SalePrice'][housing_training_data['GarageQual'] =
GarageQu_Po = housing_training_data['SalePrice'][housing_training_data['GarageQual'] =

ANOVA_GarageQu = f_oneway(GarageQu_None, GarageQu_TA, GarageQu_Gd, GarageQu_Fa,
                            GarageQu_Ex, GarageQu_Po)

# GarageCond ANOVA
GarageCond_None = housing_training_data['SalePrice'][housing_training_data['GarageCond
GarageCond_TA = housing_training_data['SalePrice'][housing_training_data['GarageCond']
GarageCond_Gd = housing_training_data['SalePrice'][housing_training_data['GarageCond']
GarageCond_Fa = housing_training_data['SalePrice'][housing_training_data['GarageCond']
GarageCond_Ex = housing_training_data['SalePrice'][housing_training_data['GarageCond']
GarageCond_Po = housing_training_data['SalePrice'][housing_training_data['GarageCond']

ANOVA_GarageCond = f_oneway(GarageCond_None, GarageCond_TA, GarageCond_Gd, GarageCond_
                            GarageCond_Ex, GarageCond_Po)

# HeatingQC ANOVA
HeatingQC_TA = housing_training_data['SalePrice'][housing_training_data['HeatingQC'] =
HeatingQC_Gd = housing_training_data['SalePrice'][housing_training_data['HeatingQC'] =
HeatingQC_Fa = housing_training_data['SalePrice'][housing_training_data['HeatingQC'] =
HeatingQC_Ex = housing_training_data['SalePrice'][housing_training_data['HeatingQC'] =
HeatingQC_Po = housing_training_data['SalePrice'][housing_training_data['HeatingQC'] =

ANOVA_HeatingQC = f_oneway(HeatingQC_TA, HeatingQC_Gd, HeatingQC_Fa,
                            HeatingQC_Ex, HeatingQC_Po)

# Compile Outputs

ANOVA_statistics = [ANOVA_ExterQual[0], ANOVA_BsmtQual[0], ANOVA_FireplaceQu[0], ANOVA
                    ANOVA_GarageQu[0], ANOVA_GarageCond[0], ]

ANOVA_p_values = [ANOVA_ExterQual[1], ANOVA_BsmtQual[1], ANOVA_FireplaceQu[1], ANOVA_K
                  ANOVA_GarageQu[1], ANOVA_GarageCond[1]]

ANOVA_outputs = {'Categorical Variable': ANOVA_variables ,'Test Statistic': ANOVA_stat
                 'P-Values': ANOVA_p_values}
```

```
ANOVA_df = pd.DataFrame(ANOVA_outputs)
ANOVA_df.style.background_gradient(cmap = 'Greens')
```

Out[27]:

	Categorical Variable	Test Statistic	P-Values
0	ExterQual	443.334831	0.000000
1	BsmtQual	316.148635	0.000000
2	FireplaceQu	121.075121	0.000000
3	KitchenQual	407.806352	0.000000
4	GarageQual	88.394462	0.000000
5	GarageCond	25.776093	0.000000
6	HeatingQC	25.750153	0.000000

We will use the Tukey-Cramer Multiple Comparison Test to confirm whether there are statistically significant differences in means when considering pairwise comparisons of categorical variable values.

In [28]:

```
# lets view the tukeyhsd for each variable
from statsmodels.stats.multicomp import pairwise_tukeyhsd

for i in ANOVA_variables:
    tukey_cramer_result = pairwise_tukeyhsd(endog=housing_training_data['SalePrice'],
    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  0.001 -323897.1579 -234854.3366   True
    Ex     Gd -135727.4513  0.001 -157297.3415 -114157.5611   True
    Ex     TA -223019.6481  0.001 -244104.9402  -201934.356   True
    Fa     Gd   143648.296  0.001  103567.1096  183729.4823   True
    Fa     TA   56356.0992 0.0016    16533.607   96178.5913   True
    Gd     TA  -87292.1968  0.001  -95594.9096   -78989.484   True
-----------------------------------------------------------------
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
group1 group2   meandiff   p-adj     lower        upper     reject
-----------------------------------------------------------------
    Ex     Fa -211349.0128  0.001 -241850.1706 -180847.8549   True
    Ex     Gd -124352.5624  0.001 -140151.0688 -108554.0559   True
    Ex   None -221388.1494  0.001  -251243.057 -191533.2419   True
    Ex     TA -186281.2231  0.001 -202017.8288 -170544.6174   True
    Fa     Gd   86996.4504  0.001   59383.7383  114609.1625   True
    Fa   None  -10039.1367    0.9  -47511.5875   27433.3141  False
    Fa     TA   25067.7896  0.095    -2509.553   52645.1322  False
    Gd   None  -97035.5871  0.001 -123932.7383  -70138.4358   True
    Gd     TA  -61928.6608  0.001  -70860.7377  -52996.5838   True
  None     TA   35106.9263 0.0034    8246.0868   61967.7658   True
-----------------------------------------------------------------
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
group1 group2   meandiff   p-adj     lower        upper     reject
-----------------------------------------------------------------
    Ex     Fa -170414.0152  0.001 -221601.4617 -119226.5687   True
    Ex     Gd -111361.0842  0.001 -151519.9919  -71202.1765   True
    Ex   None -196381.0174  0.001 -236000.3876 -156761.6472   True
    Ex     Po   -207948.35  0.001 -265717.2832 -150179.4168   True
    Ex     TA -131989.0112  0.001 -172402.4484   -91575.574   True
    Fa     Gd   59052.9309  0.001   24425.9294   93679.9324   True
    Fa   None  -25967.0022 0.2479  -59966.7943    8032.7899  False
    Fa     Po  -37534.3348 0.3543  -91604.1139   16535.4442  False
    Fa     TA    38425.004 0.0213    3503.1301   73346.8778   True
    Gd   None  -85019.9332  0.001  -97208.8236  -72831.0427   True
    Gd     Po  -96587.2658  0.001 -140360.8217  -52813.7099   True
    Gd     TA   -20627.927  0.001  -35192.2827   -6063.5712   True
  None     Po  -11567.3326    0.9  -54846.4364   31711.7712  False
  None     TA   64392.0062  0.001   51389.0416   77394.9708   True
    Po     TA   75959.3388  0.001   31952.1549  119966.5227   True
-----------------------------------------------------------------
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
group1 group2   meandiff   p-adj     lower        upper     reject
-----------------------------------------------------------------
    Ex     Fa -222989.4649  0.001 -251454.2769 -194524.6528   True
    Ex     Gd -116438.6461  0.001 -132752.1208 -100125.1714   True
    Ex     TA -188592.1584  0.001 -204662.7844 -172521.5325   True
    Fa     Gd  106550.8188  0.001   81616.8092  131484.8283   True
    Fa     TA   34397.3064 0.0021    9621.5039    59173.109   True
    Gd     TA  -72153.5123  0.001  -80503.6215  -63803.4031   True
-----------------------------------------------------------------
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
================================================================
group1 group2   meandiff   p-adj     lower        upper    reject
```

```
-----------------------------------------------------------------
    Ex     Fa -117426.6458  0.101  -246946.454  12093.1623  False
    Ex     Gd  -25139.2857    0.9 -163601.7848 113323.2134  False
    Ex   None  -137682.716 0.0264 -265641.1324  -9724.2997   True
    Ex     Po -140833.3333 0.2107 -318533.0441  36866.3774  False
    Ex     TA   -53510.164 0.8078 -179306.5197  72286.1917  False
    Fa     Gd   92287.3601  0.001   26180.9523  158393.768   True
    Fa   None  -20256.0702 0.6669  -59898.8479  19386.7075  False
    Fa     Po  -23406.6875    0.9 -152926.4956 106113.1206  False
    Fa     TA   63916.4818  0.001   31933.4144  95899.5493   True
    Gd   None -112543.4303  0.001 -175535.7535 -49551.1071   True
    Gd     Po -115694.0476 0.1625 -254156.5467  22768.4515  False
    Gd     TA  -28370.8783 0.7094  -86846.5105  30104.7539  False
  None     Po   -3150.6173    0.9 -131109.0337 124807.7991  False
  None     TA   84172.5521  0.001   59254.8429 109090.2612   True
    Po     TA   87323.1693 0.3544  -38473.1863  213119.525  False
-----------------------------------------------------------------
      Multiple Comparison of Means - Tukey HSD, FWER=0.05
================================================================
group1 group2   meandiff  p-adj     lower        upper    reject
----------------------------------------------------------------
    Ex     Fa  -9345.9714    0.9 -167580.7655 148888.8227  False
    Ex     Gd     55930.0    0.9  -114211.491  226071.491  False
    Ex   None  -20682.716    0.9 -176469.8843 135104.4522  False
    Ex     Po    -15500.0    0.9 -190004.8015 159004.8015  False
    Ex     TA  63885.7353 0.8252  -90129.0525 217900.5231  False
    Fa     Gd  65275.9714 0.1987  -16067.2196 146619.1625  False
    Fa   None -11336.7446    0.9  -55362.0733  32688.5841  False
    Fa     Po  -6154.0286    0.9  -96267.9206  83959.8634  False
    Fa     TA  73231.7067  0.001   35960.5027 110502.9107   True
    Gd   None  -76612.716 0.0493 -153085.6293   -139.8028   True
    Gd     Po    -71430.0 0.4306 -181113.1364  38253.1364  False
    Gd     TA   7955.7353    0.9  -64838.6294     80750.1  False
  None     Po    5182.716    0.9  -80560.5359   90925.968  False
  None     TA  84568.4513  0.001    59657.924 109478.9787   True
    Po     TA  79385.7353 0.0672   -3093.4641 161864.9347  False
----------------------------------------------------------------
       Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================================
group1 group2   meandiff  p-adj     lower        upper    reject
-----------------------------------------------------------------
    Ex     Fa  -90994.9394  0.001 -119740.4887   -62249.39   True
    Ex     Gd  -58055.5578  0.001  -72506.7051 -43604.4104   True
    Ex     Po -127914.4291 0.3799 -322924.5088  67095.6505  False
    Ex     TA   -72551.553  0.001  -84383.0712 -60720.0347   True
    Fa     Gd   32939.3816 0.0271    2400.2524  63478.5107   True
    Fa     Po  -36919.4898    0.9 -233776.6307 159937.6511  False
    Fa     TA   18443.3864 0.4273  -10946.8769  47833.6496  False
    Gd     Po  -69858.8714 0.8508 -265141.3923 125423.6495  False
    Gd     TA  -14495.9952 0.0864  -30190.4728   1198.4824  False
    Po     TA   55362.8762    0.9 -139743.2799 250469.0322  False
-----------------------------------------------------------------
```

Given that the ANOVA and Tukey-Cramer tests the levels of these variables appear to be statistically significant from one another (for the most part). This demonstrates that it makes since to ordinally encode these with HeatingQC as the exception.

This also reveals that sale prices for rows with 'None' is not statistically different from sale prices of rows with 'FA'. For each variable, so we will encode 'None' with the same value as 'FA'.

In [29]:
```
# for heatingQC excellent is the only value that is statistically significant from the
housing_training_data['HeatingEx'] = np.where(housing_training_data['HeatingQC'] == 'E
housing_training_data.drop(columns=['HeatingQC'],inplace=True)
```

Encode important categorical variables

In [30]:
```
from sklearn.preprocessing import LabelEncoder
# let's ordinally encode the other variables we explored above (aside from heatingQC w
important_categorical = ['ExterQual', 'BsmtQual', 'FireplaceQu', 'KitchenQual', 'Garag

ordinal_mapping = {
    'Ex': 4,
    'Gd': 3,
    'TA': 2,
    'Fa': 1,
    'None':1,
    'Po': 0,
}

# process columns, replace to categorical features with ordinal ranking
for i in important_categorical:
    housing_training_data[i] = housing_training_data[i].replace(ordinal_mapping)

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

```
Shape all_data: (1460, 77)
```

Feature Creation

New features may enable us to create more accurate prediction models for home sale prices. Accordingly, we will create a feature to reflect the number of years since a home has been remodeled.

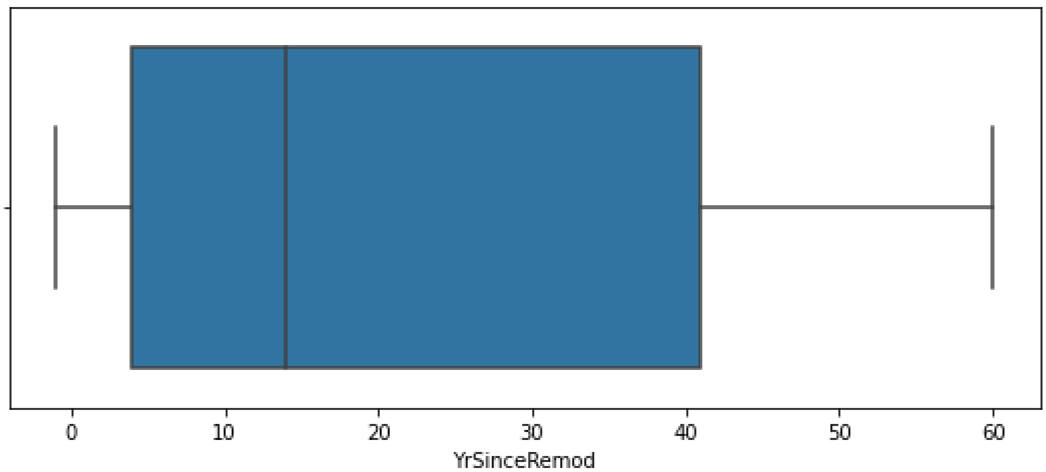
In [31]:
```
# create new variable, years since the house has been remodeled from selling date (use
housing_training_data['YrSinceRemod'] = housing_training_data['YrSold'] - housing_trai
housing_training_data['YrSinceRemod'].describe()
# create boxplot of YrSinceRemod
sns.boxplot(x = 'YrSinceRemod', data=housing_training_data)

#Pearson correlation coefficient and p value for sale price and GarageArea (Size of ga
res6 = stats.pearsonr(housing_training_data.YrSinceRemod, housing_training_data.SalePr
print("Pearson correlation coefficient and p value for sale price and Years since Hous
res6
```

Out[31]:
```
count    1460.000000
mean       22.950000
std        20.640653
min        -1.000000
25%         4.000000
50%        14.000000
75%        41.000000
max        60.000000
Name: YrSinceRemod, dtype: float64
```

Out[31]:
```
<AxesSubplot:xlabel='YrSinceRemod'>
```

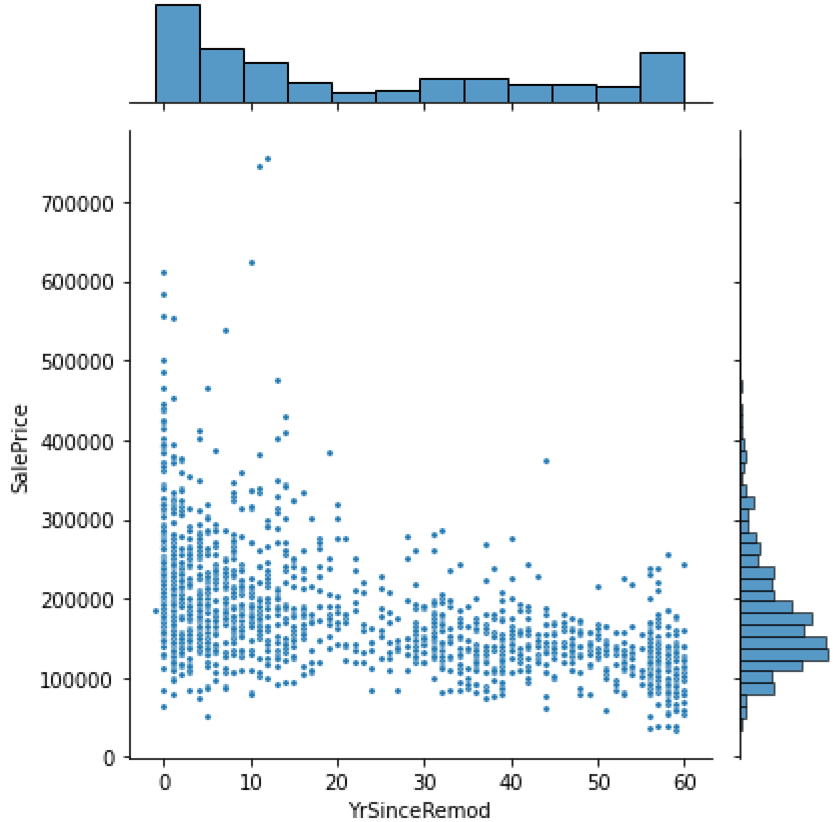
```
Pearson correlation coefficient and p value for sale price and Years since House was
remodeled/built:
```

Out[31]:
```
(-0.509078738015629, 4.3748554463775595e-97)
```



We can create a scatterplot to visualize the relationship between years since remodel and sale price.

In [32]:
```
sns.jointplot(x='YrSinceRemod', y='SalePrice', data = housing_training_data, joint_kws
```

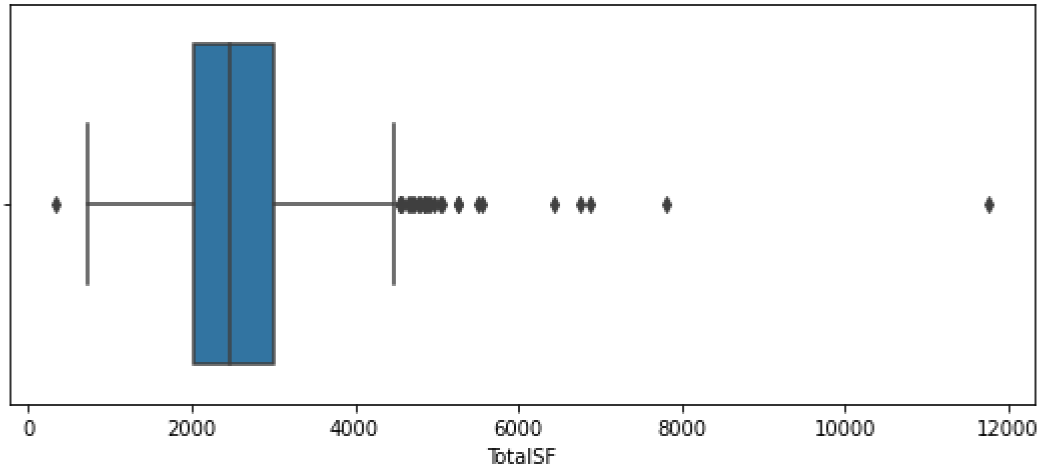


We will also create a feature to reflect the number of total square feet in a home.

In [33]:
```
# create new variable TotalSF
housing_training_data['TotalSF'] = housing_training_data['TotalBsmtSF'] + housing_trai
housing_training_data['TotalSF'].describe()
# create boxplot of TotalSF
sns.boxplot(x = 'TotalSF', data=housing_training_data)
```

Out[33]:
```
count     1460.000000
mean      2572.893151
std        823.598492
min        334.000000
25%       2014.000000
50%       2479.000000
75%       3008.500000
max      11752.000000
Name: TotalSF, dtype: float64
```

Out[33]:
```
<AxesSubplot:xlabel='TotalSF'>
```

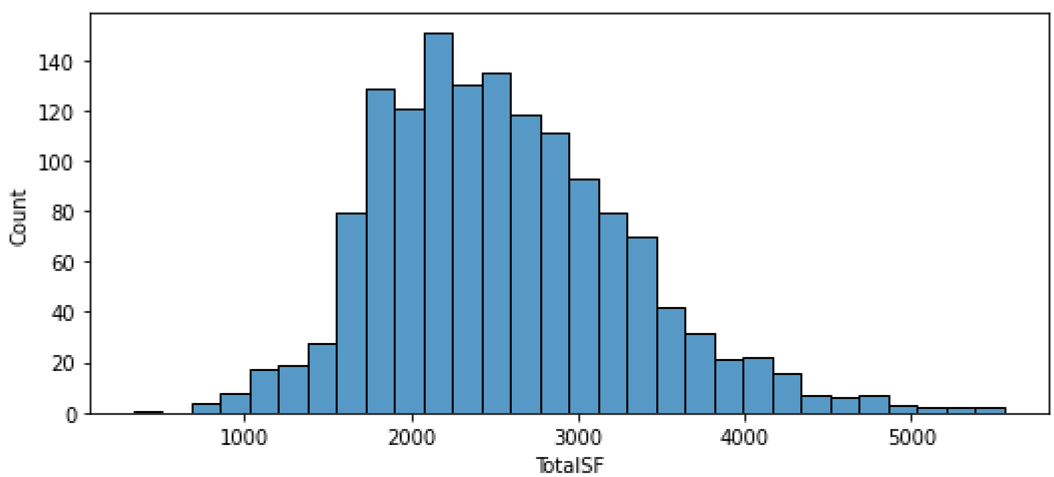


In [34]:
```
# drop large outlier from the dataframe
housing_training_data.drop(housing_training_data[housing_training_data['TotalSF'] > 60
# visualize distribution without extreme outliers
sns.histplot(data=housing_training_data, x="TotalSF")

#Pearson correlation coefficient and p value for sale price and TotalSF):
res7 = stats.pearsonr(housing_training_data.TotalSF, housing_training_data.SalePrice)
print("Pearson correlation coefficient and p value for sale price and TotalSF (Total s
res7
```

Out[34]:
```
<AxesSubplot:xlabel='TotalSF', ylabel='Count'>
```

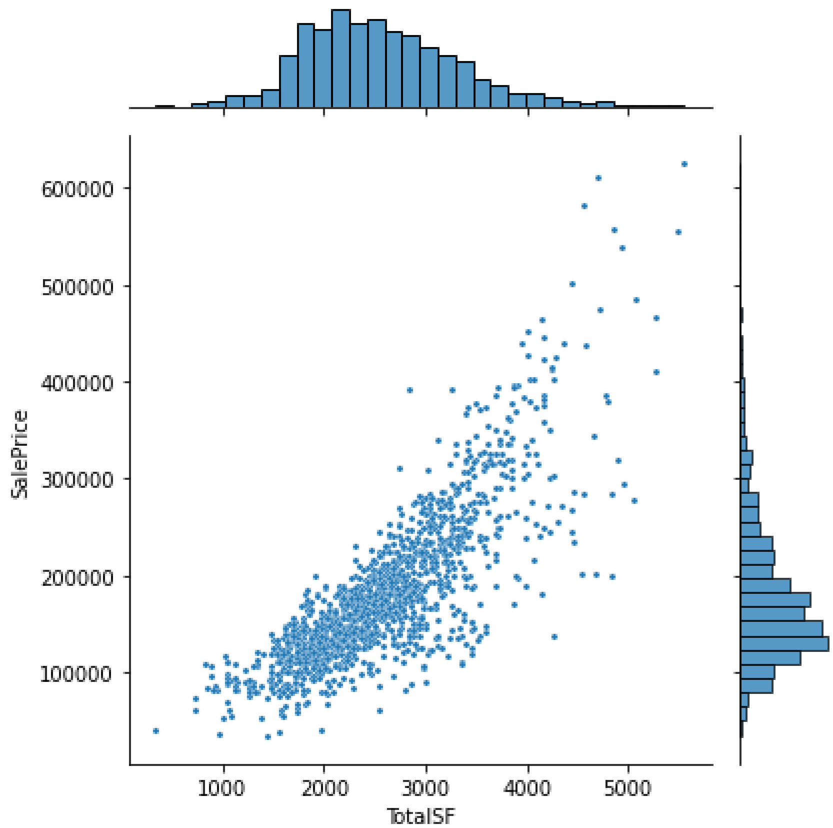
```
Pearson correlation coefficient and p value for sale price and TotalSF (Total square
feet - includes basement):
```

Out[34]:
```
(0.81999629129728, 0.0)
```



We can create a scatterplot to examine the relationship between total square feet and sale price.

In [35]:
```
sns.jointplot(x='TotalSF', y='SalePrice', data = housing_training_data, joint_kws={"s'
```



Dummy Encoding

Let's dummy encode the remaining categorical variables and create a new dataframe with these encoded columns and the original numeric columns.

In [36]:
```
# create new df with current numeric columns
housing_training_numeric_df = pd.DataFrame(housing_training_data.select_dtypes(exclude

# create new df with current categorical columns
housing_training_object_df = pd.DataFrame(housing_training_data.select_dtypes(exclude=
orig_object_cols = housing_training_object_df.columns

# dummy encode the categorical columns
housing_training_data_dummy = pd.get_dummies(housing_training_object_df, columns=orig_

# create new df with original numeric and new dummy encoded columns
housing_training_data_large = pd.concat([housing_training_numeric_df, housing_training
housing_training_data_large
```

Out[36]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1	60	65.0	8450	7	5	2003	2003
1	2	20	80.0	9600	6	8	1976	1976
2	3	60	68.0	11250	7	5	2001	2002
3	4	70	60.0	9550	7	5	1915	1970
4	5	60	84.0	14260	8	5	2000	2000
...	...	...	...	...	...	...	...	...
1455	1456	60	62.0	7917	6	5	1999	2000
1456	1457	20	85.0	13175	6	6	1978	1988
1457	1458	70	66.0	9042	7	9	1941	2006
1458	1459	20	68.0	9717	5	6	1950	1996
1459	1460	20	75.0	9937	5	6	1965	1965

1455 rows × 256 columns

In [37]:

```
# Correlation with Sale Price
cormat_housing_training = housing_training_data_large.corr()
cor_target = abs(cormat_housing_training["SalePrice"])

# Selecting correlated features
relevant_features = cor_target[cor_target>0.25]

# display features that have correlations with SalePrice over 0.25
with pd.option_context("display.max_rows",300):
    relevant_features.sort_values(ascending=False)

# create new dataframe with only those features
feature_list = relevant_features.index
housing_training_data_large_subset = housing_training_data_large[feature_list]
housing_training_data_large_subset.shape
```

Out[37]:
```
SalePrice                1.000000
TotalSF                  0.819996
OverallQual              0.801196
GrLivArea                0.718340
ExterQual                0.695051
KitchenQual              0.667111
BsmtQual                 0.657389
GarageCars               0.650951
TotalBsmtSF              0.643275
GarageArea               0.638538
1stFlrSF                 0.621678
FullBath                 0.556516
YearBuilt                0.535781
TotRmsAbvGrd             0.535425
FireplaceQu              0.525552
YrSinceRemod             0.524075
YearRemodAdd             0.522479
GarageYrBlt              0.516351
Foundation_PConc         0.505411
MasVnrArea               0.476732
Fireplaces               0.467166
HeatingEx                0.440949
BsmtFinType1_GLQ         0.434806
GarageFinish_Fin         0.422119
Neighborhood_NridgHt     0.421602
GarageFinish_Unf         0.417580
BsmtFinSF1               0.393687
MasVnrType_None          0.383184
SaleType_New             0.379298
SaleCondition_Partial    0.373552
GarageType_Detchd        0.361482
MasVnrType_Stone         0.350892
Foundation_CBlock        0.346991
GarageType_Attchd        0.340156
LotFrontage              0.339451
OpenPorchSF              0.330605
Exterior2nd_VinylSd      0.328221
Exterior1st_VinylSd      0.326943
WoodDeckSF               0.320869
BsmtExposure_Gd          0.309613
2ndFlrSF                 0.300248
MSZoning_RM              0.294802
HalfBath                 0.285725
Neighborhood_NoRidge     0.280726
GarageQual               0.280230
LotArea                  0.270350
LotShape_Reg             0.263206
BsmtExposure_No          0.262934
SaleType_WD              0.258981
CentralAir_Y             0.258415
GarageCond               0.258367
Name: SalePrice, dtype: float64
```

Out[37]:
```
(1455, 51)
```

We may use this dataframe later on in the analysis. For the start, we will investigate the relationship between SalePrice and a select number of features, like Total Square Feet.

Prepare Test Data

In [38]:
```
# load test data
housing_testing_data = pd.read_csv('test.csv')
```

Handle Null values, matches how we dealt with Nulls in the training dataset

In [39]:
```
# 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

# show null counts, percentage of null values, and column type for columns with more t
null_summary = pd.concat([null_count, null_percentage, column_type], axis=1, keys=['Mi
null_summary_only_missing = null_summary[null_count != 0].sort_values('Percentage Miss
null_summary_only_missing
```

Out[39]:

	Missing Count	Percentage Missing	Column Type
PoolQC	1456	99.794380	object
MiscFeature	1408	96.504455	object
Alley	1352	92.666210	object
Fence	1169	80.123372	object
FireplaceQu	730	50.034270	object
LotFrontage	227	15.558602	float64
GarageCond	78	5.346127	object
GarageYrBlt	78	5.346127	float64
GarageQual	78	5.346127	object
GarageFinish	78	5.346127	object
GarageType	76	5.209047	object
BsmtCond	45	3.084304	object
BsmtExposure	44	3.015764	object
BsmtQual	44	3.015764	object
BsmtFinType1	42	2.878684	object
BsmtFinType2	42	2.878684	object
MasVnrType	16	1.096642	object
MasVnrArea	15	1.028101	float64
MSZoning	4	0.274160	object
BsmtFullBath	2	0.137080	float64
BsmtHalfBath	2	0.137080	float64
Functional	2	0.137080	object
Utilities	2	0.137080	object
GarageCars	1	0.068540	float64
GarageArea	1	0.068540	float64
TotalBsmtSF	1	0.068540	float64
KitchenQual	1	0.068540	object
BsmtUnfSF	1	0.068540	float64
BsmtFinSF2	1	0.068540	float64
BsmtFinSF1	1	0.068540	float64
Exterior2nd	1	0.068540	object
Exterior1st	1	0.068540	object
SaleType	1	0.068540	object

In [40]:
```
# PoolQC, MiscFeature, Alley, Fence all have over 50% of missing values, we will remov
housing_testing_data.drop(['Alley','PoolQC','Fence','MiscFeature'],axis=1,inplace=True

columns_None = ['SaleType','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','Ga
# set Nulls in non-numeric columns to 'None'
housing_testing_data[columns_None] = housing_testing_data[columns_None].fillna('None')
```

In [41]:
```
# change Null values to 0 for the following variables
columns_zero = ['MasVnrArea','GarageArea','GarageCars','TotalBsmtSF','BsmtUnfSF','Bsmt
housing_testing_data[columns_zero] = housing_testing_data[columns_zero].fillna(0)


# fill Nulls for Lot Frontage with median value
housing_testing_data['LotFrontage'].fillna(housing_testing_data['LotFrontage'].median(


# fill Nulls with year garage was built with median value
housing_testing_data['GarageYrBlt'].fillna(housing_testing_data['GarageYrBlt'].median(
```

In [42]:
```
# convert Paved Drive to dichotomous, indicator variable
housing_testing_data['PavedDrive'] = np.where(housing_testing_data['PavedDrive'] == 'Y
housing_testing_data['PavedDrive'].value_counts()

# dummy encode the Street, Central Air, Paved Drive indicator variables, we will exclu
housing_testing_data = pd.get_dummies(housing_testing_data, columns=['Street','Central
```

Out[42]:
```
Y    1301
N     158
Name: PavedDrive, dtype: int64
```

In [43]:
```
# for heatingQC encode this to be a binary variable to match how we encoded this colum
housing_testing_data['HeatingEx'] = np.where(housing_testing_data['HeatingQC'] == 'Ex'
housing_testing_data.drop(columns=['HeatingQC'],inplace=True)
```

In [44]:
```
# encode categorical columns with ordinal values
important_categorical = ['ExterQual', 'BsmtQual', 'FireplaceQu', 'KitchenQual', 'Garag

ordinal_mapping = {
    'Ex': 4,
    'Gd': 3,
    'TA': 2,
    'Fa': 1,
    'None': 1,
    'Po': 0,
}

# process columns, replace to categorical features with ordinal ranking
for i in important_categorical:
    housing_testing_data[i] = housing_testing_data[i].replace(ordinal_mapping)

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

```
Shape all_data: (1459, 76)
```

In [45]:
```
# create new variable TotalSF
housing_testing_data['TotalSF'] = housing_testing_data['TotalBsmtSF'] + housing_testir
```

```
# create new variable, years since the house has been remodeled from selling date (use
housing_testing_data['YrSinceRemod'] = housing_testing_data['YrSold'] - housing_testir
```

In [46]:
```
### Dummy Encoding
```

In [47]:
```
# create new df with current numeric columns
housing_testing_numeric_data = pd.DataFrame(housing_testing_data.select_dtypes(exclude

# create new df with current categorical columns
housing_testing_object_data = pd.DataFrame(housing_testing_data.select_dtypes(exclude=
orig_object_cols = housing_testing_object_data.columns

# dummy encode the categorical columns
housing_testing_data_dummy = pd.get_dummies(housing_testing_object_data, columns=orig_

# create new df with original numeric and new dummy encoded columns
housing_testing_data_large = pd.concat([housing_testing_numeric_data, housing_testing_
housing_testing_data_large
```

Out[47]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1461	20	80.0	11622	5	6	1961	1961
1	1462	20	81.0	14267	6	6	1958	1958
2	1463	60	74.0	13830	5	5	1997	1998
3	1464	60	78.0	9978	6	6	1998	1998
4	1465	120	43.0	5005	8	5	1992	1992
...	...	...	...	...	...	...	...	...
1454	2915	160	21.0	1936	4	7	1970	1970
1455	2916	160	21.0	1894	4	5	1970	1970
1456	2917	20	160.0	20000	5	7	1960	1996
1457	2918	85	62.0	10441	5	5	1992	1992
1458	2919	60	74.0	9627	7	5	1993	1994

1459 rows × 246 columns

In [48]:
```
# create new dataframe with correlated variables with SalePrice observed in the traini
feature_list = feature_list.drop('SalePrice')
housing_testing_data_large_subset = housing_testing_data_large[feature_list]
housing_testing_data_large_subset
```

Out[48]:

	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	ExterQual	BsmtQua
0	80.0	11622	5	1961	1961	0.0	2	
1	81.0	14267	6	1958	1958	108.0	2	
2	74.0	13830	5	1997	1998	0.0	2	
3	78.0	9978	6	1998	1998	20.0	2	
4	43.0	5005	8	1992	1992	0.0	3	
...	...	...	...	...	...	...	...	
1454	21.0	1936	4	1970	1970	0.0	2	
1455	21.0	1894	4	1970	1970	0.0	2	
1456	160.0	20000	5	1960	1996	0.0	2	
1457	62.0	10441	5	1992	1992	0.0	2	
1458	74.0	9627	7	1993	1994	94.0	2	

1459 rows × 50 columns

In [49]:

```
# Find the common columns
common_columns = housing_testing_data_large.columns.intersection(housing_training_data

# Keep only the common columns in both DataFrames
housing_testing_data_large_common = housing_testing_data_large[common_columns]
housing_training_data_large_common = housing_training_data_large[common_columns]
```

In [50]:

```
housing_testing_data_large_common
housing_training_data_large_common
common_columns
```

Out[50]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1461	20	80.0	11622	5	6	1961	1961
1	1462	20	81.0	14267	6	6	1958	1958
2	1463	60	74.0	13830	5	5	1997	1998
3	1464	60	78.0	9978	6	6	1998	1998
4	1465	120	43.0	5005	8	5	1992	1992
...	...	...	...	...	...	...	...	...
1454	2915	160	21.0	1936	4	7	1970	1970
1455	2916	160	21.0	1894	4	5	1970	1970
1456	2917	20	160.0	20000	5	7	1960	1996
1457	2918	85	62.0	10441	5	5	1992	1992
1458	2919	60	74.0	9627	7	5	1993	1994

1459 rows × 240 columns

Out[50]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1	60	65.0	8450	7	5	2003	2003
1	2	20	80.0	9600	6	8	1976	1976
2	3	60	68.0	11250	7	5	2001	2002
3	4	70	60.0	9550	7	5	1915	1970
4	5	60	84.0	14260	8	5	2000	2000
...	...	...	...	...	...	...	...	...
1455	1456	60	62.0	7917	6	5	1999	2000
1456	1457	20	85.0	13175	6	6	1978	1988
1457	1458	70	66.0	9042	7	9	1941	2006
1458	1459	20	68.0	9717	5	6	1950	1996
1459	1460	20	75.0	9937	5	6	1965	1965

1455 rows × 240 columns

Out[50]:

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'ExterQual',
       ...
       'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth', 'SaleType_WD',
       'SaleCondition_Abnorml', 'SaleCondition_AdjLand',
       'SaleCondition_Alloca', 'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=240)
```

In [ ]:

Constructing Models to Predict Home Prices

Model Assumptions

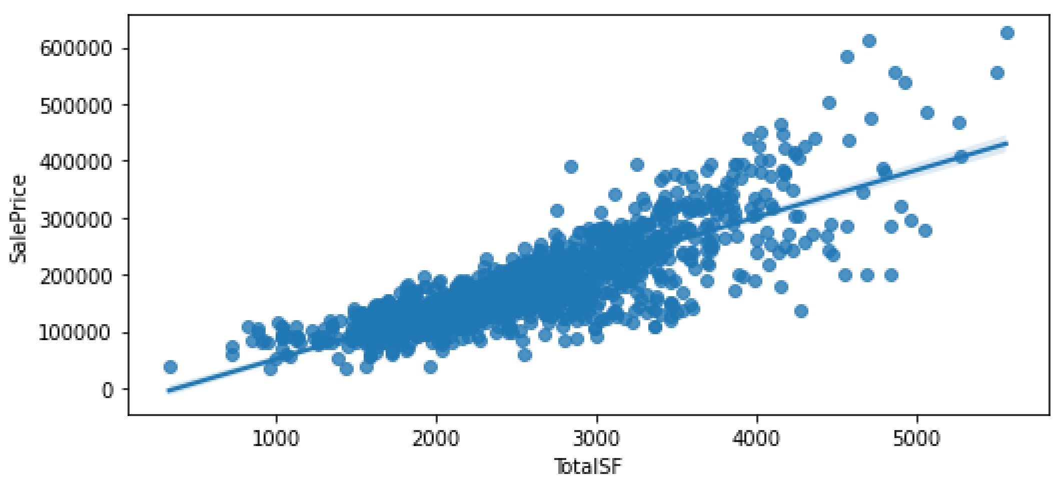
1. Linearity
2. Homoscedasticity
3. Independence of Errors
4. Multivariate Normality
5. No or little Multicollinearity

Below are simple and multiple regressions that examine the associations between variables of interest and sale price.

In [51]:
```
import numpy as np
import statsmodels.api as sm
# New feature is highly correlated, lets 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)
```

```
                            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, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:31   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 spec
ified.
[2] The condition number is large, 9.41e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

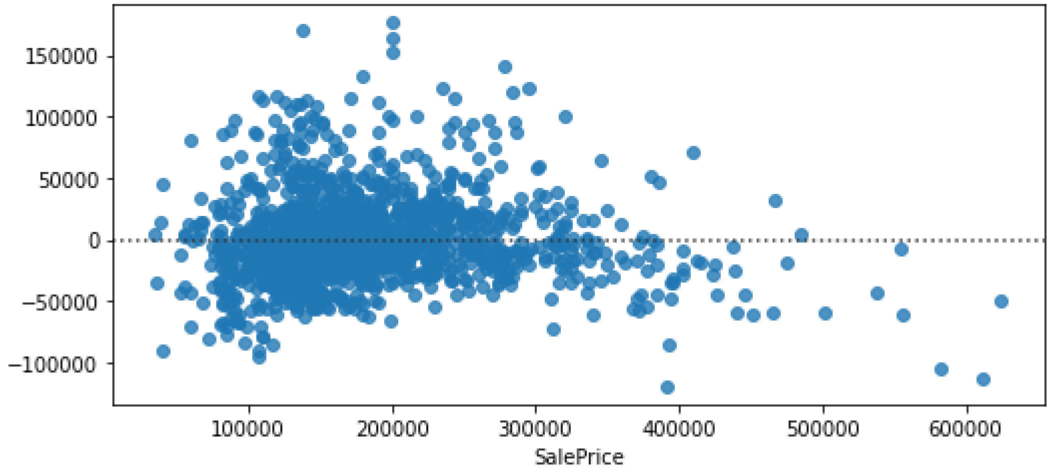
Out[51]: <AxesSubplot:xlabel='TotalSF', ylabel='SalePrice'>



In [52]:

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

Out[52]: <AxesSubplot:xlabel='SalePrice'>



The residual plot shows evidence of Heteroscedasticity since the residuals are not evenly scattered. For higher Sales Prices, the residuals are negative indicating that the model is over estimating homes with higher Sales Prices. There is evidence that this model violates the linearity assumption, Homoscedasticity assumption and the Independence of Errors assumption.

In [53]:

```
# qqplot
import matplotlib.pyplot as plt
stats.probplot(y-y_pred, dist="norm", plot=plt)
plt.title("Normal QQ Plot")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.show()
```

Out[53]:

```
((array([-3.30417817, -3.04690148, -2.90382339, ...,  2.90382339,
          3.04690148,  3.30417817]),
  array([-184865.74057368, -169571.36528068, -156517.81407152, ...,
          195404.36999739,  236556.60682101,  253807.82492125])),
 (42839.88750440423, 1.5086045390486717e-10, 0.9775030165443651))
```

Out[53]:

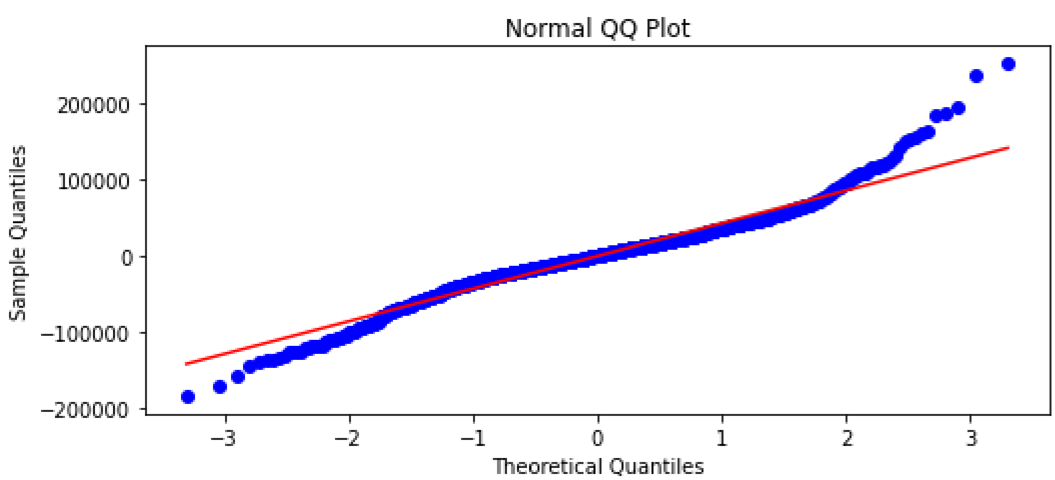
```
Text(0.5, 1.0, 'Normal QQ Plot')
```

Out[53]:

```
Text(0.5, 0, 'Theoretical Quantiles')
```

Out[53]:

```
Text(0, 0.5, 'Sample Quantiles')
```



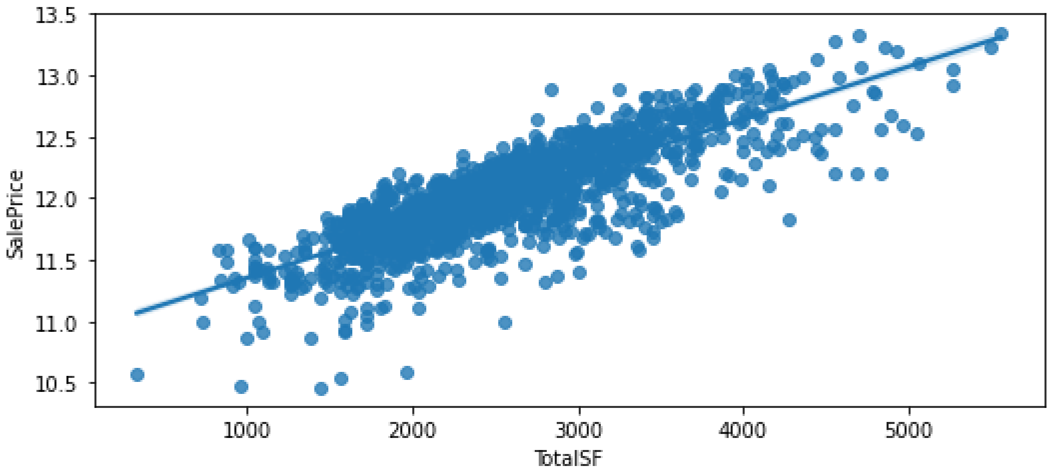
The tails of the distribution deviate from the qqline indicating that the errors are not Normally distributed.

Let's try transforming the independent variable, Sales Price since we saw earlier in this analysis that this helped to Normalize its distribution.

In [54]:
```
# 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)
```

```
                            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, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:32   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 spec
ified.
[2] The condition number is large, 9.41e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Out[54]:
```
<AxesSubplot:xlabel='TotalSF', ylabel='SalePrice'>
```

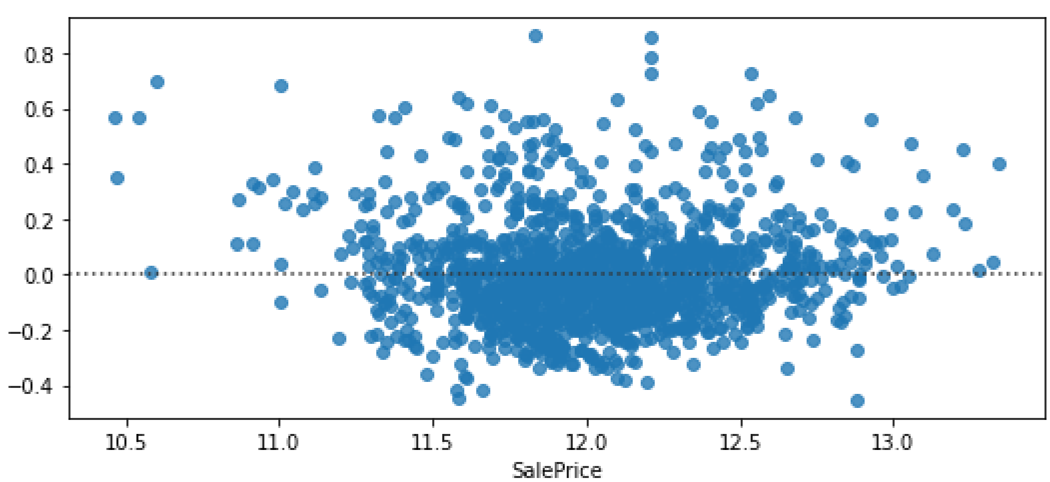


The relationship between log(Sale Price) and TotalSF appears to be linear. This meets the Linearity assumption.

An r-squared value of 0.669 means that the model explains 66.9 percent of the variance in the dependent variable. The omnibus test, however, indicates that residuals are not normally distributed and the high kurtosis value indicates that the distribution of the residuals are more peaked than a normal distribution. The condition number is 1.12e+04, which is quite high and suggests that there may be multicollinearity in the model.

In [55]:
```
# plot the residuals
y_pred=model.predict(X)
sns.residplot(x=y_log, y=y_pred)
```

Out[55]: <AxesSubplot:xlabel='SalePrice'>



The residuals appear to be more randomly scattered across values of Sales Price. This appears to better meet the Homoscedasticity assumption and the Independence of Errors assumptions than the model with the untransformed Sales Price.

In [56]:
```
# qqplot
stats.probplot(y_log-y_pred, dist="norm", plot=plt)
plt.title("Normal QQ Plot")
plt.xlabel("Theoretical Quantiles")
```

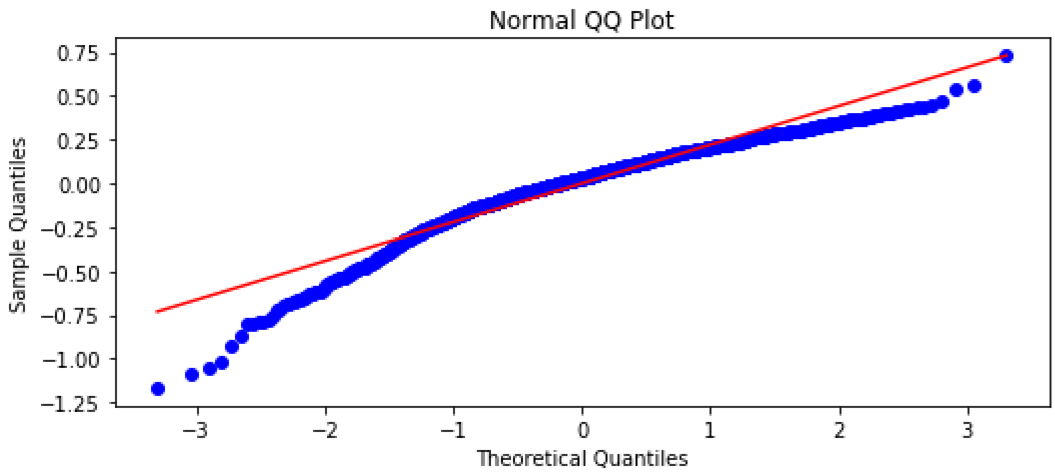
```
plt.ylabel("Sample Quantiles")
plt.show()
```

Out[56]:
```
((array([-3.30417817, -3.04690148, -2.90382339, ...,  2.90382339,
          3.04690148,  3.30417817]),
  array([-1.17224411, -1.08303201, -1.05546774, ...,  0.54315541,
          0.55983944,  0.73613196])),
 (0.22111127940895553, 6.253817917920981e-15, 0.9700026442288759))
```

Out[56]:
```
Text(0.5, 1.0, 'Normal QQ Plot')
```

Out[56]:
```
Text(0.5, 0, 'Theoretical Quantiles')
```

Out[56]:
```
Text(0, 0.5, 'Sample Quantiles')
```



The tails of the distribution still stray from the qqline, indicating the distribution of the errors is not Normal.

We can see the assumptions of a linear model are still not met by transforming Sales Price, lets try building a polynomial model.

Polynomial Regression

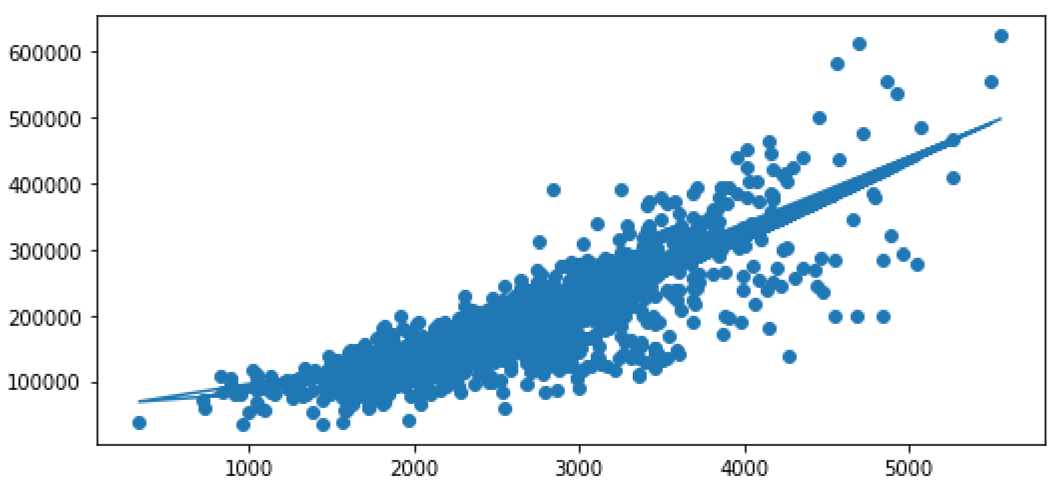
Third Order Polynomial Regression with Total SF as a predictor of Sales Price

In [57]:
```
from sklearn.preprocessing import PolynomialFeatures
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)
```

Out[57]:
```
(1455, 4)

                            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, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:34   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 spec
ified.
[2] The condition number is large, 5.7e+11. This might indicate that there are
strong multicollinearity or other numerical problems.
```

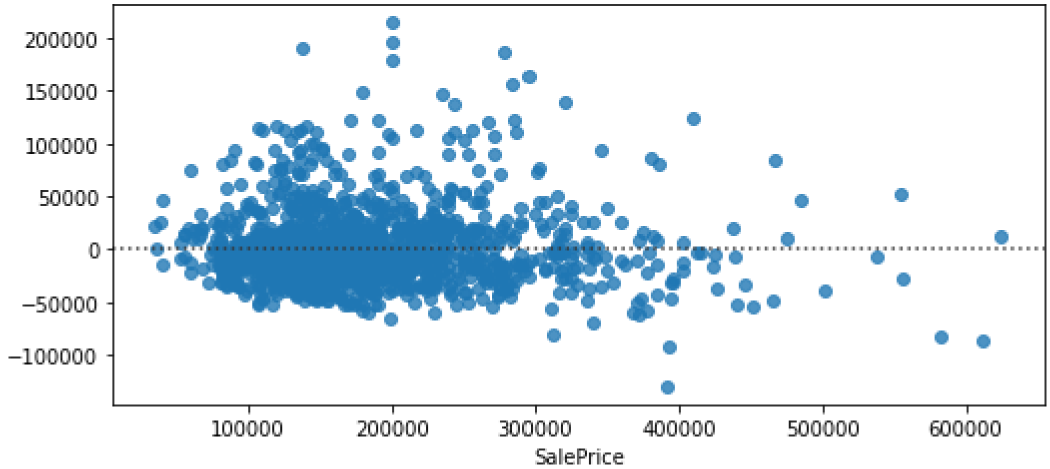
Out[57]: <matplotlib.collections.PathCollection at 0x7fee806f4950>

Out[57]: [<matplotlib.lines.Line2D at 0x7fee8085ae10>]



In [58]:
```
# plot the residuals
sns.residplot(x=y, y=y_pred)
```

Out[58]: <AxesSubplot:xlabel='SalePrice'>



The residuals are fairly scattered across Sales Prices. Let's try adding more predictors to a linear regression model.

Multiple Linear Regression

Check the correlation between the two new variables. If they are highly correlated we won't construct a mutliple linear regression model with the both of those variables as predictors.

In [59]:
```
df_corr_mult_lreg = housing_training_data[['TotalSF','YrSinceRemod']]
df_corr_mult_lreg.corr()
```

Out[59]:

	TotalSF	YrSinceRemod
TotalSF	1.000000	-0.352279
YrSinceRemod	-0.352279	1.000000

In [ ]:
```
```

TotalSF and YrSinceRemod are not highly correlated so we will construct a multiple linear regression model using the two variables as predictors of the log transformed Sales Price.

In [60]:
```
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())
```

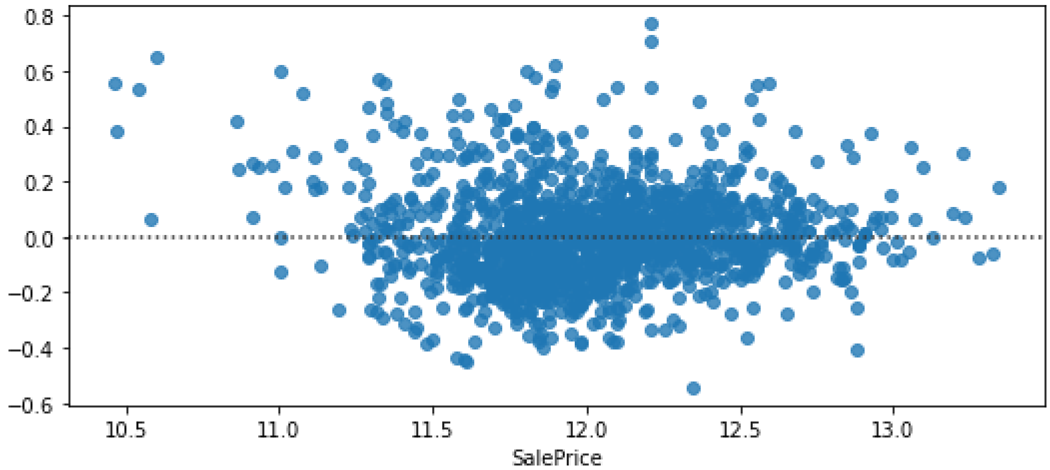
```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.761
Model:                            OLS   Adj. R-squared:                  0.760
Method:                 Least Squares   F-statistic:                     2308.
Date:                Sun, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:34   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 spec
ified.
[2] The condition number is large, 1.15e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

An r-squared value of 0.765 means that the model explains 76.5 percent of the variance in the dependent variable. The adjusted R-squared value is about the same as the r-squared value, indicating that we aren't overfitting the model by addding multiple variables. The omnibus test, however, indicates that residuals are not normally distributed and the high kurtosis value indicates that the distribution of the residuals are more peaked than a normal distribution. The condition number is 1.12e+04, which is quite high and suggests that there may be multicollinearity in the model.

In [61]:

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

Out[61]: <AxesSubplot:xlabel='SalePrice'>



The residuals appear to be more randomly scattered across values of Sales Price. This appears to better meet the Homoscedasticity assumption and the Independence of Errors assumptions than the model with the untransformed Sales Price. But as we mentioned above, the high omnibus value indicates the distribution of the residuals is not normally distributed.

Piecewise Regression

We can try fitting a linear regression model of sale price using a piecewise regression model.

In [62]:
```
import pwlf

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)

piecewise_regression_output = housing_training_data[["TotalSF"]]
piecewise_regression_output['Log Sale Price'] = y.tolist()
piecewise_regression_output['Predicted Log Sale Price'] = yHat.tolist()
piecewise_regression_output['residual'] = piecewise_regression_output['Log Sale Price'
piecewise_regression_output['squared residuals'] = piecewise_regression_output['residu

RMSE = (piecewise_regression_output['squared residuals'].sum() / len(piecewise_regress
print(f"The Root Mean Squared Error of this piecewise regression model is {RMSE}.")

correlation = piecewise_regression_output['Log Sale Price'].corr(piecewise_regression_
print(f"The correlation of this piecewise regreession model is {correlation}.")
```

```
The Root Mean Squared Error of this piecewise regression model is 0.2253851335019151
6.
The correlation of this piecewise regreession model is 0.8215282922354664.
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
```

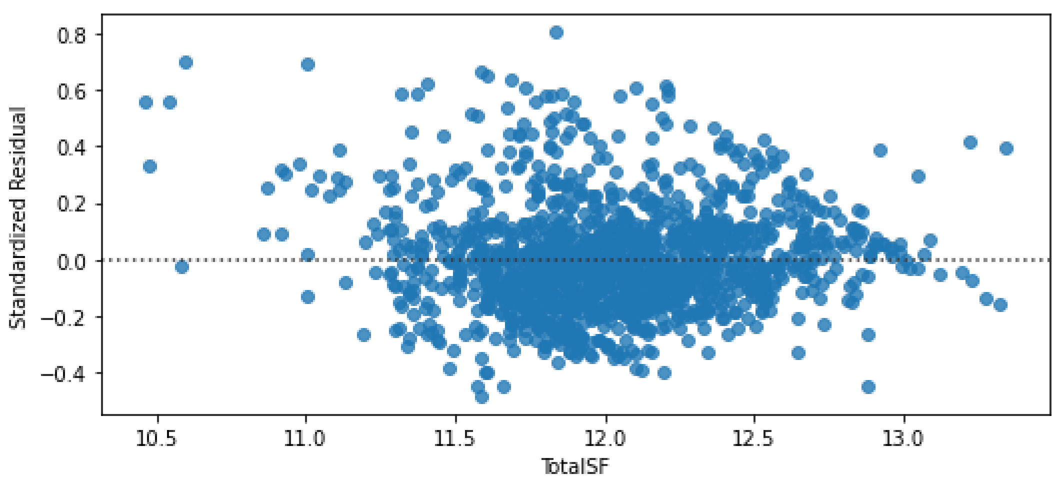
We can visualize the residuals plot to examine whether the regression model may violate any of the regression assumptions.

In [63]:
```
sns.residplot(x=y, y=yHat)
plt.ylabel('Standardized Residual')
plt.xlabel('TotalSF')
```

Out[63]:
```
<AxesSubplot:>
```

Out[63]:
```
Text(0, 0.5, 'Standardized Residual')
```

Out[63]:
```
Text(0.5, 0, 'TotalSF')
```



We can visualize a Q-Q plot to determine whether this regression model violates the linear regression assumption that residuals are noramlly distributed.

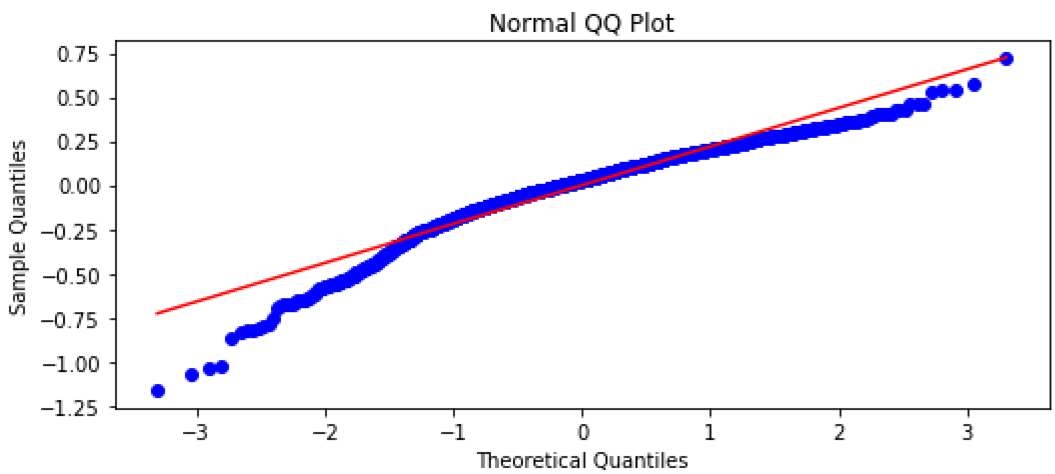
In [64]:
```
# qqplot
stats.probplot(y - yHat, dist="norm", plot=plt)
plt.title("Normal QQ Plot")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.show()
```

Out[64]:
```
((array([-3.30417817, -3.04690148, -2.90382339, ...,  2.90382339,
          3.04690148,  3.30417817]),
  array([-1.1651156 , -1.0640493 , -1.0393697 , ...,  0.54546573,
          0.57744714,  0.72360866])),
 (0.2193429300027091, 3.1475258810892375e-15, 0.9712466424655569))
```

Out[64]:
```
Text(0.5, 1.0, 'Normal QQ Plot')
```

Out[64]:
```
Text(0.5, 0, 'Theoretical Quantiles')
```

Out[64]:
```
Text(0, 0.5, 'Sample Quantiles')
```



We will try and add more predictors to our model. First, lets observe any multicollinearity.

Inspection of multicollinearity: VIF, correlations

The correlation between garage cars and total square feet is moderately high. We will take that into consideration when analyzing the output of the model.

In [65]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# the independent variables set
x = housing_training_data[['TotalSF','YrSinceRemod','GarageCars','ExterQual','CentralA

# 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)
x.corr()
```

```
       feature        VIF
0       TotalSF  19.799177
1  YrSinceRemod   1.901546
2    GarageCars  11.051476
3     ExterQual  22.737370
4  CentralAir_Y  12.377024
```

Out[65]:

	TotalSF	YrSinceRemod	GarageCars	ExterQual	CentralAir_Y
TotalSF	1.000000	-0.352279	0.556693	0.528016	0.180174
YrSinceRemod	-0.352279	1.000000	-0.422033	-0.587649	-0.299245
GarageCars	0.556693	-0.422033	1.000000	0.524166	0.233414
ExterQual	0.528016	-0.587649	0.524166	1.000000	0.206058
CentralAir_Y	0.180174	-0.299245	0.233414	0.206058	1.000000

These VIF values suggests that our model contains some multicollinearity across all features. Some resources stated that greater than 10 suggests multicollinearity, while others used a cut-off of 5.0 or 1.0. This may not be an issue for a predictive model, but could have implications for an inferential model.

In [66]:

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


polynomial_features= PolynomialFeatures(degree=2)
xp = polynomial_features.fit_transform(x)
xp.shape

#add constant to predictor variables
x = sm.add_constant(xp)

#fit polynomial regression model
model = sm.OLS(y_log, x).fit()

#view model summary
print(model.summary())
```

Out[66]:

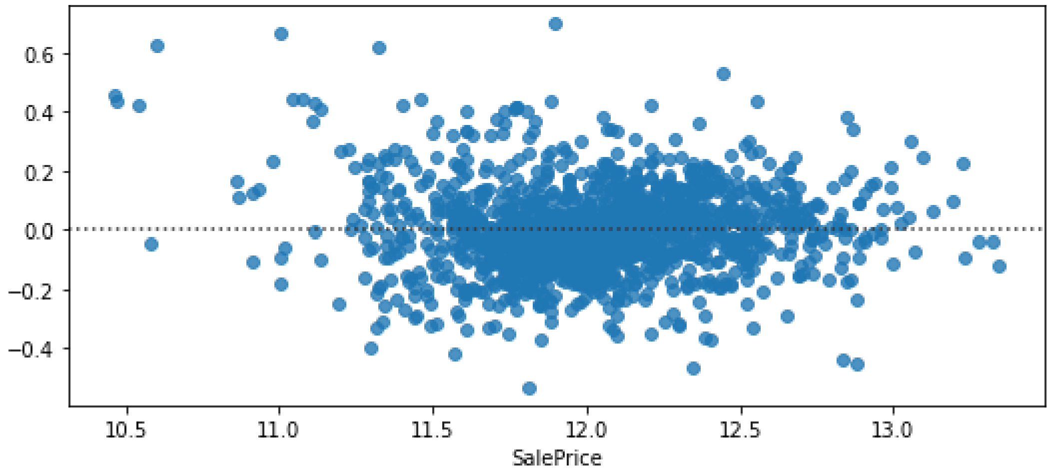
```
(1455, 21)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.838
Model:                            OLS   Adj. R-squared:                  0.835
Method:                 Least Squares   F-statistic:                     389.4
Date:                Sun, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:36   Log-Likelihood:                 608.02
No. Observations:                1455   AIC:                            -1176.
Df Residuals:                    1435   BIC:                            -1070.
Df Model:                          19
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         10.7869      0.166     65.166      0.000      10.462      11.112
x1             0.0004   5.06e-05      8.017      0.000       0.000       0.001
x2            -0.0022      0.003     -0.826      0.409      -0.007       0.003
x3             0.1521      0.051      2.955      0.003       0.051       0.253
x4            -0.0450      0.096     -0.468      0.640      -0.234       0.144
x5             0.0780      0.065      1.195      0.232      -0.050       0.206
x6         -2.858e-08   6.91e-09     -4.135      0.000   -4.21e-08    -1.5e-08
x7         -1.603e-06   4.55e-07     -3.523      0.000    -2.5e-06   -7.11e-07
x8           3.71e-06    1.1e-05      0.336      0.737    -1.8e-05    2.54e-05
x9           3.17e-05   1.51e-05      2.100      0.036    2.09e-06    6.13e-05
x10        -1.362e-05   2.64e-05     -0.516      0.606   -6.55e-05    3.82e-05
x11        -1.634e-05   1.74e-05     -0.939      0.348   -5.05e-05    1.78e-05
x12           -0.0011      0.000     -2.609      0.009      -0.002      -0.000
x13            0.0017      0.001      1.910      0.056   -4.62e-05       0.003
x14            0.0018      0.001      1.812      0.070      -0.000       0.004
x15           -0.0348      0.008     -4.562      0.000      -0.050      -0.020
x16            0.0361      0.019      1.870      0.062      -0.002       0.074
x17           -0.0044      0.023     -0.193      0.847      -0.049       0.040
x18           -0.0057      0.018     -0.320      0.749      -0.040       0.029
x19           -0.0054      0.058     -0.092      0.926      -0.119       0.108
x20            0.0780      0.065      1.195      0.232      -0.050       0.206
==============================================================================
Omnibus:                      240.889   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              706.892
Skew:                          -0.845   Prob(JB):                    3.17e-154
Kurtosis:                       5.967   Cond. No.                     1.01e+21
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The smallest eigenvalue is 9.79e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [67]:

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

Out[67]: <AxesSubplot:xlabel='SalePrice'>



The adjusted r-squared value is high for the 2nd order polynomial model, but the residuals are not randomly scattered. The plot indicates that the model is underestimating the value of homes with actual low sales prices and overestimating the value of homes with actual high sales prices.

In [68]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
```

Let's use PCA to find some more important predictors to add to our models

Regress on principal components

In [69]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.linear_model import LinearRegression
import os
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score

scaler = StandardScaler()

pca = PCA(n_components=8)


## independent variables ###

x_raw = housing_training_data.select_dtypes(exclude=['object']).drop(columns = ['SaleP
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
```

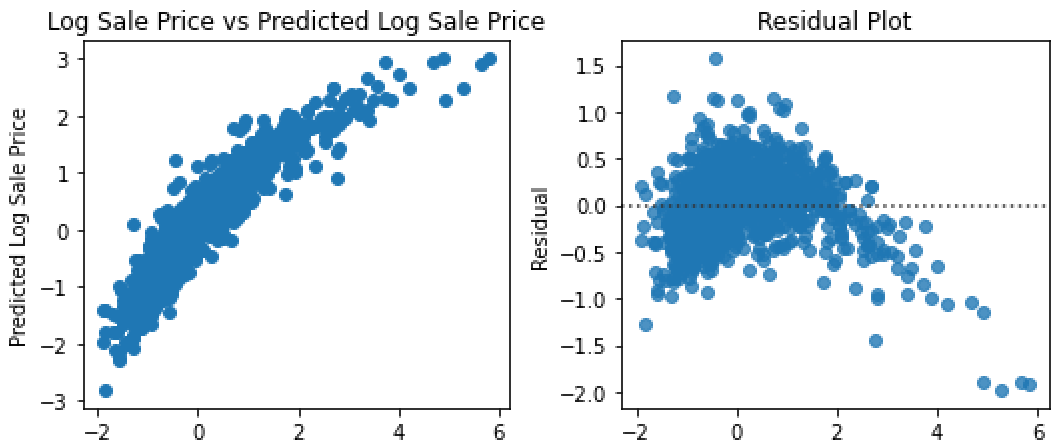
```
## importance scores of the principal components ##
# show loadings for features
loadings = pd.DataFrame(pca.components_.T, columns=['PC1','PC2','PC3','PC4','PC5','PC6
loadings


## 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)


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')

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

# Mean Squared Error
MSE = np.square(np.subtract(y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_scale, y_pred)
print("R_sq:",r2)
```

Out[69]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Id	0.003671	0.007609	-0.012863	-0.018848	-0.031304	0.014721	-0.000231	-0.230336
MSSubClass	0.013197	0.056123	-0.258946	-0.164299	0.272634	0.191425	-0.164072	-0.186888
LotFrontage	-0.098662	0.109474	0.215217	0.061892	-0.136464	-0.021591	0.195174	0.212405
LotArea	-0.060616	0.085147	0.241554	-0.042553	0.031626	-0.034394	0.141926	0.328845
OverallQual	-0.256954	0.013253	-0.057438	0.035494	0.024299	-0.095312	-0.106163	-0.076213
OverallCond	0.055815	0.014262	0.008466	-0.135176	-0.040599	-0.481640	0.270476	-0.152957
YearBuilt	-0.219252	-0.220127	-0.134601	0.005329	0.012171	0.181039	-0.024807	0.179010
YearRemodAdd	-0.202447	-0.135251	-0.224474	0.081358	0.105161	-0.206615	0.237536	0.013918
MasVnrArea	-0.143366	0.042596	0.050969	-0.027038	0.048708	0.116316	-0.144984	-0.030711
ExterQual	-0.233778	-0.064617	-0.114988	0.116459	0.063804	-0.076014	-0.067799	-0.095841
BsmtQual	-0.229908	-0.114895	-0.085267	0.067084	0.123963	0.021872	-0.090064	0.080774
BsmtFinSF1	-0.099578	-0.153913	0.299269	-0.151573	0.374517	0.060625	0.034273	-0.065014
BsmtFinSF2	0.008778	-0.017667	0.170686	-0.097427	0.022247	-0.067193	0.181676	0.085114
BsmtUnfSF	-0.096454	0.126643	-0.099844	0.415105	-0.359291	-0.007090	-0.097226	-0.009365
TotalBsmtSF	-0.205445	-0.032307	0.274765	0.249944	0.015222	0.029578	0.003010	-0.044868
1stFlrSF	-0.191291	0.047038	0.322391	0.228092	-0.012687	0.051398	0.047960	-0.094858
2ndFlrSF	-0.083826	0.325601	-0.263244	-0.286736	0.078471	-0.039820	-0.004822	0.045017
LowQualFinSF	0.016325	0.121673	0.010900	0.022469	0.018783	-0.096268	0.098606	-0.087547
GrLivArea	-0.212725	0.330630	0.009550	-0.079419	0.060909	-0.006249	0.040909	-0.039396
BsmtFullBath	-0.061426	-0.165104	0.243546	-0.121882	0.422020	0.067214	0.096279	-0.103952
BsmtHalfBath	0.012224	-0.002542	0.046869	-0.089462	-0.056439	-0.142792	0.079802	0.210566
FullBath	-0.201502	0.150867	-0.134477	0.064371	0.018548	0.139731	0.098124	0.012173
HalfBath	-0.093650	0.134351	-0.197361	-0.319660	0.076294	-0.018475	-0.112438	0.178204
BedroomAbvGr	-0.054909	0.356909	-0.017353	-0.089966	-0.083644	0.073435	0.242245	0.114330
KitchenAbvGr	0.041675	0.189837	-0.014183	0.086211	0.119344	0.377492	0.156014	-0.164956
KitchenQual	-0.224767	-0.072831	-0.089469	0.069620	0.071262	-0.165911	0.015015	-0.123078
TotRmsAbvGrd	-0.160080	0.368444	-0.015952	-0.048376	0.017501	0.055540	0.135361	-0.016877
Fireplaces	-0.140927	0.115976	0.199048	-0.128389	0.035761	-0.219753	-0.387406	0.022184
FireplaceQu	-0.162657	0.098898	0.121926	-0.017785	0.001883	-0.234443	-0.425147	-0.046188
GarageYrBlt	-0.217834	-0.202084	-0.162327	-0.010719	-0.048295	0.184504	0.047086	0.077313
GarageCars	-0.235138	-0.029470	0.027626	-0.079300	-0.147468	0.178528	0.004658	-0.089752
GarageArea	-0.224938	-0.039640	0.079160	-0.078222	-0.156884	0.177333	0.047977	-0.111435
GarageQual	-0.123336	-0.120531	0.066392	-0.325948	-0.343787	0.082770	0.031902	-0.204625
GarageCond	-0.121213	-0.139310	0.070769	-0.330448	-0.357112	0.073309	0.032527	-0.173778

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
WoodDeckSF	-0.104889	-0.019359	0.068648	-0.091644	0.117418	-0.031780	0.130396	0.072254
OpenPorchSF	-0.109004	0.064471	-0.040801	0.012955	0.060412	-0.076822	-0.057477	0.156020
EnclosedPorch	0.070821	0.133254	0.054251	0.045052	0.002906	-0.130765	0.058703	-0.383462
3SsnPorch	-0.016219	-0.018176	0.021199	0.031480	-0.041260	-0.060069	0.052732	0.065086
ScreenPorch	-0.024086	0.050690	0.111115	-0.107502	-0.031379	-0.128971	-0.230816	0.025060
PoolArea	-0.008178	0.038436	0.050052	-0.086850	-0.021028	-0.046144	0.093498	-0.148385
MiscVal	0.011021	0.023471	0.012395	-0.031549	0.006030	-0.034770	0.141798	-0.014292
MoSold	-0.019893	0.031797	0.001362	0.020580	-0.024677	-0.040583	-0.106347	0.175026
YrSold	0.007311	-0.029935	0.015310	-0.009106	0.065255	0.002643	0.143420	-0.129127
Street_Pave	-0.012603	0.006765	-0.080769	0.021471	-0.031430	-0.114273	0.005132	-0.366851
CentralAir_Y	-0.112814	-0.148689	0.011083	-0.165249	-0.133699	-0.179586	0.102746	0.064415
PavedDrive_Y	-0.108617	-0.164614	0.035119	-0.175303	-0.145556	0.098251	0.001568	0.098963
HeatingEx	-0.165890	-0.093989	-0.152127	0.112837	0.039147	-0.138834	0.035621	-0.020257
YrSinceRemod	0.202978	0.133363	0.225526	-0.081969	-0.100990	0.206846	-0.228373	-0.022236
TotalSF	-0.250927	0.199534	0.155107	0.083260	0.048228	0.011921	0.028484	-0.050165

```
Out[69]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
Out[69]: <AxesSubplot:>
Out[69]: <matplotlib.collections.PathCollection at 0x7fee82082650>
Out[69]: Text(0.5, 0, 'Log Sale Price (Standard Scaled)')
Out[69]: Text(0, 0.5, 'Predicted Log Sale Price')
Out[69]: Text(0.5, 1.0, 'Log Sale Price vs Predicted Log Sale Price')
Out[69]: <AxesSubplot:>
Out[69]: <AxesSubplot:>
Out[69]: Text(0.5, 0, 'Log Sale Price')
Out[69]: Text(0, 0.5, 'Residual')
Out[69]: Text(0.5, 1.0, 'Residual Plot')
MSE: 0.15473155070155967
R_sq: 0.8452684492984404
```



In [70]:
```
x_raw.columns
```

Out[70]:
```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'ExterQual',
       'BsmtQual', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'Street_Pave',
       'CentralAir_Y', 'PavedDrive_Y', 'HeatingEx', 'YrSinceRemod', 'TotalSF'],
      dtype='object')
```

In [71]:
```
#define cross-validation method to use

cv = KFold(n_splits=10, random_state=1, shuffle=True)

#build multiple linear regression model
model = LinearRegression()

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


#use k-fold CV to evaluate model R2
scores_r2 = cross_val_score(model, x_pca_scale, 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()
```

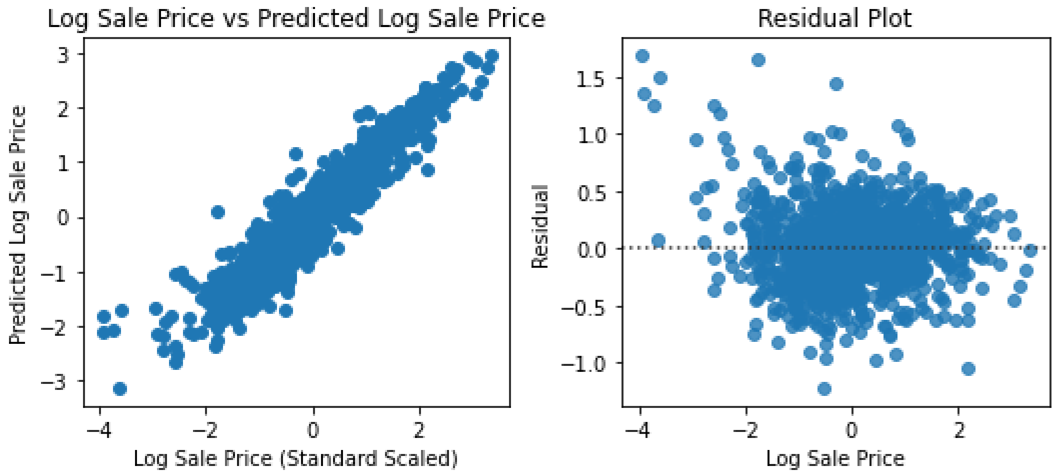
```
Over 10 folds: -0.28 MSE with a standard deviation of 0.04
Over 10 folds: 0.85 r2 with a standard deviation of 0.03
```

There appears to be a lot of large negative residuals for higher Sales Prices indicating that the model is overestimating the price of homes with large sales prices.

In [72]:
```
# Try log of Y

y_log_scale = scaler.fit_transform(np.array(np.log(housing_training_data['SalePrice'])

## train linear model ##
```

```
regr2 = LinearRegression()

regr2.fit(x_pca_scale, y_log_scale)

y_pred = regr2.predict(x_pca_scale)

plt.subplot(1, 2, 1)

plt.scatter(y_log_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')

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

##RMSE
# Mean Squared Error

MSE = np.square(np.subtract(y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_log_scale, y_pred)
print("R_sq:",r2)
```

```
Out[72]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
Out[72]: <AxesSubplot:>
Out[72]: <matplotlib.collections.PathCollection at 0x7fee820d0c10>
Out[72]: Text(0.5, 0, 'Log Sale Price (Standard Scaled)')
Out[72]: Text(0, 0.5, 'Predicted Log Sale Price')
Out[72]: Text(0.5, 1.0, 'Log Sale Price vs Predicted Log Sale Price')
Out[72]: <AxesSubplot:>
Out[72]: <AxesSubplot:>
Out[72]: Text(0.5, 0, 'Log Sale Price')
Out[72]: Text(0, 0.5, 'Residual')
Out[72]: Text(0.5, 1.0, 'Residual Plot')
MSE: 0.1675065487775354
R_sq: 0.8864767791398497
```



In [73]:
```
#define cross-validation method to use

cv = KFold(n_splits=10, random_state=1, shuffle=True)

#build multiple linear regression model
model = LinearRegression()

#use k-fold CV to evaluate model
scores_mse = cross_val_score(model, x_pca_scale, y_log_scale, scoring='neg_mean_absolu
                         cv=cv, n_jobs=-1)
print("Over 10 folds: %0.2f MSE with a standard deviation of %0.2f" % (scores_mse.mear


#use k-fold CV to evaluate model R2
scores_r2 = cross_val_score(model, x_pca_scale, y_log_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()
```

```
Over 10 folds: -0.25 MSE with a standard deviation of 0.01
Over 10 folds: 0.88 r2 with a standard deviation of 0.02
```

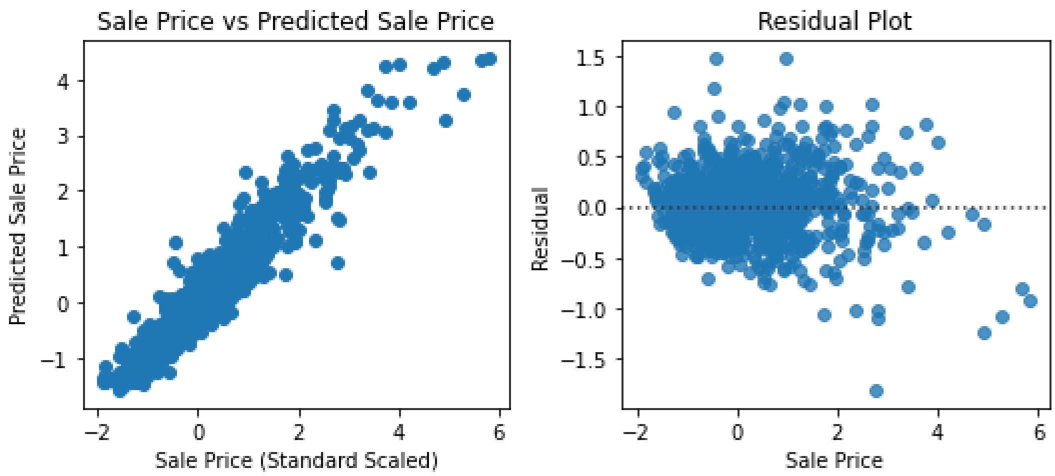
There are a lot of positive residuals for lower sales prices indicating that the model is underestimating the value of homes with lower sales prices.

In [74]:
```
### FROM PCA ON SALE PRICE WE SEE QUADRATIC PATTERN -- TRY DEGREE 2 POLYNOMIAL

from sklearn.preprocessing import PolynomialFeatures

#define our polynomial model, degree 2

degree=2

# PolynomialFeatures will create a new matrix consisting of all polynomial combinatior
# of the features with a degree less than or equal to the degree we just gave the mode
poly_model = PolynomialFeatures(degree=degree)

# transform out polynomial features
poly_x_values = poly_model.fit_transform(x_pca_scale)

regression_model = LinearRegression()
regression_model.fit(poly_x_values, y_scale)
y_pred = regression_model.predict(poly_x_values)
```

```
plt.subplot(1, 2, 1)
plt.scatter(y_scale, y_pred)
plt.xlabel('Sale Price (Standard Scaled)')
plt.ylabel('Predicted Sale Price')
plt.title('Sale Price vs Predicted Sale Price')

# Residuals

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


# SCORES

SS_Residual = sum((y_scale-y_pred)**2)
SS_Total = sum((y_scale-np.mean(y_scale))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y_scale)-1)/(len(y)-poly_x_values.shape[1]

print('x_shape_transformed',poly_x_values.shape)
print('x_shape_original',x_pca_scale.shape)

MSE = np.square(np.subtract(y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_scale, y_pred)
print("R_sq:",r_squared[0])

print("R_sq_adjusted:", 1 - (1-r_squared[0])*(len(y_scale)-1)/(len(y_scale)-poly_x_val
```

```
Out[74]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
Out[74]: <AxesSubplot:>
Out[74]: <matplotlib.collections.PathCollection at 0x7fee82271c10>
Out[74]: Text(0.5, 0, 'Sale Price (Standard Scaled)')
Out[74]: Text(0, 0.5, 'Predicted Sale Price')
Out[74]: Text(0.5, 1.0, 'Sale Price vs Predicted Sale Price')
Out[74]: <AxesSubplot:>
Out[74]: <AxesSubplot:>
Out[74]: Text(0.5, 0, 'Sale Price')
Out[74]: Text(0, 0.5, 'Residual')
Out[74]: Text(0.5, 1.0, 'Residual Plot')
```

```
x_shape_transformed (1455, 45)
x_shape_original (1455, 8)
MSE: 0.08764742224548526
R_sq: 0.9123525777545145
R_sq_adjusted: 0.9095533343187112
```



In [75]:
```
#define cross-validation method to use

cv = KFold(n_splits=10, random_state=1, shuffle=True)

#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
                         cv=cv, n_jobs=-1)
print("Over 10 folds: %0.2f MSE with a standard deviation of %0.2f" % (scores_mse.mear


#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()
```

```
Over 10 folds: -0.22 MSE with a standard deviation of 0.02
Over 10 folds: 0.90 r2 with a standard deviation of 0.01
```

We will need to verify that better fit is not just from including more variables. We do this using cross validation to assess out of sample accuracy.

The residuals are a bit more scattered, but there are still some more large negative residuals for higher sales prices.

In [76]:
```
# Lastly, try a regularization technique

from sklearn.linear_model import ElasticNet

scaler = StandardScaler()

# As an experiment, use on an original set of x variables...

x_raw = housing_training_data[numerical_vars].drop(columns = 'SalePrice')
x_scale = scale(x_raw)

#x_raw_new = housing_training_data.select_dtypes(exclude=['object']).drop(columns = [
#x_scale = scaler.fit_transform(x_raw_new)

regr = ElasticNet()
```

```
regr.fit(x_scale, y_log_scale)

ElasticNet(random_state=0)
print(regr.coef_)
print(regr.intercept_)

y_pred = regr.predict(x_scale)

plt.subplot(1, 2, 1)
plt.scatter(y_log_scale, y_pred)
plt.xlabel('Sale Price (Standard Scaled)')
plt.ylabel('Predicted Sale Price')
plt.title('ElasticNet: Sale Price vs Predicted Sale Price')

# Residuals

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


MSE = np.square(np.subtract(y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_scale, y_pred)
print("R_sq:",r2)
```

Out[76]:
```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[76]:
```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=0, selection='cyclic', tol=0.0001, warm_start=False)
[ 0.          0.          0.17296878 -0.          0.          0.
  0.          0.          0.00587394  0.          0.         -0.
  0.06700516  0.         -0.          0.          0.          0.
 -0.          0.          0.          0.02809537  0.00115474  0.
  0.         -0.          0.          0.          0.         -0.        ]
[-2.46470368e-15]
```

Out[76]: `<AxesSubplot:>`

Out[76]: `<matplotlib.collections.PathCollection at 0x7fee824f0f10>`

Out[76]: `Text(0.5, 0, 'Sale Price (Standard Scaled)')`

Out[76]: `Text(0, 0.5, 'Predicted Sale Price')`

Out[76]: `Text(0.5, 1.0, 'ElasticNet: Sale Price vs Predicted Sale Price')`

Out[76]: `<AxesSubplot:>`

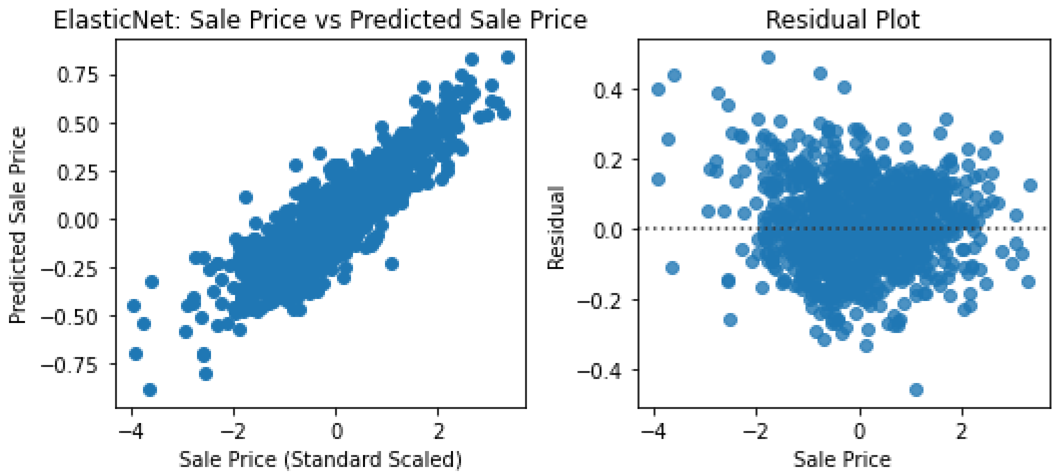
Out[76]: `<AxesSubplot:>`

Out[76]: `Text(0.5, 0, 'Sale Price')`

Out[76]: `Text(0, 0.5, 'Residual')`

Out[76]: `Text(0.5, 1.0, 'Residual Plot')`

```
MSE: 1.058250120091334
R_sq: 0.36078782270420806
```



In [77]:

```
x = housing_training_data[['OverallQual','TotalBsmtSF','GrLivArea','GarageCars','Garag
y_log = np.log(housing_training_data['SalePrice'])

#polynomial_features= PolynomialFeatures(degree=2)
#xp = polynomial_features.fit_transform(x)
#xp.shape

#add constant to predictor variables
#x = sm.add_constant(xp)
x = sm.add_constant(x)

#fit polynomial regression model
model = sm.OLS(y_log, x).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.825
Method:                 Least Squares   F-statistic:                     1370.
Date:                Sun, 16 Apr 2023   Prob (F-statistic):               0.00
Time:                        16:12:42   Log-Likelihood:                 555.36
No. Observations:                1455   AIC:                            -1099.
Df Residuals:                    1449   BIC:                            -1067.
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const          10.5342      0.020    524.743      0.000      10.495      10.574
OverallQual     0.1194      0.005     25.963      0.000       0.110       0.128
TotalBsmtSF     0.0002    1.3e-05     15.556      0.000       0.000       0.000
GrLivArea       0.0002   1.11e-05     21.632      0.000       0.000       0.000
GarageCars      0.0689      0.013      5.246      0.000       0.043       0.095
GarageArea      0.0001   4.51e-05      3.118      0.002    5.22e-05       0.000
==============================================================================
Omnibus:                      321.730   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1037.251
Skew:                          -1.085   Prob(JB):                    5.80e-226
Kurtosis:                       6.522   Cond. No.                     9.24e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 9.24e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

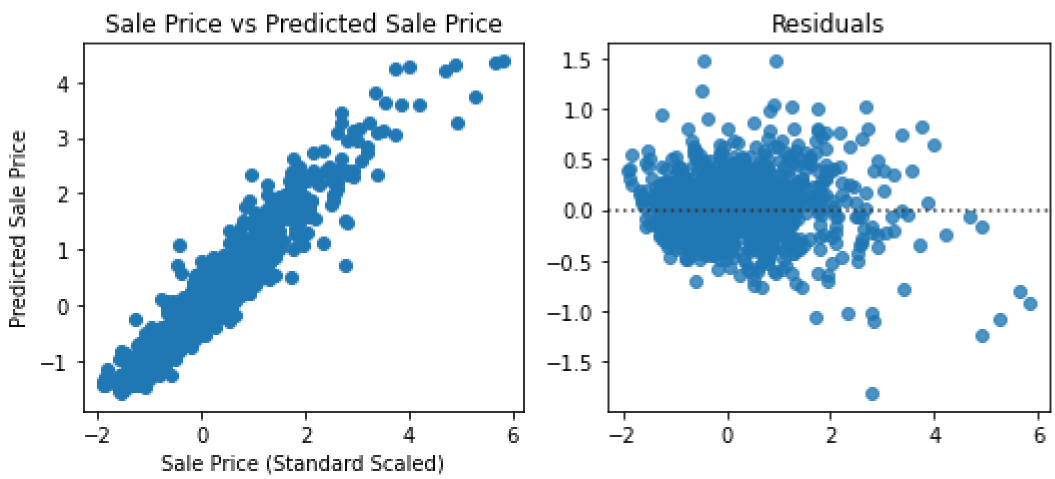
Polynomial regression using predictors from PCA for Kaggle

In [78]:

```
from sklearn.preprocessing import PolynomialFeatures
ypolyscaler = StandardScaler()
poly_y_scale = ypolyscaler.fit_transform(np.array(housing_training_data['SalePrice']).
#define our polynomial model, with whatever degree we want
degree=2

# PolynomialFeatures will create a new matrix consisting of all polynomial combinatior
# of the features with a degree less than or equal to the degree we just gave the mode
poly_model = PolynomialFeatures(degree=degree)

# transform out polynomial features
poly_x_values = poly_model.fit_transform(x_pca_scale)

poly_regression_model = LinearRegression()
poly_regression_model.fit(poly_x_values, poly_y_scale)
y_pred = poly_regression_model.predict(poly_x_values)

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')
```

```
# Residuals
plt.subplot(1,2,2)
plt.title('Residuals')
sns.residplot(x=poly_y_scale, y=y_pred)

# Mean Squared Error

MSE = np.square(np.subtract(poly_y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_scale, y_pred)
print("R_sq:",r2)
```

```
Out[78]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
Out[78]: <AxesSubplot:>
Out[78]: <matplotlib.collections.PathCollection at 0x7fee826cf9d0>
Out[78]: Text(0.5, 0, 'Sale Price (Standard Scaled)')
Out[78]: Text(0, 0.5, 'Predicted Sale Price')
Out[78]: Text(0.5, 1.0, 'Sale Price vs Predicted Sale Price')
Out[78]: <AxesSubplot:>
Out[78]: Text(0.5, 1.0, 'Residuals')
Out[78]: <AxesSubplot:title={'center':'Residuals'}>
MSE: 0.08764742224548526
R_sq: 0.9123525777545147
```



Ridge regression using predictors from PCA with Cross Validation to find the best alpha

Principal components have no collinearity by definition. However, we were interested in applying a Ridge regularization model to our PCA and seeing how it would perform, including what value of alpha it would select. There are many methods aimed at making our PCA more robust/reduce overfitting inherent in their calculation. See below for comments about the loading within our components.

In [79]:
```
from sklearn.model_selection import GridSearchCV
yscaler = StandardScaler()
y_log_scale = yscaler.fit_transform(np.array(np.log(housing_training_data['SalePrice']

pca = PCA(n_components=8)

## independent variables ###
xscaler = StandardScaler()
x_raw = housing_training_data.select_dtypes(exclude=['object']).drop(columns = ['SaleP
x_scale = xscaler.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


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_pca_scale, y_log_scale, test_siz

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

# Create the Ridge Regression model
ridge = Ridge()

# Set up the grid search with cross-validation
param_grid = {'alpha': alpha_values}
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error',

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

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

# Create and fit the Ridge Regression model with the best alpha value
ridge_best = Ridge(alpha=best_alpha)

ridgemodel = ridge_best.fit(X_train, y_train)
# View the coefficients of the ridge model
coefficients = ridgemodel.coef_
print("Coefficients:", coefficients)

# Predict
y_pred = ridgemodel.predict(X_test)

# plot predictors against log scaled
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 the mean squared error (MSE) of the predictions
MSE = np.square(np.subtract(y_test,y_pred)).mean()
print("MSE:",MSE)

# Calculate the R^2 of the predictions
r2 = r2_score(y_test, y_pred)
print("R_sq:",r2)
```

Out[79]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=-1,
             param_grid={'alpha': array([1.00000000e-04, 1.45634848e-04, 2.12095089e-
04, 3.08884360e-04,
       4.49843267e-04, 6.55128557e-04, 9.54095476e-04, 1.38949549e-03,
       2.02358965e-...
       1.67683294e+01, 2.44205309e+01, 3.55648031e+01, 5.17947468e+01,
       7.54312006e+01, 1.09854114e+02, 1.59985872e+02, 2.32995181e+02,
       3.39322177e+02, 4.94171336e+02, 7.19685673e+02, 1.04811313e+03,
       1.52641797e+03, 2.22299648e+03, 3.23745754e+03, 4.71486636e+03,
       6.86648845e+03, 1.00000000e+04])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
Best alpha value: 24.420530945486497
Coefficients: [[-0.28015625  0.04133806  0.05587674 -0.04646577  0.05143315 -0.078062
32
   0.01581126 -0.01322861]]
```

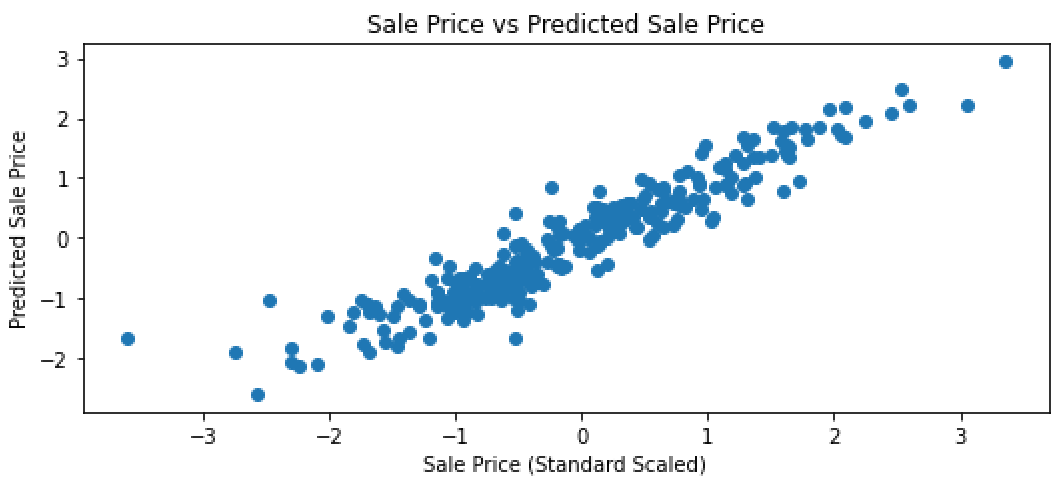
Out[79]: `<matplotlib.collections.PathCollection at 0x7fee87819e50>`

Out[79]: `Text(0.5, 0, 'Sale Price (Standard Scaled)')`

Out[79]: `Text(0, 0.5, 'Predicted Sale Price')`

Out[79]: `Text(0.5, 1.0, 'Sale Price vs Predicted Sale Price')`

```
MSE: 0.12096607983798188
R_sq: 0.8936548969444329
```



In [80]:
```
x_raw.columns
```

Out[80]:
```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'ExterQual',
       'BsmtQual', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'Street_Pave',
       'CentralAir_Y', 'PavedDrive_Y', 'HeatingEx', 'YrSinceRemod', 'TotalSF'],
      dtype='object')
```

Interpret the coefficients

Coef0: (0.29916159) has a positive impact on Sale Price. Given thatAs this feature increases, the predicted house price also increases. This coefficient has the largest positive effect among all features. In the PCA loadings for PC1, TotalSF and OverQual accounting for over 0.50 of the component, indicating both these features have a larger postive impact on Sale Price.

Coef1: (0.01065569) has a small positive impact on Sale Price. As this feature increases, the predicted house price marginally increases.

Coef2: (0.05379505) has a positive impact on Sale Price. As this feature increases, the predicted house price increases.

Coef3: (0.08326186) has a positive impact on Sale Price. As this feature increases, the predicted house price increases.

Coef4: (-0.09188438) has a negative impact on Sale Price. As this feature increases, the predicted house price decreases. This coefficient has the largest negative effect among all features.

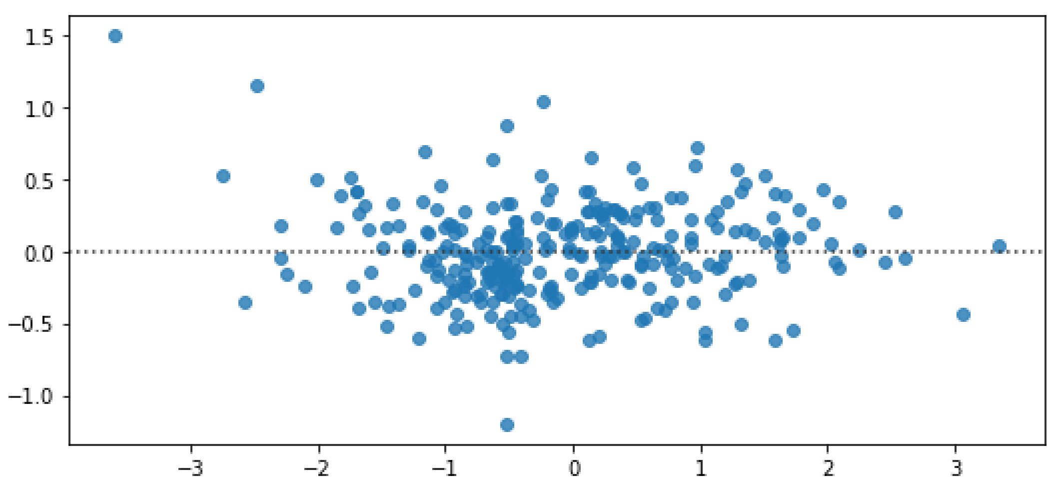
Coef5: (0.03202796) has a positive impact on Sale Price. As this feature increases, the predicted house price increases.

Coef6: (0.01423303) has a small positive impact on Sale Price. As this feature increases, the predicted house price marginally increases.

Coef7: (-0.01638527) has a small negative impact on Sale Price. As this feature increases, the predicted house price marginally decreases.

In [81]:
```
# Residuals

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

Out[81]: <AxesSubplot:>



Lasso Regression

We will now try to fit a Lasso Regression to the housing sales dataframe.

In [82]:
```
# Import libraries relevant to Lasso
from sklearn.linear_model import LassoCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import pandas as pd


# Create dataframe that can be used for lasso.  Only keep variables that aren't causir
lasso_sandbox = housing_training_data[ ['LotFrontage', 'LotArea', 'OverallQual', 'Over
                 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '
                 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
                 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF
                 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', '


# Perform a Log Transformation on the outcome variable SalePrice and drop the original
lasso_sandbox_log_saleprice = np.log(housing_training_data['SalePrice'])
lasso_sandbox_x = lasso_sandbox.drop(columns=['SalePrice'])

# Split the dataset into training and testing dataframes
lasso_X_train, lasso_X_validation, lasso_y_train, lasso_y_validation = train_test_spli
                                                                          lasso_sanc
                                                                          random_sta

# Standardize the numeric predictors - which can help strengthen the model fit
numerical_predictors = ['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'MasVr
                 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '
                 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
                 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF
                 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal']

from sklearn.preprocessing import StandardScaler
lasso_scaler = StandardScaler().fit(lasso_X_train[numerical_predictors])
lasso_X_train[numerical_predictors] = lasso_scaler.transform(lasso_X_train[numerical_p
lasso_X_validation[numerical_predictors] = lasso_scaler.transform(lasso_X_validation[r

# Let's visualize the Lasso coefficients as a function of the tuning parameter, alpha
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso

alphas = np.linspace(0.01,500,100)
lasso = Lasso(max_iter=10000)
coefs = []

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(lasso_X_train, lasso_y_train)
    coefs.append(lasso.coef_)

ax = plt.gca()

ax.plot(alphas, coefs)
```

```
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('alpha')
plt.ylabel('Standardized Coefficients')
plt.title('Lasso coefficients as a function of alpha');

# Fit a Lasso Regression Model with ten-fold cross-validation
lasso_model = LassoCV(cv=5, random_state=1, max_iter = 10000)
lasso_model.fit(lasso_X_train, lasso_y_train)


lasso_model.coef_
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/pandas/
core/frame.py:3678: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  self[col] = igetitem(value, i)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/pandas/
core/frame.py:3678: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  self[col] = igetitem(value, i)
```

```
Out[82]: Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=5.06040404040404, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=5.06040404040404, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=10.11080808080808, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=10.11080808080808, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=15.161212121212122, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=15.161212121212122, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=20.211616161616163, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=20.211616161616163, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=25.262020202020203, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=25.262020202020203, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=30.312424242424246, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=30.312424242424246, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=35.3628282828282828, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=35.3628282828282828, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=40.41323232323232, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=40.41323232323232, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=45.46363636363636, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=45.46363636363636, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=50.5140404040404, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=50.5140404040404, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=55.56444444444444, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=55.56444444444444, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=60.61484848484849, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=60.61484848484849, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=65.66525252525253, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=65.66525252525253, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=70.71565656565657, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=70.71565656565657, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=75.7660606060606, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=75.7660606060606, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=80.81646464646465, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=80.81646464646465, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=85.8668686868687, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=85.8668686868687, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=90.91727272727273, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=90.91727272727273, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=95.96767676767678, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=95.96767676767678, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=101.01808080808081, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=101.01808080808081, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=106.06848484848486, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=106.06848484848486, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=111.11888888888889, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=111.11888888888889, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=116.16929292929294, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=116.16929292929294, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=121.21969696969698, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=121.21969696969698, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=126.27010101010102, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=126.27010101010102, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=131.32050505050503, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=131.32050505050503, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=136.3709090909091, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=136.3709090909091, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=141.42131313131313, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=141.42131313131313, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=146.47171717171716, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=146.47171717171716, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=151.5221212121212, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=151.5221212121212, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=156.57252525252525, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=156.57252525252525, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=161.62292929292929, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=161.62292929292929, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=166.67333333333332, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=166.67333333333332, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=171.72373737373738, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=171.72373737373738, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=176.7741414141414, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=176.7741414141414, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=181.82454545454544, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=181.82454545454544, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=186.87494949494948, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=186.87494949494948, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=191.92535353535354, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=191.92535353535354, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=196.97575757575757, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=196.97575757575757, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=202.0261616161616, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=202.0261616161616, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=207.07656565656566, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=207.07656565656566, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=212.1269696969697, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=212.1269696969697, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=217.17737373737373, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=217.17737373737373, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=222.22777777777776, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=222.22777777777776, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=227.27818181818182, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=227.27818181818182, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=232.32858585858585, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=232.32858585858585, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=237.3789898989899, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=237.3789898989899, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=242.42939393939395, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=242.42939393939395, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=247.47979797979798, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=247.47979797979798, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=252.530202020202, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=252.530202020202, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=257.58060606060604, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=257.58060606060604, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=262.6310101010101, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=262.6310101010101, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=267.6814141414141, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=267.6814141414141, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=272.7318181818182, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[82]:
```
Lasso(alpha=272.7318181818182, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=277.78222222222223, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=277.78222222222223, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=282.83262626262626, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=282.83262626262626, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=287.8830303030303, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=287.8830303030303, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=292.9334343434343, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=292.9334343434343, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=297.98383838383836, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=297.98383838383836, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=303.0342424242424, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=303.0342424242424, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=308.0846464646465, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=308.0846464646465, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=313.1350505050505, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=313.1350505050505, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=318.18545454545455, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=318.18545454545455, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=323.2358585858586, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=323.2358585858586, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=328.2862626262626, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=328.2862626262626, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=333.33666666666664, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=333.33666666666664, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=338.3870707070707, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=338.3870707070707, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=343.43747474747477, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=343.43747474747477, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=348.4878787878788, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=348.4878787878788, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=353.53828282828283, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=353.53828282828283, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=358.58868686868686, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=358.58868686868686, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=363.6390909090909, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=363.6390909090909, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=368.68949494949493, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=368.68949494949493, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=373.73989898989896, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=373.73989898989896, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=378.79030303030305, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=378.79030303030305, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=383.8407070707071, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=383.8407070707071, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=388.8911111111111, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=388.8911111111111, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=393.94151515151515, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=393.94151515151515, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=398.9919191919192, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=398.9919191919192, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=404.0423232323232, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=404.0423232323232, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=409.09272727272725, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=409.09272727272725, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=414.14313131313133, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=414.14313131313133, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=419.19353535353537, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=419.19353535353537, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=424.2439393939394, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=424.2439393939394, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[82]: Lasso(alpha=429.29434343434343, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=429.29434343434343, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=434.34474747474746, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=434.34474747474746, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=439.3951515151515, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=439.3951515151515, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=444.44555555555553, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=444.44555555555553, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=449.4959595959596, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=449.4959595959596, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=454.54636363636365, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=454.54636363636365, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=459.5967676767677, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=459.5967676767677, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=464.6471717171717, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=464.6471717171717, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=469.69757575757575, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=469.69757575757575, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=474.7479797979798, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
Out[82]: Lasso(alpha=474.7479797979798, copy_X=True, fit_intercept=True, max_iter=10000,
               normalize=False, positive=False, precompute=False, random_state=None,
               selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=479.7983838383838, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=479.7983838383838, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=484.8487878787879, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=484.8487878787879, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=489.89919191919194, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=489.89919191919194, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=494.94959595959597, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

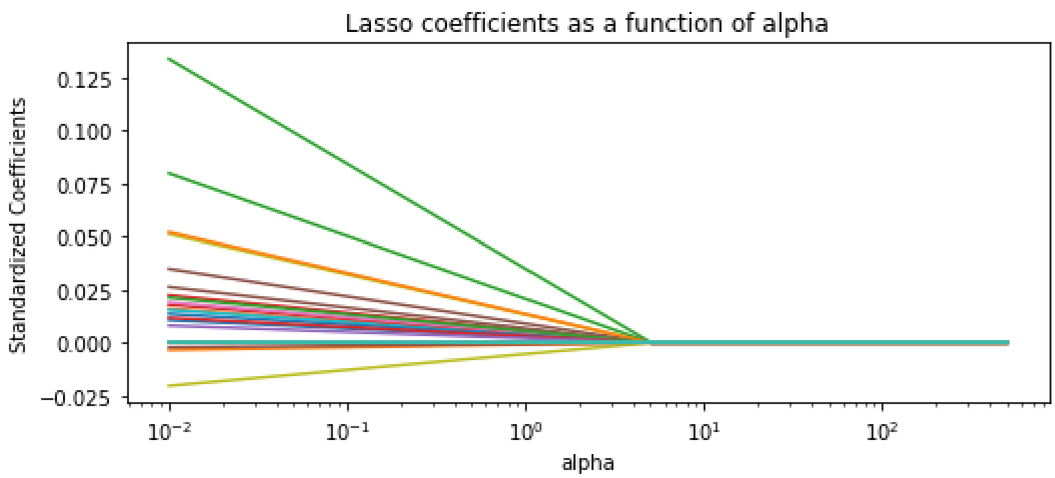
Out[82]:
```
Lasso(alpha=494.94959595959597, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=500.0, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
Lasso(alpha=500.0, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[82]:
```
[<matplotlib.lines.Line2D at 0x7fee87dd4210>,
 <matplotlib.lines.Line2D at 0x7fee87dd4450>,
 <matplotlib.lines.Line2D at 0x7fee87dd4590>,
 <matplotlib.lines.Line2D at 0x7fee87dd4690>,
 <matplotlib.lines.Line2D at 0x7fee87dd4790>,
 <matplotlib.lines.Line2D at 0x7fee87dd47d0>,
 <matplotlib.lines.Line2D at 0x7fee87dd4a10>,
 <matplotlib.lines.Line2D at 0x7fee87dd4b10>,
 <matplotlib.lines.Line2D at 0x7fee87dd4c10>,
 <matplotlib.lines.Line2D at 0x7fee87dd48d0>,
 <matplotlib.lines.Line2D at 0x7fee87d62b90>,
 <matplotlib.lines.Line2D at 0x7fee87d8d110>,
 <matplotlib.lines.Line2D at 0x7fee87dd4e50>,
 <matplotlib.lines.Line2D at 0x7fee87dd4f50>,
 <matplotlib.lines.Line2D at 0x7fee87dd4f90>,
 <matplotlib.lines.Line2D at 0x7fee87de1190>,
 <matplotlib.lines.Line2D at 0x7fee87de1290>,
 <matplotlib.lines.Line2D at 0x7fee87de1390>,
 <matplotlib.lines.Line2D at 0x7fee87de1490>,
 <matplotlib.lines.Line2D at 0x7fee87de1590>,
 <matplotlib.lines.Line2D at 0x7fee87de1690>,
 <matplotlib.lines.Line2D at 0x7fee87de1790>,
 <matplotlib.lines.Line2D at 0x7fee87de1890>,
 <matplotlib.lines.Line2D at 0x7fee87de1990>,
 <matplotlib.lines.Line2D at 0x7fee87de1a90>,
 <matplotlib.lines.Line2D at 0x7fee87de1b90>,
 <matplotlib.lines.Line2D at 0x7fee87de1c90>,
 <matplotlib.lines.Line2D at 0x7fee87de1d90>,
 <matplotlib.lines.Line2D at 0x7fee87de1e90>,
 <matplotlib.lines.Line2D at 0x7fee87de1f90>]
```

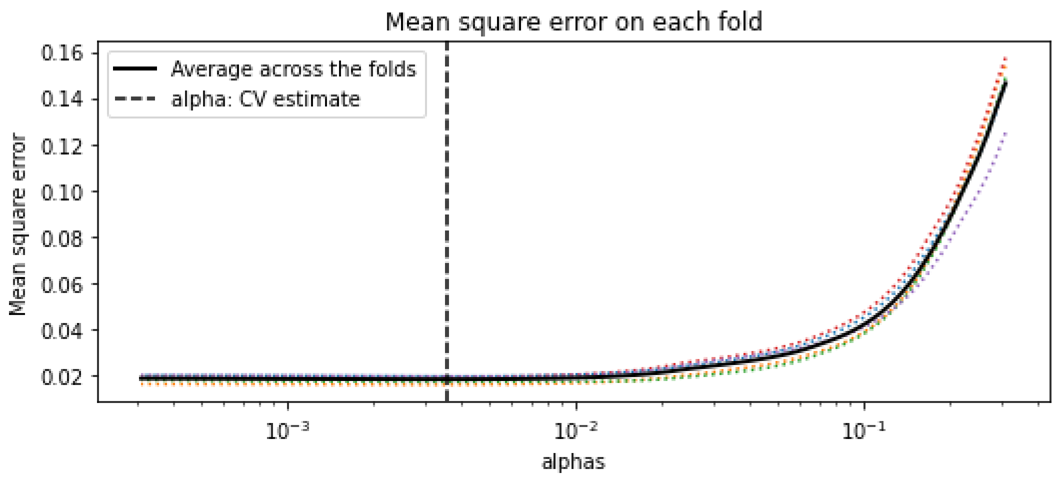
```
Out[82]: (0.005821722491960459,
          858.8523425678187,
          -0.028034197659991166,
          0.1415390558301679)
Out[82]: Text(0.5, 0, 'alpha')
Out[82]: Text(0, 0.5, 'Standardized Coefficients')
Out[82]: Text(0.5, 1.0, 'Lasso coefficients as a function of alpha')
Out[82]: LassoCV(alphas=None, copy_X=True, cv=5, eps=0.001, fit_intercept=True,
                 max_iter=10000, n_alphas=100, n_jobs=None, normalize=False,
                 positive=False, precompute='auto', random_state=1, selection='cyclic',
                 tol=0.0001, verbose=False)
Out[82]: array([ 0.0144498 ,  0.01915226,  0.13032816,  0.03233272,  0.        ,
                 0.0253451 ,  0.        , -0.        ,  0.05136664,  0.02051305,
                 0.        , -0.00935478,  0.07656874,  0.02401095,  0.        ,
                 0.04353805,  0.02686431, -0.        , -0.02599228,  0.        ,
                 0.01277892,  0.05172721,  0.0221585 ,  0.01390011,  0.01036037,
                -0.00643844, -0.        ,  0.00428796,  0.        ,  0.        ])
```



Let's plot the mean squared error as a function of our tuning parameter, alpha, from our cross-validation

```
In [83]: plt.semilogx(lasso_model.alphas_, lasso_model.mse_path_, ":")
plt.plot(
    lasso_model.alphas_ ,
    lasso_model.mse_path_.mean(axis=-1),
    "k",
    label="Average across the folds",
    linewidth=2,
)
plt.axvline(
    lasso_model.alpha_, linestyle="--", color="k", label="alpha: CV estimate"
)

plt.legend()
plt.xlabel("alphas")
plt.ylabel("Mean square error")
plt.title("Mean square error on each fold")
plt.axis("tight")
```

Out[83]:
```
[<matplotlib.lines.Line2D at 0x7fee880dca50>,
 <matplotlib.lines.Line2D at 0x7fee880fad50>,
 <matplotlib.lines.Line2D at 0x7fee880fae90>,
 <matplotlib.lines.Line2D at 0x7fee880faf90>,
 <matplotlib.lines.Line2D at 0x7fee880fafd0>]
```
Out[83]:
```
[<matplotlib.lines.Line2D at 0x7fee880e2990>]
```
Out[83]:
```
<matplotlib.lines.Line2D at 0x7fee8806a210>
```
Out[83]:
```
<matplotlib.legend.Legend at 0x7fee880e2d10>
```
Out[83]:
```
Text(0.5, 0, 'alphas')
```
Out[83]:
```
Text(0, 0.5, 'Mean square error')
```
Out[83]:
```
Text(0.5, 1.0, 'Mean square error on each fold')
```
Out[83]:
```
(0.00022029931480145153,
 0.4395549208368023,
 0.008856435504039587,
 0.16445482294561087)
```



Let's fit a Lasso Regression using the newly identified best value for the tuning parameter, alpha.

In [84]:
```
alphas = np.linspace(0.01,500,100)

best_alpha
alphas[0:10]
```
Out[84]:
```
24.420530945486497
```
Out[84]:
```
array([1.00000000e-02, 5.06040404e+00, 1.01108081e+01, 1.51612121e+01,
       2.02116162e+01, 2.52620202e+01, 3.03124242e+01, 3.53628283e+01,
       4.04132323e+01, 4.54636364e+01])
```

In [85]:
```
### plot training vs testing MSE for increasing lambda

from sklearn.metrics import mean_squared_error

lasso = Lasso(max_iter=10000)
alphas = np.linspace(0.00005,0.015,100)
#alphas = np.linspace(0.00001,0.00003,1000)

scaler = StandardScaler()
lasso_x = scaler.fit_transform(lasso_sandbox_x)
lasso_y = scaler.fit_transform(np.array(lasso_sandbox['SalePrice']).reshape(-1,1))
```

```
# Split the dataset into training and testing dataframes
lasso_X_train, lasso_X_validation, lasso_y_train, lasso_y_validation = train_test_spli
                                                                          lasso_y, t
                                                                          random_sta

training_mse = []
testing_mse = []

#best_alpha = lasso_model.alpha_

for a in alphas:
    lasso.set_params(alpha=a)
    lasso.fit(lasso_X_train, lasso_y_train)
    training_mse.append(mean_squared_error(lasso_y_train, lasso.predict(lasso_X_train)
    testing_mse.append(mean_squared_error(lasso_y_validation, lasso.predict(lasso_X_va
    #coefs.append(lasso.coef_)
```

Out[85]:
```
Lasso(alpha=5e-05, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.33398039476
475105, tolerance: 0.10374553746774125
  positive)
```

Out[85]:
```
Lasso(alpha=5e-05, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000201010101010101, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000201010101010101, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.00035202020202020203, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.00035202020202020203, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000503030303030303, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000503030303030303, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000654040404040404, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.000654040404040404, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.0008050505050505051, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.0008050505050505051, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.000956060606060606, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.000956060606060606, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001107070707070707, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001107070707070707, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001258080808080808, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001258080808080808, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0014090909090909089, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0014090909090909089, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00156010101010101, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00156010101010101, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001711111111111111, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.001711111111111111, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0018621212121212119, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0018621212121212119, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.002013131313131313, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.002013131313131313, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0021641414141414144, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0021641414141414144, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0023151515151515153, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0023151515151515153, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.002466161616161616162, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.002466161616161616162, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.002617171717171717, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.002617171717171717, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.002768181818181818, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.002768181818181818, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0029191919191919194, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0029191919191919194, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0030702020202020204, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0030702020202020204, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0032212121212121213, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0032212121212121213, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003372222222222222, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003372222222222222, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003523232323232323, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003523232323232323, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003674242424242424, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.003674242424242424, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0038252525252525254, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.0038252525252525254, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.003976262626262626255, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.003976262626262626255, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004127272727272727, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004127272727272727, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004278282828282828, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004278282828282828, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004429292929292929, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004429292929292929, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00458030303030303, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00458030303030303, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004731313131313131305, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004731313131313131305, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004882323232323232, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.004882323232323232, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005033333333333333, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005033333333333333, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005184343434343434, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005184343434343434, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005335353535353535, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005335353535353535, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.005486363636363636, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005486363636363636, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005637373737373737, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005637373737373737, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005788383838383838, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005788383838383838, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005939393939393939, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.005939393939393939, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00609040404040404, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00609040404040404, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006241414141414141, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006241414141414141, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006392424242424242, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006392424242424242, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006543434343434343, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006543434343434343, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006694444444444444, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006694444444444444, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006845454545454545, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006845454545454545, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.006996464646464646, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.006996464646464646, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007147474747474747, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007147474747474747, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0072984848484848475, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.0072984848484848475, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007449494949494949, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007449494949494949, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00760050505050505, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00760050505050505, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007751515151515151, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007751515151515151, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007902525252525251, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.007902525252525251, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008053535353535353, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008053535353535353, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008204545454545454, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008204545454545454, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008355555555555555, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008355555555555555, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.008506565656565657, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008506565656565657, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008657575757575756, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008657575757575756, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008808585858585858, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008808585858585858, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008959595959595959, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.008959595959595959, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00911060606060606, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.00911060606060606, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009261616161616162, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009261616161616162, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009412626262626261, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009412626262626261, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009563636363636363, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009563636363636363, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009714646464646464, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009714646464646464, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009865656565656565, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.009865656565656565, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.010016666666666667, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010016666666666667, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010167676767676766, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010167676767676766, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010318686868686868, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010318686868686868, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010469696969696969, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010469696969696969, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01062070707070707, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01062070707070707, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010771717171717172, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010771717171717172, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010922727272727271, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.010922727272727271, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011073737373737373, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011073737373737373, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011224747474747474, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011224747474747474, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011375757575757576, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011375757575757576, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

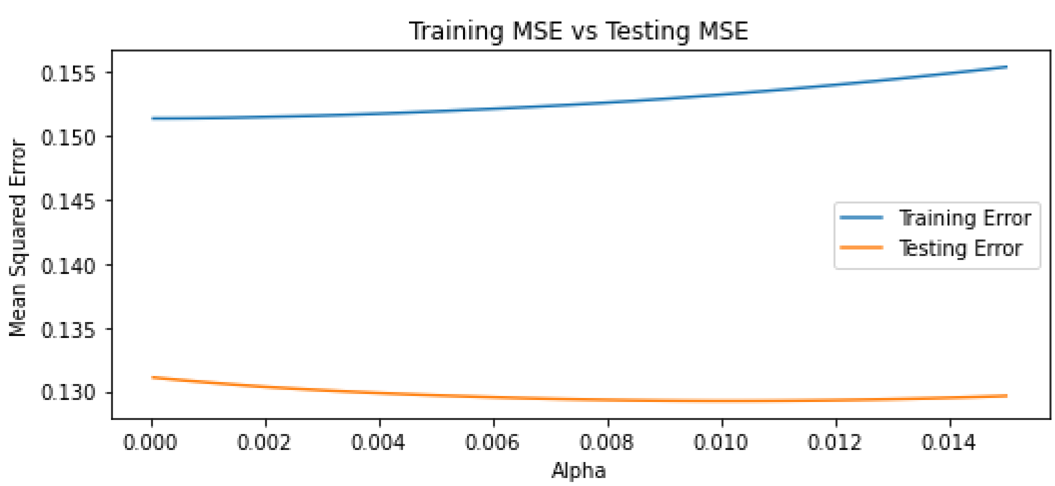
```
Out[85]: Lasso(alpha=0.011526767676767677, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011526767676767677, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011677777777777777, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011677777777777777, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011828787878787878, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.011828787878787878, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01197979797979798, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01197979797979798, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01213080808080808, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01213080808080808, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012281818181818182, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012281818181818182, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012432828282828282, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012432828282828282, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012583838383838383, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012583838383838383, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012734848484848484, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012734848484848484, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012885858585858586, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.012885858585858586, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
Out[85]: Lasso(alpha=0.013036868686868687, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013036868686868687, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013187878787878787, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013187878787878787, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013338888888888888, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013338888888888888, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01348989898989899, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01348989898989899, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01364090909090909, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.01364090909090909, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013791919191919192, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013791919191919192, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013942929292929292, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.013942929292929292, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014093939393939393, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014093939393939393, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014244949494949494, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014244949494949494, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014395959595959596, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
Out[85]: Lasso(alpha=0.014395959595959596, copy_X=True, fit_intercept=True,
               max_iter=10000, normalize=False, positive=False, precompute=False,
               random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[85]:
```
Lasso(alpha=0.014546969696969695, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.014546969696969695, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.014697979797979797, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.014697979797979797, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.014848989898989898, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.014848989898989898, copy_X=True, fit_intercept=True,
      max_iter=10000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.015, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```
Out[85]:
```
Lasso(alpha=0.015, copy_X=True, fit_intercept=True, max_iter=10000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [86]:
```python
diff = np.array(testing_mse) - np.array(training_mse)
#diff
```

In [87]:
```python
plt.plot(alphas, training_mse, label='Training Error')
plt.plot(alphas, testing_mse, label='Testing Error')
#plt.axvline(best_alpha, linestyle="--", color="k", label="Optimal Alpha")
plt.legend()

ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
plt.title('Training MSE vs Testing MSE');

# Fit a Lasso Regression Model with ten-fold cross-validation
#lasso_model = LassoCV(cv=5, random_state=1, max_iter = 10000)
#lasso_model.fit(lasso_X_train, lasso_y_train)
```



In [88]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, get_scorer
from sklearn.linear_model import Lasso, Ridge, LassoCV,LinearRegression, ElasticNet
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import KFold, RepeatedKFold, GridSearchCV, cross_validate

def regmodel_param_test(
    alphas_to_try, X, y, cv, scoring = 'r2',
    model_name = 'LASSO', X_test = None, y_test = None,
    draw_plot = False, filename = None):

    validation_scores = []
    train_scores = []
    results_list = []
    if X_test is not None:
        test_scores = []
        scorer = get_scorer(scoring)
    else:
        test_scores = None

    for curr_alpha in alphas_to_try:

        if model_name == 'LASSO':
            regmodel = Lasso(alpha = curr_alpha)
        elif model_name == 'Ridge':
            regmodel = Ridge(alpha = curr_alpha)
        elif model_name == 'ElasticNet':
            regmodel = ElasticNet(alpha = curr_alpha)
        else:
            return None

        results = cross_validate(
            regmodel, X, y, scoring=scoring, cv=cv,
            return_train_score = True)

        validation_scores.append(np.mean(results['test_score']))
        train_scores.append(np.mean(results['train_score']))
        results_list.append(results)

        if X_test is not None:
            regmodel.fit(X,y)
            y_pred = regmodel.predict(X_test)
            test_scores.append(scorer(regmodel, X_test, y_test))

    chosen_alpha_id = np.argmax(validation_scores)
    chosen_alpha = alphas_to_try[chosen_alpha_id]
    max_validation_score = np.max(validation_scores)
    if X_test is not None:
        test_score_at_chosen_alpha = test_scores[chosen_alpha_id]
    else:
        test_score_at_chosen_alpha = None

    if draw_plot:
        regmodel_param_plot(
            validation_scores, train_scores, alphas_to_try, chosen_alpha,
            scoring, model_name, test_scores, filename)
```

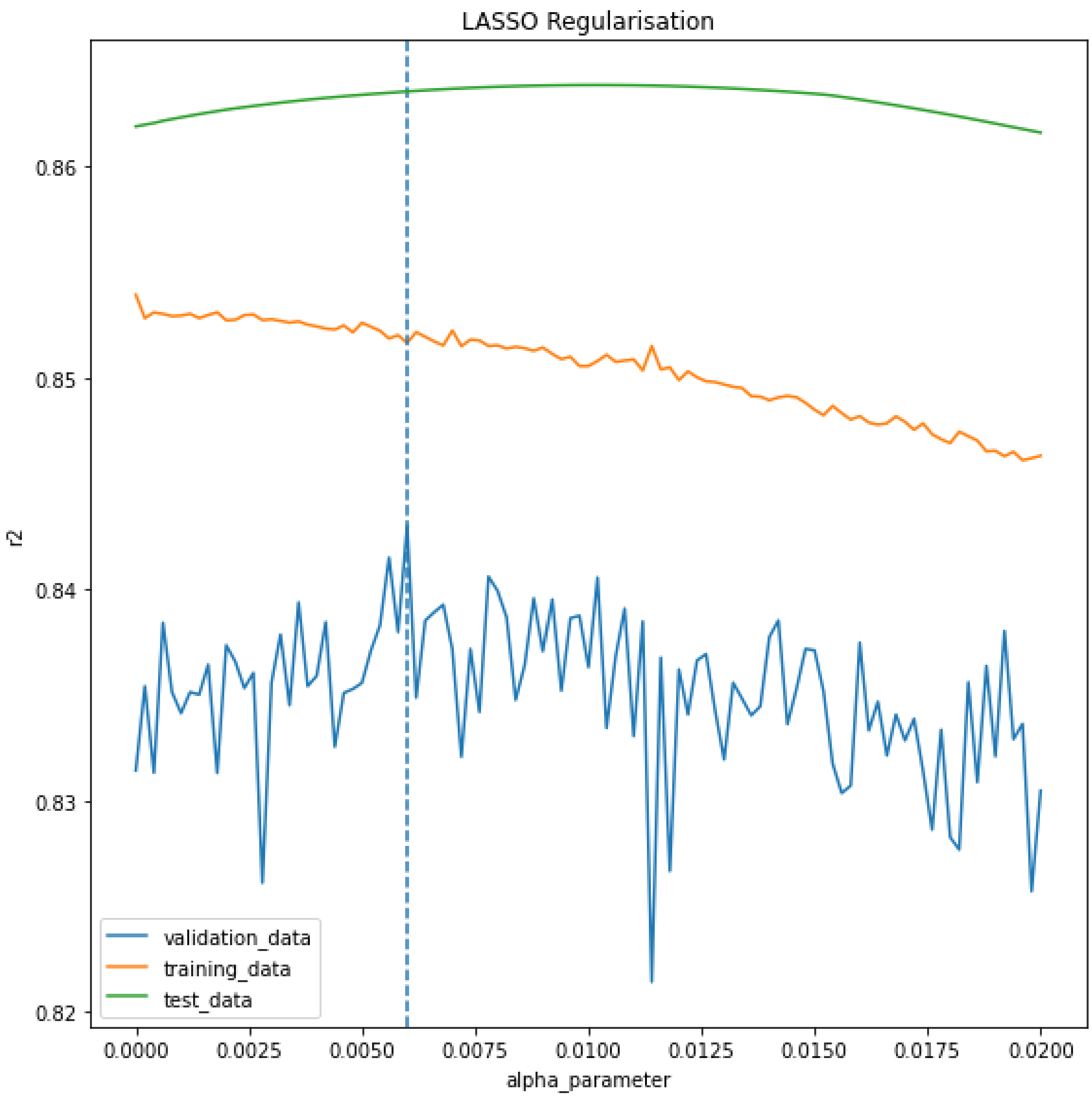
```
    return chosen_alpha, max_validation_score, test_score_at_chosen_alpha

def regmodel_param_plot(
    validation_score, train_score, alphas_to_try, chosen_alpha,
    scoring, model_name, test_score = None, filename = None):

    plt.figure(figsize = (8,8))
    sns.lineplot(y = validation_score, x = alphas_to_try,
                 label = 'validation_data')
    sns.lineplot(y = train_score, x = alphas_to_try,
                 label = 'training_data')
    plt.axvline(x=chosen_alpha, linestyle='--')
    if test_score is not None:
        sns.lineplot(y = test_score, x = alphas_to_try,
                     label = 'test_data')
    plt.xlabel('alpha_parameter')
    plt.ylabel(scoring)
    plt.title(model_name + ' Regularisation')
    plt.legend()
    if filename is not None:
        plt.savefig(str(filename) + ".png")
    plt.show()
```

In [89]:
```
cv = KFold(n_splits=5, shuffle=True)

lasso_alphas = np.linspace(0, 0.02, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, lasso_X_train, lasso_y_train,
        cv, scoring = 'r2', model_name = 'LASSO',
        X_test = lasso_X_validation, y_test = lasso_y_validation,
        draw_plot = True, filename = 'lasso_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 59.3151184062
4592, tolerance: 0.08146137472628749
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 62.4133800932
1754, tolerance: 0.08383375419276164
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 52.0013250398
59324, tolerance: 0.07646279640865453
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 65.0095184659
7536, tolerance: 0.08599641085489657
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 64.7876398341
8031, tolerance: 0.08721964620031875
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:44: UserWarning: With alpha=0, this algorithm does not converge well.
You are advised to use the LinearRegression estimator
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 77.0555878122
8252, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.34496764577
092165, tolerance: 0.08126724283571046
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.12904315448
244574, tolerance: 0.07752593667652338
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.40067582136
090607, tolerance: 0.08818182766329767
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.21124932477
2524, tolerance: 0.08300929344027744
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.42393140899
55144, tolerance: 0.08483106610569778
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.36810356882
534734, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.23611889218
67381, tolerance: 0.08262500137838019
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.40371434196
553935, tolerance: 0.07940128899327321
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.20887390371
```

```
1023, tolerance: 0.08018441167777415
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.49385535614
66392, tolerance: 0.083430130263864
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.24958147857
60473, tolerance: 0.0893172914193282
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.39077165557
705484, tolerance: 0.10374553746774125
  positive)
```




```
Chosen alpha: 0.00600
Validation score: 0.84299
Test score at chosen alpha: 0.86357
```

In [90]:

```
### BASED ON LASSOCV:

# Show best value of tuning parameter (alpha) chosen by cross validation
print(f"The best value of the tuning parameter, alpha, chosen by cross-validation is {
```

```
# Set best tuning parameter (alpha) and use it to fit our final Lasso regression model
lasso_best = Lasso(alpha=lasso_model.alpha_)
lasso_best.fit(lasso_X_train, lasso_y_train)

# Show best Lasso model coefficients and names
best_lasso_coeffs_data = (list(zip(lasso_X_train, lasso_best.coef_)))

best_lasso_coeffs_df = pd.DataFrame(best_lasso_coeffs_data, columns = ['Predictor', 'E


with pd.option_context("display.max_rows",300):
    best_lasso_coeffs_df.style.background_gradient(cmap = 'Greens')
```

```
The best value of the tuning parameter, alpha, chosen by cross-validation is 0.003577
8259276540722.
```

Out[90]:

```
Lasso(alpha=0.0035778259276540722, copy_X=True, fit_intercept=True,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[90]:

	Predictor	Best Lasso Coefficient
0	[-0.60134894 -0.12192403 -0.7943627 -0.51768686 -0.57401865 0.58069606 -0.28918978 -0.59461551 -0.14495197 -0.45356901 -0.79680412 -0.12045114 -1.04155359 -0.81857737 3.96564919 -1.02643215 -0.75919202 0.16691984 -0.21183287 -0.93449284 -0.95104897 0.31592367 0.94577272 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.041777
1	[-0.93621711 -0.45101191 -0.06376995 0.38008952 -0.57401865 -0.09178658 -0.28918978 0.31764015 0.13181675 -0.14422762 -0.79680412 -0.12045114 -0.81315996 -0.81857737 -0.23990443 -1.02643215 -0.75919202 -1.05968807 -0.21183287 -0.93449284 0.61077524 0.31592367 -0.1486834 0.577744 1.12931613 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.067138
2	[-0.02728921 -0.13358662 -0.06376995 2.17564227 0.84977655 -0.08480579 0.10705153 -0.52199815 -0.61031529 -0.97370056 1.13164459 -0.12045114 0.25604563 -0.81857737 3.96564919 -1.02643215 1.23112219 1.39352774 -0.21183287 0.30836555 -0.95104897 0.31592367 0.05888586 -0.74905884 -0.0948976 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.321800
3	[0.4510939 0.01924402 1.39741555 -0.51768686 1.32437495 1.98150764 -0.28918978 -0.33818544 1.60628371 1.53935609 -0.79680412 -0.12045114 0.42986733 1.114913 -0.23990443 0.80378694 -0.75919202 0.16691984 -0.21183287 0.30836555 0.61077524 1.65606634 1.67226514 0.91344111 -0.03368691 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.064117
4	[0.11622572 -0.1832794 0.6668228 -0.51768686 -0.57401865 -1.0155776 -0.28918978 1.82898909 0.79067325 0.59216916 -0.79680412 -0.12045114 -0.26946184 -0.81857737 -0.23990443 0.80378694 -0.75919202 0.16691984 -0.21183287 -0.31306364 -0.95104897 0.31592367 0.27117261 -0.74905884 1.43536957 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.073773
5	[0.92947701 0.20422285 1.39741555 -0.51768686 -0.57401865 2.37243181 -0.28918978 -0.90323932 1.40789198 1.28202785 -0.79680412 -0.12045114 0.23987617 1.114913 -0.23990443 0.80378694 -0.75919202 -1.05968807 -0.21183287 -0.93449284 0.61077524 1.65606634 1.3467588 -0.74905884 1.0375001 2.79957949 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.092032
6	[-0.93621711 -0.4729173 -0.7943627 2.17564227 -0.57401865 -1.0155776 -0.28918978 -0.85331488 -2.1043765 -0.7382726 -0.79680412 -0.12045114 -1.25175658 -0.81857737 -0.23990443 -1.02643215 -0.75919202 -1.05968807 -0.21183287 -1.55592204 -0.95104897 -1.024219 -0.93650311 0.40190266 -0.40095103 0.95699807 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.002737
7	[-0.21864245 -0.26806136 -1.52495546 -2.31323961 -0.57401865 0.43177257 -0.28918978 -0.14983414 0.17835308 0.47171764 0.71674998 -0.12045114 0.96143835 1.114913 -0.23990443 0.80378694 -0.75919202 1.39352774 -0.21183287 0.30836555 2.17259946 0.31592367 0.49289432 2.69583288 -0.03368691 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.000000
8	[-0.12296583 0.19337157 -0.06376995 2.17564227 -0.57401865 -1.0155776 -0.28918978 0.26771571 -0.89443185 -1.29125456 0.38993355 -0.12045114 -0.62518997 -0.81857737 -0.23990443 -1.02643215 -0.75919202 0.16691984 -0.21183287 0.30836555 -0.95104897 0.31592367 0.26645513 -0.74905884 -0.00308157 -0.35913151 -0.11654172 -0.27070619 -0.05813532 1.5224731]	0.139404
9	[-0.21864245 -0.20254802 -0.7943627 0.38008952 -0.57401865 0.78779279 -0.28918978 -0.64680925 0.01670056 -0.27289174 -0.79680412 -0.12045114 -0.90815554 1.114913 -0.23990443 -1.02643215 -0.75919202 0.16691984 -0.21183287 -0.31306364 0.61077524 -1.024219 -0.79026113 -0.74905884 0.59372263 2.66796653 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.041091

	Predictor	Best Lasso Coefficient
10	[-0.31431907 -0.26086098 0.6668228 -0.51768686 -0.35931937 -1.0155776 -0.28918978 1.52263458 0.46002036 0.26640256 -0.79680412 -0.12045114 -0.50998257 -0.81857737 -0.23990443 0.80378694 -0.75919202 0.16691984 -0.21183287 -0.31306364 0.61077524 0.31592367 -0.32794777 1.00935457 -0.38564836 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.000000
11	[-0.79270218 -0.68588634 0.6668228 -0.51768686 0.16047887 -1.0155776 -0.28918978 1.83125838 0.79312253 1.09313797 -0.79680412 -0.12045114 0.10041457 -0.81857737 -0.23990443 0.80378694 -0.75919202 -1.05968807 -0.21183287 0.30836555 0.61077524 1.65606634 0.88916292 0.39390987 -0.40095103 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.025678
12	[0.25974066 -0.06310401 -0.7943627 -0.51768686 -0.57401865 -1.0155776 -0.28918978 -1.2844805 -2.56973983 1.39152923 -0.79680412 -0.12045114 0.32072348 -0.81857737 -0.23990443 0.80378694 -0.75919202 1.39352774 4.32076744 0.92979475 -0.95104897 -2.36436167 -2.22437604 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.253948
13	[-0.98405542 -0.4692664 -2.25554821 2.17564227 -0.57401865 -0.41988364 -0.28918978 0.77149869 0.27632431 0.02002445 -0.79680412 -0.12045114 -0.691889 1.114913 -0.23990443 -1.02643215 -0.75919202 0.16691984 -0.21183287 -0.31306364 -0.95104897 -1.024219 -1.18652972 0.38591708 0.79265736 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.022657
14	[-0.93621711 -0.45101191 -0.7943627 2.17564227 -0.57401865 -1.0155776 -0.28918978 0.72157425 -0.40457572 -0.74374767 0.2786769 -0.12045114 -0.31797022 1.114913 -0.23990443 -1.02643215 -0.75919202 0.16691984 -0.21183287 -0.93449284 0.61077524 -1.024219 -1.20539966 -0.74905884 -0.70700446 3.0628054 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.016310
15	[0.30757897 -0.28570737 -0.7943627 3.07341864 -0.57401865 -1.0155776 -0.28918978 -0.46753512 -1.68799879 -0.33859256 1.01111655 -0.12045114 0.6198585 -0.81857737 -0.23990443 0.80378694 -0.75919202 1.39352774 4.32076744 0.92979475 0.61077524 0.31592367 0.94577272 2.0004603 -0.70700446 0.29893328 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.061960
16	[-0.45783401 0.03577448 0.6668228 2.17564227 -0.57401865 -1.0155776 -0.28918978 0.34487166 -0.81115631 1.15062619 1.47005026 -0.12045114 2.11957598 -0.81857737 -0.23990443 -1.02643215 1.23112219 0.16691984 -0.21183287 0.92979475 -0.95104897 0.31592367 1.09673218 -0.74905884 0.02752377 1.99345012 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.034481
17	[0.49893221 0.25726228 -0.7943627 0.38008952 2.01932261 1.969873 0.62092699 -1.2844805 0.93273153 1.34499114 -0.79680412 -0.12045114 0.28636337 1.114913 -0.23990443 0.80378694 -0.75919202 0.16691984 -0.21183287 0.30836555 0.61077524 0.31592367 0.70518107 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.120404
18	[0.4510939 -0.09190553 0.6668228 -0.51768686 0.94017625 -1.0155776 -0.28918978 2.12399714 1.10907974 0.94804864 -0.79680412 -0.12045114 -0.0067081 -0.81857737 -0.23990443 0.80378694 -0.75919202 0.16691984 -0.21183287 0.30836555 -0.95104897 0.31592367 0.81368319 -0.74905884 1.0375001 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.076264
19	[0.92947701 0.02360482 -0.7943627 -2.31323961 -0.32541949 0.74358113 -0.28918978 1.84714343 2.66192368 2.68364551 -0.79680412 -0.12045114 1.27472166 -0.81857737 -0.23990443 0.80378694 -0.75919202 1.39352774 -0.21183287 0.30836555 -0.95104897 0.31592367 0.26645513 -0.74905884 -0.24792432 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.106356

	Predictor	Best Lasso Coefficient
20	[0.49893221 0.63330477 0.6668228 -0.51768686 0.46557785 0.82269673 2.61451608 -0.98266457 0.83965886 0.64691985 2.42963892 -0.12045114 2.58444797 1.114913 -0.23990443 2.63400602 1.23112219 2.62013565 -0.21183287 3.41551155 2.17259946 0.31592367 0.43628453 -0.74905884 1.00689476 -0.35913151 -0.11654172 7.6107848 14.60962252 3.93794201]	-0.000000
21	[0.49893221 -0.1928123 -1.52495546 -0.51768686 -0.57401865 -1.0155776 -0.28918978 1.25939662 0.1759038 -0.094952 -0.79680412 -0.12045114 -0.77677867 -0.81857737 -0.23990443 0.80378694 -0.75919202 -1.05968807 -0.21183287 -0.93449284 -0.95104897 0.31592367 -0.1486834 0.3059892 0.27236652 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.065079
22	[-0.64918725 1.01249108 2.12800831 -0.51768686 4.34146478 2.26539305 -0.28918978 -0.29053029 1.95653085 1.89523557 1.24290125 -0.12045114 2.47126175 1.114913 -0.23990443 0.80378694 1.23112219 1.39352774 -0.21183287 2.17265315 2.17259946 1.65606634 1.10616715 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.064081
23	[1.12083025 0.07228346 0.6668228 -0.51768686 -0.57401865 1.46260236 -0.28918978 1.08693038 2.59824238 2.61246961 -0.79680412 -0.12045114 1.22217091 1.114913 -0.23990443 0.80378694 1.23112219 0.16691984 -0.21183287 0.92979475 2.17259946 0.31592367 0.23815023 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.030624
24	[-1.22324698 -0.41338738 0.6668228 -0.51768686 0.14917891 0.69006175 -0.28918978 0.13382744 0.75638332 0.55384368 -0.79680412 -0.12045114 -0.2977584 1.114913 -0.23990443 0.80378694 -0.75919202 -1.05968807 -0.21183287 -0.31306364 0.61077524 0.31592367 0.05888586 0.7855565 -0.17141096 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.028564
25	[0.0205491 -0.2551818 -0.7943627 -0.51768686 -0.57401865 1.65573751 -0.28918978 -1.2844805 0.24203438 -0.02103856 -0.79680412 -0.12045114 -0.72220674 1.114913 -0.23990443 -1.02643215 -0.75919202 0.16691984 -0.21183287 -0.31306364 -0.95104897 -1.024219 0.94577272 -0.74905884 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.022844
26	[-0.55351063 0.64121505 0.6668228 -0.51768686 -0.40451922 -1.0155776 -0.28918978 1.42732428 0.35715057 0.10762556 0.69588936 -0.12045114 0.67443043 -0.81857737 -0.23990443 0.80378694 1.23112219 1.39352774 -0.21183287 0.30836555 -0.95104897 0.31592367 0.06832083 -0.74905884 0.53251194 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.013004
27	[-0.02728921 -0.02040878 -1.52495546 3.07341864 -0.57401865 0.49227274 -0.28918978 -1.2844805 -0.98260595 -0.53569505 -0.79680412 -0.12045114 -1.10218906 1.114913 3.96564919 -2.85665123 -0.75919202 -3.51290387 -0.21183287 -2.17735124 -0.95104897 -1.024219 -0.50721213 -0.04569348 -0.70700446 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	0.012988
28	[-0.26648076 -0.35172777 -0.06376995 -0.51768686 -0.57401865 1.44165999 -0.28918978 -1.2844805 0.01670056 -0.27289174 -0.79680412 -0.12045114 -0.90815554 3.04840337 -0.23990443 -2.85665123 -0.75919202 -3.51290387 4.32076744 -1.55592204 -0.95104897 0.31592367 0.49289432 1.36103725 0.14994515 -0.35913151 -0.11654172 -0.27070619 -0.05813532 1.11989495]	0.000040
29	[-0.07512752 -0.16238815 -0.06376995 -0.51768686 -0.57401865 -1.0155776 -0.28918978 0.41748903 -0.73277933 -1.11057729 0.94158114 -0.12045114 -0.01075047 -0.81857737 -0.23990443 0.80378694 1.23112219 0.16691984 -0.21183287 -0.31306364 -0.95104897 0.31592367 0.43628453 -0.74905884 -0.17141096 -0.35913151 -0.11654172 -0.27070619 -0.05813532 -0.08783951]	-0.001388

Let's evaluate the performance of the best Lasso model that we were able to construct

In [91]:
```
# Calculate the R squared for the training and validation datasets
lasso_r_squared_train = round(lasso_best.score(lasso_X_train, lasso_y_train), 3)
lasso_r_squared_validation = round(lasso_best.score(lasso_X_validation, lasso_y_valida
print(f"The R squared for the best Lasso model for the training set is: {lasso_r_squar
print(f"The R squared for the best Lasso model for the validation set is: {lasso_r_squ

# Calculate the MSE of the Lasso model's predictions using the training and testing
lasso_y_predictions_train = lasso_model.predict(lasso_X_train)
lasso_mse_train = mean_squared_error(lasso_y_train, lasso_y_predictions_train)
print(f"The mean squared error of the Lasso Regression model's predictions using the l

lasso_y_predictions_validation = lasso_model.predict(lasso_X_validation)
lasso_mse_validation = mean_squared_error(lasso_y_validation, lasso_y_predictions_vali
print(f"The mean squared error of the Lasso Regression model's predictions using the l
```

```
The R squared for the best Lasso model for the training set is: 0.851.
The R squared for the best Lasso model for the validation set is: 0.863.
The mean squared error of the Lasso Regression model's predictions using the log home
sale prices in the training dataset is 144.98172527888673.
The mean squared error of the Lasso Regression model's predictions using the log home
sale prices in the validation dataset is 146.1750136575638.
```

Rework LASSO analysis on full set of variables including encoded

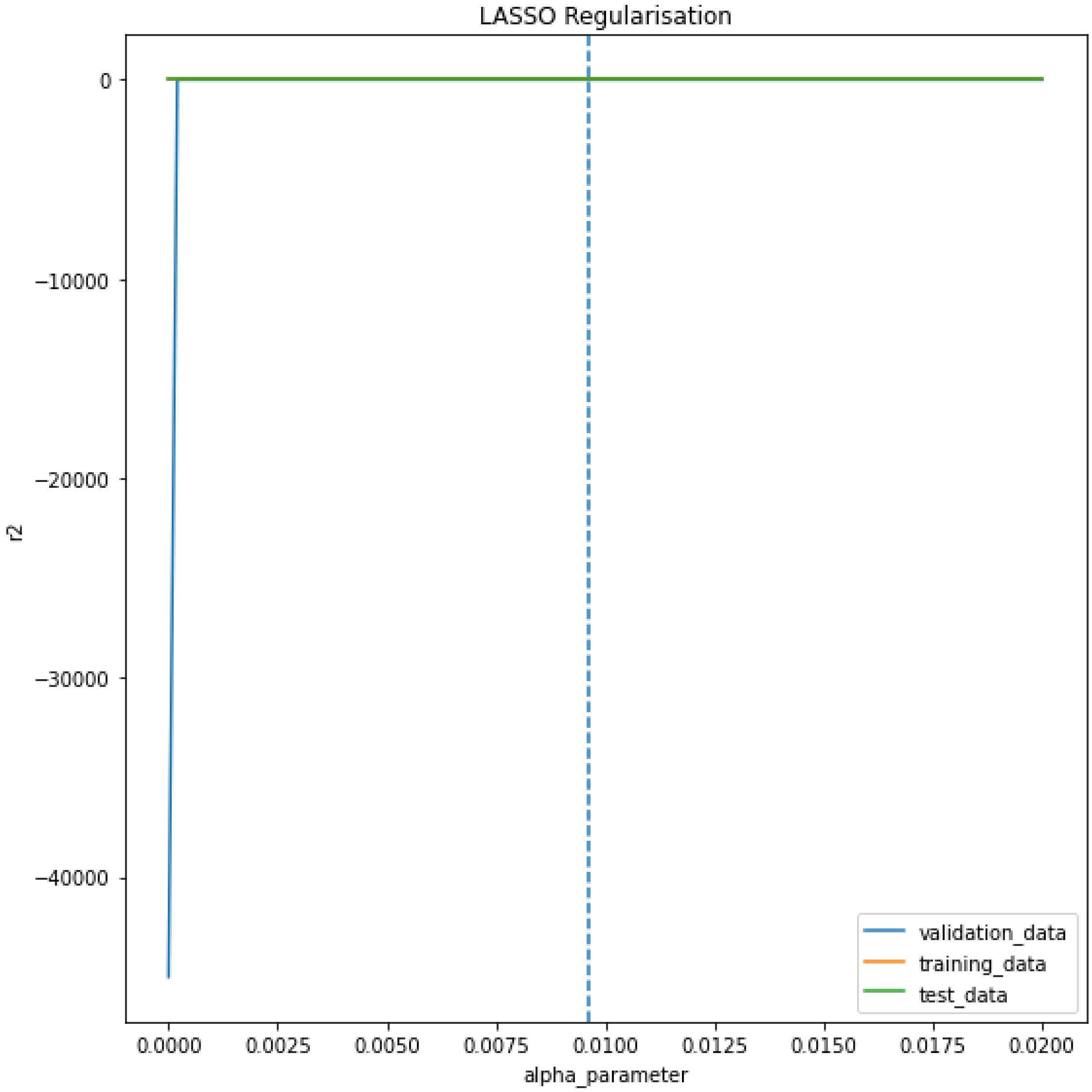
In [92]:
```
# Split the dataset into training and testing dataframes

scaler = StandardScaler()
x_raw = housing_training_data_large.select_dtypes(exclude=['object']).drop(columns = [
x_scale = xscaler.fit_transform(x_raw)
#y_scale = scaler.fit_transform(np.array(housing_training_data['SalePrice']).reshape(-
y = housing_training_data_large['SalePrice']
new_lasso_X_train, new_lasso_X_validation, new_lasso_y_train, new_lasso_y_validation =
                                                                            y, test_si
                                                                            random_sta
cv = KFold(n_splits=5, shuffle=True)

lasso_alphas = np.linspace(0, 0.02, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, x_scale, y_scale,
        cv, scoring = 'r2', model_name = 'LASSO',
        X_test = new_lasso_X_validation, y_test = new_lasso_y_validation,
        draw_plot = True, filename = 'lasso_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 35.3103798322
7483, tolerance: 0.1175439996562544
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 41.1884587561
6272, tolerance: 0.11994908768764215
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 37.2113407937
0813, tolerance: 0.11510572462008437
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 36.8492994960
1944, tolerance: 0.11130167392847584
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 37.8278550766
66005, tolerance: 0.11805824792012207
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:44: UserWarning: With alpha=0, this algorithm does not converge well.
You are advised to use the LinearRegression estimator
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 49.5886073510
90744, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.53230726080
91621, tolerance: 0.1176629040603749
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.57879148543
52749, tolerance: 0.12076187790379247
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.32350882588
03529, tolerance: 0.11008462468197588
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.53296904776
23359, tolerance: 0.11636219756433114
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.52106977290
2834, tolerance: 0.11705095348674897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.69185636815
31253, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.61277135871
3502, tolerance: 0.12034767772202624
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.59478800365
51112, tolerance: 0.1132149441195675
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.62415284785
```

```
41554, tolerance: 0.11364270760054555
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.63271476533
22243, tolerance: 0.11675327959970194
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.66818232914
09472, tolerance: 0.1179409528077164
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.65142209022
16863, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.89474017524
48201, tolerance: 0.11705600642769155
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.70265298839
25292, tolerance: 0.11494023970738529
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.72073933585
6166, tolerance: 0.12229740025743394
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.72350790526
83488, tolerance: 0.10953361819285792
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.74971139394
63134, tolerance: 0.11802449832063418
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.92663345080
7131, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.29492617996
53158, tolerance: 0.1127727010985301
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.86132869772
22856, tolerance: 0.12101298580921589
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.78439334450
```

```
01596, tolerance: 0.11781216750580778
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.76762793116
36591, tolerance: 0.11565185578018948
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.87109507098
91137, tolerance: 0.11464783456618256
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.04378349035
35048, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.88836066593
24386, tolerance: 0.11931861656109605
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.95035449356
21306, tolerance: 0.11931983307743459
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.91304865034
4507, tolerance: 0.11420908182302415
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.53232980674
61479, tolerance: 0.11525065812982799
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.51068433650
6273, tolerance: 0.1455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.14887124087
974257, tolerance: 0.11038849436093542
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.43817160077
62151, tolerance: 0.10741983918266827
  positive)
```




```
Chosen alpha: 0.00960
Validation score: 0.90266
Test score at chosen alpha: -5.52195
```

In [95]:
```
### BASED ON LASSOCV:

# Show best value of tuning parameter (alpha) chosen by cross validation
print(f"The best value of the tuning parameter, alpha, chosen by cross-validation is {

# Set best tuning parameter (alpha) and use it to fit our final Lasso regression model
lasso_best = Lasso(alpha=lasso_model.alpha_)
lasso_best.fit(x_scale, y_scale)

lasso_best.coef_
```

```
The best value of the tuning parameter, alpha, chosen by cross-validation is 0.003577
8259276540722.
```

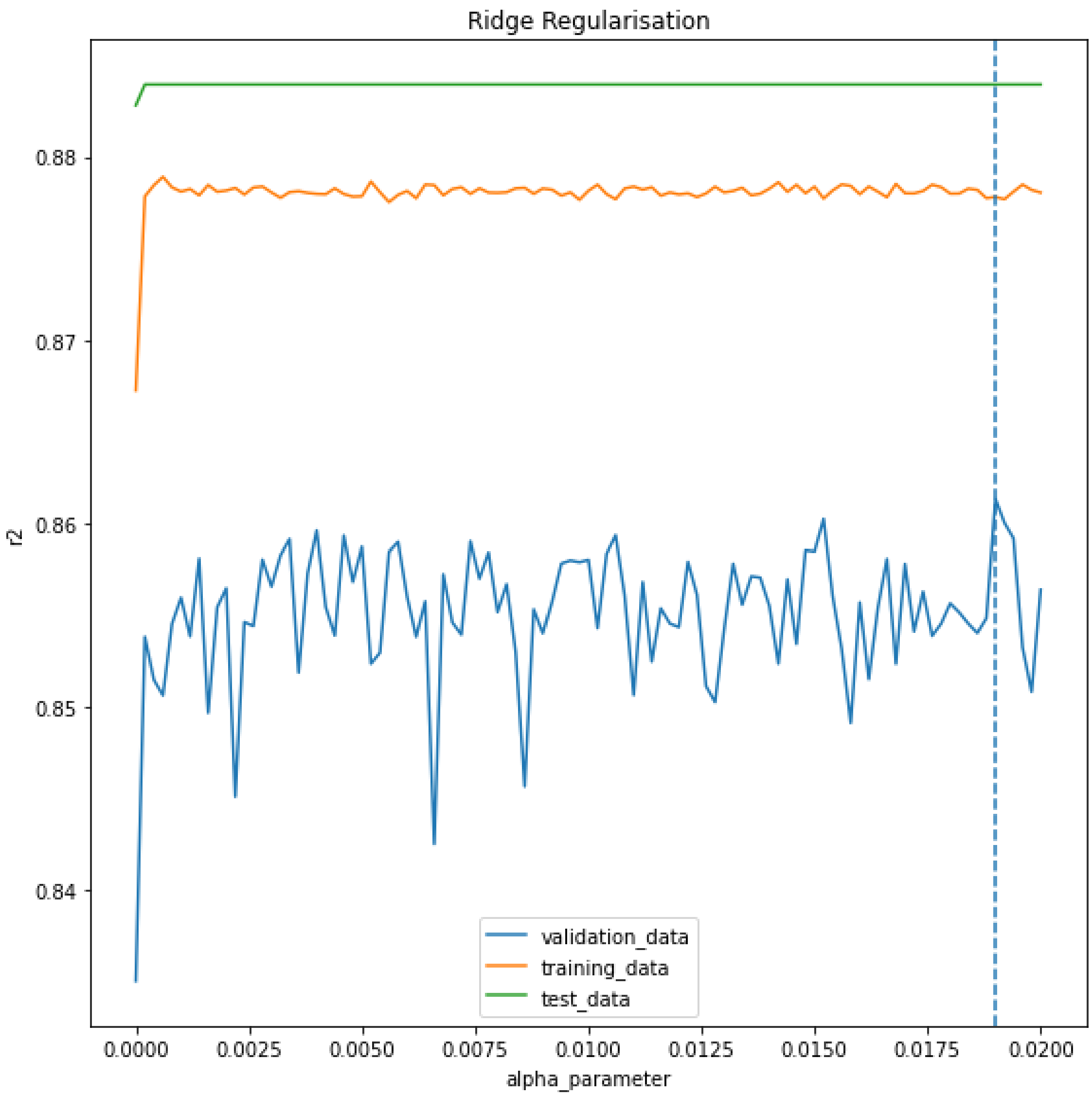
Out[95]:
```
Lasso(alpha=0.0035778259276540722, copy_X=True, fit_intercept=True,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

Out[95]:
```
array([-0.00000000e+00, -4.11641928e-02,  1.68674529e-02,  7.24623204e-02,
        1.42221206e-01,  6.37091222e-02,  6.65378779e-02,  6.44723957e-03,
        5.91435950e-02,  4.70831383e-02,  4.33865364e-02,  9.97823843e-02,
        0.00000000e+00, -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00, -1.02289184e-02,  2.09573638e-01,  3.33189236e-03,
       -5.46213183e-03,  1.39061621e-03,  1.71937649e-02, -4.92538653e-02,
       -2.95555253e-02,  5.11046900e-02,  4.97189772e-02,  1.15072485e-02,
       -0.00000000e+00,  0.00000000e+00,  3.68962259e-02,  4.41013884e-02,
        1.83428461e-02, -0.00000000e+00,  2.18563071e-02,  1.96597065e-02,
        0.00000000e+00,  9.05923345e-03,  2.55688945e-02,  7.29072340e-03,
        0.00000000e+00, -9.49453220e-03,  0.00000000e+00,  1.21863168e-02,
        0.00000000e+00,  8.79282952e-04,  1.38182045e-02, -0.00000000e+00,
        1.78978199e-01, -1.88381909e-02,  5.29227611e-03,  0.00000000e+00,
        0.00000000e+00, -1.65701625e-02, -5.31651431e-03,  1.08271695e-02,
       -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  9.79819716e-03,
       -1.36340235e-02, -0.00000000e+00,  8.64029619e-03, -2.56381400e-17,
       -1.38794294e-03,  2.78942011e-02, -7.66727125e-03, -2.95661698e-03,
        0.00000000e+00, -0.00000000e+00,  3.91229824e-03, -2.73234175e-02,
        0.00000000e+00,  7.81750228e-04,  1.26543881e-02,  2.66377844e-02,
        0.00000000e+00, -8.56622265e-04,  4.76431492e-02, -3.94484436e-03,
       -7.05947109e-04,  0.00000000e+00, -0.00000000e+00, -5.91519048e-03,
       -3.32237844e-03,  1.44601176e-02, -9.20523700e-03,  4.60277418e-02,
        9.21870562e-02, -0.00000000e+00, -0.00000000e+00,  8.47176710e-03,
        0.00000000e+00,  2.18324747e-02,  7.29085570e-02, -0.00000000e+00,
        7.89579560e-03, -1.37850752e-03, -5.16928423e-04,  3.29263639e-02,
       -0.00000000e+00,  0.00000000e+00, -1.44121563e-02,  9.93173579e-04,
       -0.00000000e+00,  1.99300323e-04,  0.00000000e+00, -0.00000000e+00,
       -0.00000000e+00,  8.20307862e-03, -5.93864085e-03, -8.25542450e-03,
        0.00000000e+00,  4.44285365e-03,  4.81227342e-02,  1.43636382e-02,
       -0.00000000e+00, -1.11366489e-02, -0.00000000e+00, -5.42761342e-03,
        2.08204330e-03,  0.00000000e+00, -3.26274640e-03, -2.84325953e-03,
       -0.00000000e+00, -3.34462380e-04,  9.68455305e-04, -0.00000000e+00,
       -1.53003023e-02, -0.00000000e+00,  1.50152661e-03,  0.00000000e+00,
        3.06431925e-03,  0.00000000e+00,  1.39802794e-02,  4.52769348e-03,
       -1.02443687e-03, -9.93173127e-03, -1.17115017e-03,  2.19971605e-02,
        0.00000000e+00, -1.29534324e-03, -0.00000000e+00,  2.47370263e-02,
       -0.00000000e+00,  4.66886522e-03, -1.06781329e-02, -0.00000000e+00,
        2.44173254e-03, -6.99740581e-03, -0.00000000e+00,  0.00000000e+00,
        0.00000000e+00, -2.17791584e-02,  0.00000000e+00,  0.00000000e+00,
       -0.00000000e+00,  0.00000000e+00, -0.00000000e+00, -0.00000000e+00,
        1.36015094e-02, -5.47260759e-03, -0.00000000e+00,  0.00000000e+00,
       -9.48333527e-03, -9.08224695e-03, -1.88499913e-03, -0.00000000e+00,
        0.00000000e+00,  7.27601399e-03, -3.62226615e-03, -1.13712099e-02,
       -2.38261473e-02,  0.00000000e+00,  1.06676924e-02,  1.07326229e-03,
        4.78343573e-03, -3.61537054e-03,  0.00000000e+00,  0.00000000e+00,
       -8.16079028e-03, -0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00, -1.15543355e-02, -1.99967616e-03, -0.00000000e+00,
        2.98255402e-02,  1.36725674e-02,  0.00000000e+00,  0.00000000e+00,
        6.97761132e-02, -1.34062669e-03, -2.05711341e-02,  0.00000000e+00,
       -0.00000000e+00,  0.00000000e+00,  1.61264655e-02, -5.88018109e-03,
        1.62512225e-03, -9.02280592e-03,  0.00000000e+00,  1.70625401e-02,
       -0.00000000e+00,  1.20341614e-02, -1.80073545e-05,  0.00000000e+00,
        0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
       -0.00000000e+00,  0.00000000e+00, -1.47006792e-02,  2.41276733e-03,
        0.00000000e+00,  0.00000000e+00,  1.38944991e-03, -7.61111234e-04,
        1.79173217e-03, -3.33226841e-03, -8.72097534e-03, -6.71160593e-03,
       -0.00000000e+00,  0.00000000e+00, -3.79393444e-03, -1.37204495e-02,
        3.87021485e-02, -1.07768334e-02, -8.66479352e-03, -0.00000000e+00,
        1.73038571e-02,  0.00000000e+00,  0.00000000e+00,  3.07172169e-02,
        0.00000000e+00,  3.11442357e-15, -2.69444207e-02, -0.00000000e+00,
```

```
-5.11827731e-03,  5.05126470e-03,  7.91052204e-03,  1.78206443e-03,
 0.00000000e+00,  0.00000000e+00,  6.37785659e-02,  2.83827499e-04,
-8.16526918e-03, -2.33267870e-02,  0.00000000e+00,  0.00000000e+00,
-9.45648888e-03,  0.00000000e+00,  0.00000000e+00])
```

REPEAT WITH RIDGE

In [96]:
```
# Split the dataset into training and testing dataframes

scaler = StandardScaler()
x_raw = housing_training_data.select_dtypes(exclude=['object']).drop(columns = ['SaleP
x_scale = xscaler.fit_transform(x_raw)
y_scale = np.array(housing_training_data['SalePrice']).reshape(-1,1)

new_ridge_X_train, new_ridge_X_validation, new_ridge_y_train, new_ridge_y_validation =
                                                                            y_scale, t
                                                                            random_sta
cv = KFold(n_splits=5, shuffle=True)

lasso_alphas = np.linspace(0, 0.02, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, new_ridge_X_train, new_ridge_y_train,
        cv, scoring = 'r2', model_name = 'Ridge',
        X_test = new_ridge_X_validation, y_test = new_ridge_y_validation,
        draw_plot = True, filename = 'ridge_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```



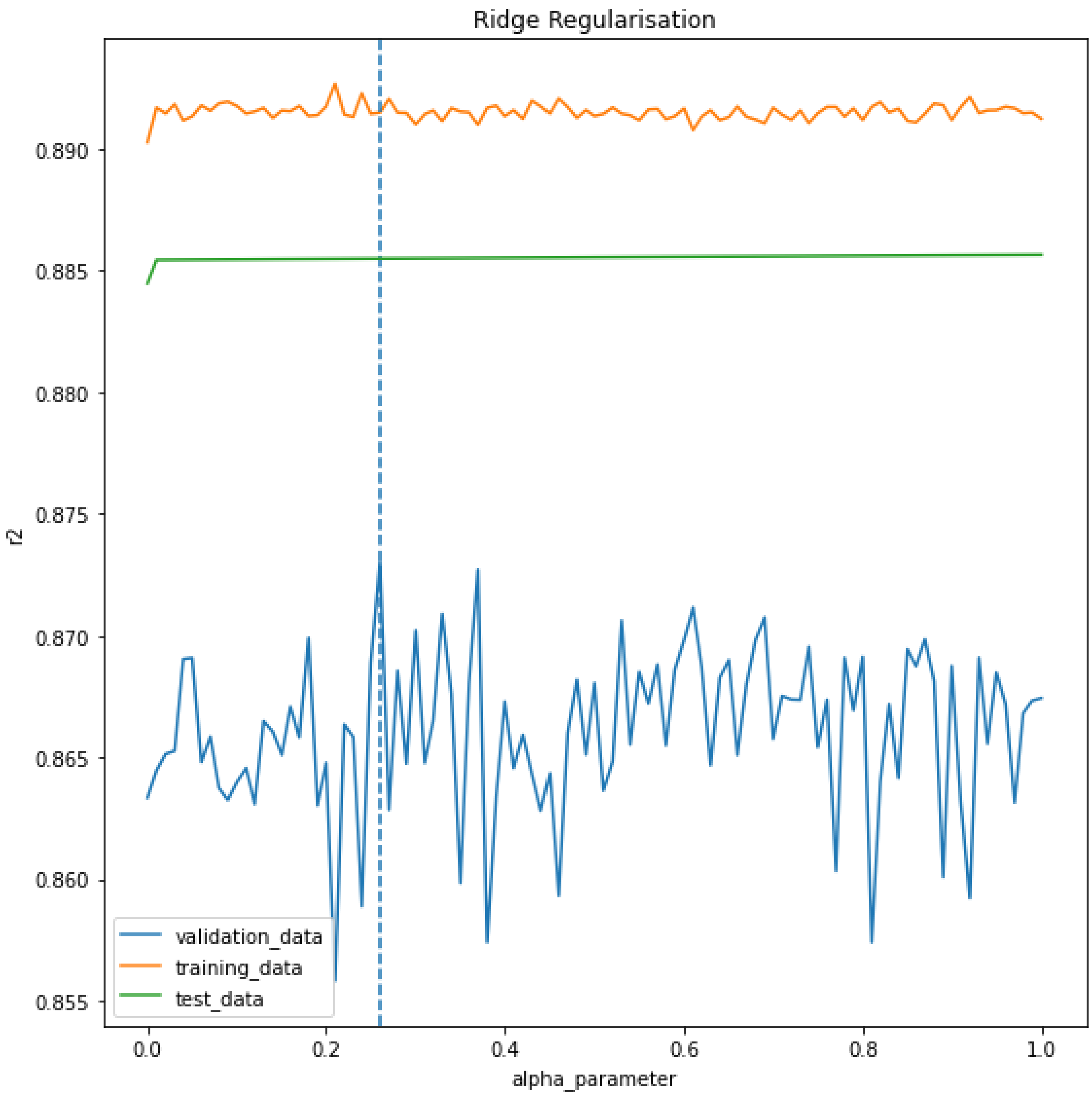

```
Chosen alpha: 0.01900
Validation score: 0.86132
Test score at chosen alpha: 0.88396
```

REPEAT WITH RIDGE USING SUBNET OF ENCODED VARIABLE SET

In [97]:

```
scaler = StandardScaler()
x_raw = housing_training_data_large_subset.select_dtypes(exclude=['object']).drop(colu
x_scale = xscaler.fit_transform(x_raw)
y_scale = np.array(housing_training_data['SalePrice']).reshape(-1,1)

new_ridge_X_train, new_ridge_X_validation, new_ridge_y_train, new_ridge_y_validation =
                                                                               y_scale, t
                                                                               random_sta
cv = KFold(n_splits=5, shuffle=True)

lasso_alphas = np.linspace(0, 1, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, new_ridge_X_train, new_ridge_y_train,
        cv, scoring = 'r2', model_name = 'Ridge',
```

```
        X_test = new_ridge_X_validation, y_test = new_ridge_y_validation,
        draw_plot = True, filename = 'ridge_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```




```
Chosen alpha: 0.26000
Validation score: 0.87303
Test score at chosen alpha: 0.88549
```

In [118…

```
# Show best Ridge model coefficients and names
regr = Ridge(alpha=chosen_alpha)
r_model = regr.fit(x_scale, y_scale)

r_model.coef_
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-118-d679e1c0ac7f> in <module>
      1 # Show best Ridge model coefficients and names
      2 regr = Ridge(alpha=chosen_alpha)
----> 3 r_model = regr.fit(x_scale, y_scale)
      4 
      5 r_model.coef_

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_ridge.py in fit(self, X, y, sample_weight)
    764         self : returns an instance of self.
    765         """
--> 766         return super().fit(X, y, sample_weight=sample_weight)
    767 
    768 

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_ridge.py in fit(self, X, y, sample_weight)
    545                                  accept_sparse=_accept_sparse,
    546                                  dtype=_dtype,
--> 547                                  multi_output=True, y_numeric=True)
    548         if sparse.issparse(X) and self.fit_intercept:
    549             if self.solver not in ['auto', 'sparse_cg', 'sag']:

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/utils/validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, o
rder, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples,
ensure_min_features, y_numeric, warn_on_dtype, estimator)
    763         y = y.astype(np.float64)
    764 
--> 765     check_consistent_length(X, y)
    766 
    767     return X, y

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/utils/validation.py in check_consistent_length(*arrays)
    210     if len(uniques) > 1:
    211         raise ValueError("Found input variables with inconsistent numbers of"
--> 212                          " samples: %r" % [int(l) for l in lengths])
    213 
    214 

ValueError: Found input variables with inconsistent numbers of samples: [1459, 1455]
```

Try ElasticNet...

In [99]:

```
# Split the dataset into training and testing dataframes

elasticscaler = StandardScaler()
x_raw = housing_training_data.select_dtypes(exclude=['object']).drop(columns = ['SaleP
x_scale = xscaler.fit_transform(x_raw)
y_scale = elasticscaler.fit_transform(np.array(housing_training_data['SalePrice']).res

new_elastic_X_train, new_elastic_X_validation, new_elastic_y_train, new_elastic_y_vali
                                                                            y_scale, t
                                                                            random_sta
cv = KFold(n_splits=5, shuffle=True)
```

```
lasso_alphas = np.linspace(0, 0.02, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, new_elastic_X_train, new_elastic_y_train,
        cv, scoring = 'r2', model_name = 'ElasticNet',
        X_test = new_elastic_X_validation, y_test = new_elastic_y_validation,
        draw_plot = True, filename = 'EN_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```

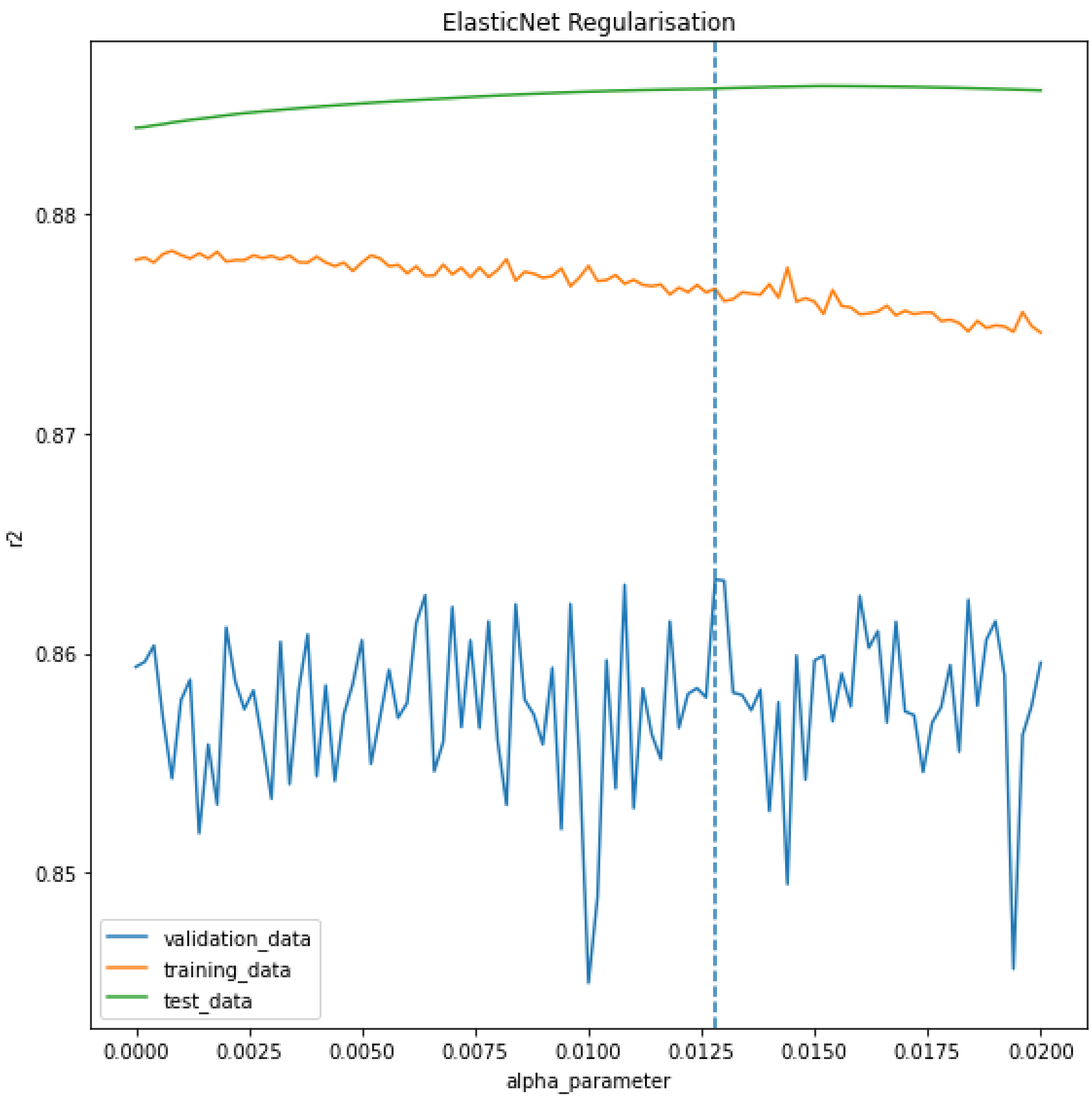
```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 49.3125375381
18214, tolerance: 0.08376794942129816
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 51.4806629249
15535, tolerance: 0.08296929794930137
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 50.3086102193
9106, tolerance: 0.0832879990067399
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 54.2289283658
1726, tolerance: 0.08423980593987467
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 47.9565815146
30635, tolerance: 0.08064710497810411
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:44: UserWarning: With alpha=0, this algorithm does not converge well.
You are advised to use the LinearRegression estimator
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 64.2155674826
2068, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 14.8324109681
52516, tolerance: 0.07984098495757876
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 17.2714897778
90878, tolerance: 0.08478563025384582
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 18.2938617689
9299, tolerance: 0.08573864849881115
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 17.1456043772
13814, tolerance: 0.08156778779801227
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 10.5966910863
72644, tolerance: 0.08301386036646154
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 21.2487475804
6724, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.58927631640
67958, tolerance: 0.08031832214608478
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.33611384848
6363, tolerance: 0.087668087863972
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.72855298349
```

```
26194, tolerance: 0.08761967961525664
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.59140017137
7643, tolerance: 0.08012040170117148
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.09457461204
0148, tolerance: 0.07922056992737007
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 2.12145323550
3438, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.50144169310
59742, tolerance: 0.08387924134623305
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.45456187827
20169, tolerance: 0.08478729028699415
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.86059401397
59554, tolerance: 0.0829635134454385
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.31158642986
90951, tolerance: 0.07829629426425311
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.15667175741
16296, tolerance: 0.08500518242117788
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.91922396447
2991, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.21984425365
16296, tolerance: 0.07805887084269865
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.63331634304
16988, tolerance: 0.07836715183096404
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.72355389023
```

```
48395, tolerance: 0.08657749475035348
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.06248143120
3679, tolerance: 0.08943469232554455
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.64926703283
87078, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.54617895968
47098, tolerance: 0.08285161879543186
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.64818024513
06606, tolerance: 0.08645037608406281
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.16631171129
20673, tolerance: 0.08853233855876791
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.86663584662
68065, tolerance: 0.07659656457777933
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.66203991493
14551, tolerance: 0.0804650902509035
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.24267570805
64101, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.84748004934
02341, tolerance: 0.08409565251635383
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.70249586936
55918, tolerance: 0.08326943437755133
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.69013453035
40012, tolerance: 0.08759098461862767
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.92795236143
```

```
25171, tolerance: 0.10374553746774125
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.59994164465
49617, tolerance: 0.08124478175214664
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.52895585352
25183, tolerance: 0.08421883793675204
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.18427547147
619805, tolerance: 0.08514750485032266
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.14913533380
695299, tolerance: 0.08255240780472282
  positive)
```




```
Chosen alpha: 0.01280
Validation score: 0.86336
Test score at chosen alpha: 0.88575
```

REPEAT WITH ELASTIC NET USING FULL ENCODED VARIABLE SET with LOG TRANSFORMED SALES PRICE

In [100…

```
# Split the dataset into training and testing dataframes

elasticscaler = StandardScaler()
x_raw = housing_training_data_large_common.select_dtypes(exclude=['object'])
x_scale = xscaler.fit_transform(x_raw)
y_scale = elasticscaler.fit_transform(np.array(np.log(housing_training_data['SalePrice

new_elastic_X_train, new_elastic_X_validation, new_elastic_y_train, new_elastic_y_vali
                                                                            y_scale, t
                                                                            random_sta
cv = KFold(n_splits=5, shuffle=True)

lasso_alphas = np.linspace(0, 0.02, 101)

chosen_alpha, max_validation_score, test_score_at_chosen_alpha = \
    regmodel_param_test(
        lasso_alphas, new_elastic_X_train, new_elastic_y_train,
        cv, scoring = 'r2', model_name = 'ElasticNet',
        X_test = new_elastic_X_validation, y_test = new_elastic_y_validation,
        draw_plot = True, filename = 'EN_wide_search')
print("Chosen alpha: %.5f" % \
    chosen_alpha)
print("Validation score: %.5f" % \
    max_validation_score)
print("Test score at chosen alpha: %.5f" % \
    test_score_at_chosen_alpha)
```

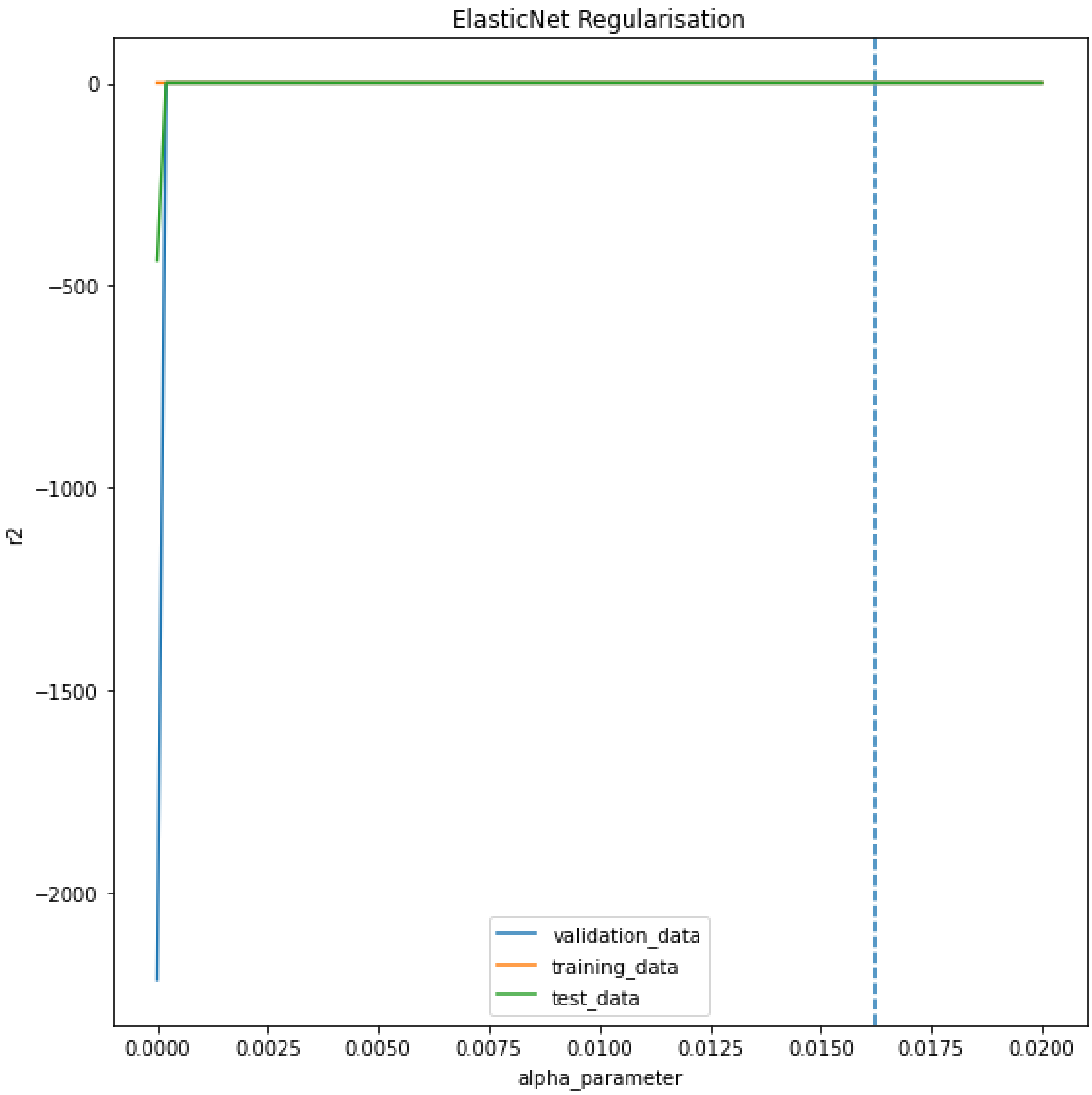
```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 16.5441130702
6035, tolerance: 0.07539341997255294
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 16.8699040526
05243, tolerance: 0.07725168572866303
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 16.7032115959
4418, tolerance: 0.0788264889258652
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 15.0946617829
26801, tolerance: 0.07603936850041647
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/model_selection/_validation.py:515: UserWarning: With alpha=0, this algorithm does
not converge well. You are advised to use the LinearRegression estimator
  estimator.fit(X_train, y_train, **fit_params)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 17.1919312264
57065, tolerance: 0.07694340945238295
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykern
el_launcher.py:44: UserWarning: With alpha=0, this algorithm does not converge well.
You are advised to use the LinearRegression estimator
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: UserWarning: Coordinate descent with no re
gularization may lead to unexpected results and is discouraged.
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 22.4779662732
66922, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.90399952595
03897, tolerance: 0.07623876165913449
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 4.06457986203
00795, tolerance: 0.07610252455058931
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 6.43442660304
82355, tolerance: 0.0759868798099048
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 8.31587049223
5138, tolerance: 0.08114298606601994
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 1.47567009231
42212, tolerance: 0.0749816143695759
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 3.11914423182
22755, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.17001955667
550916, tolerance: 0.07601276646072924
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.95842731615
08392, tolerance: 0.0777792267773953
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.17253331035
```

```
206187, tolerance: 0.07360550635910844
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.46909483036
90341, tolerance: 0.07955760019098691
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.56946233962
95739, tolerance: 0.07742283223216413
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.22512411125
43383, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.21804421017
856157, tolerance: 0.07877312557424299
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.43176096402
54183, tolerance: 0.07947117074664821
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.57062523729
28873, tolerance: 0.0758603512428788
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.22306834893
089444, tolerance: 0.07425728258752042
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.87029802379
476, tolerance: 0.07612537381325887
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.22437728195
586004, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.16971187968
005808, tolerance: 0.07634739405347273
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.14486588347
944718, tolerance: 0.07332520826459797
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.54815887425
```

```
55985, tolerance: 0.08151762677599025
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.16442160677
052797, tolerance: 0.07554830737803726
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.34365312008
965887, tolerance: 0.07776746432901534
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.21164941245
021396, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.14959850747
870718, tolerance: 0.07278342135686353
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.11671506983
923763, tolerance: 0.07719096820309204
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.15285353928
59922, tolerance: 0.08063668034245289
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.34636192639
790764, tolerance: 0.07820399985909574
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.18532742750
152664, tolerance: 0.09613331437474897
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.11294847871
543112, tolerance: 0.07960459611255288
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.33713275353
62276, tolerance: 0.07744520486830796
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.10990394275
642146, tolerance: 0.0796201418006938
  positive)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not conv
erge. You might want to increase the number of iterations. Duality gap: 0.14075228147
```

```
40787, tolerance: 0.07584299507683794
  positive)
```




```
Chosen alpha: 0.01620
Validation score: 0.92317
Test score at chosen alpha: 0.90052
```

In [101…

```
# Let's elastic build model with this alpha
regr = ElasticNet(alpha=chosen_alpha)
elastic_model = regr.fit(x_scale, y_scale)

ElasticNet(random_state=1)
print(regr.coef_)
print(regr.intercept_)

y_pred = elastic_model.predict(x_scale)

plt.subplot(1, 2, 1)
plt.scatter(y_scale, y_pred)
plt.xlabel('Sale Price (Standard Scaled)')
plt.ylabel('Predicted Sale Price')
plt.title('ElasticNet: Sale Price vs Predicted Sale Price')

# Residuals

plt.subplot(1, 2, 2)
```

```
sns.residplot(x=y_scale, y=y_pred)
plt.xlabel('Sale Price')
plt.ylabel('Residual')
plt.title('Residual Plot')


MSE = np.square(np.subtract(y_scale,y_pred)).mean()
print("MSE:",MSE)

r2 = r2_score(y_scale, y_pred)
print("R_sq:",r2)
```

Out[101]:

```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=1, selection='cyclic', tol=0.0001, warm_start=False)
```

```
[-1.25799231e-03 -0.00000000e+00  2.52828503e-02  3.99026969e-02
  1.76904467e-01  9.13294632e-02  1.02098694e-01  0.00000000e+00
  2.96895048e-03  1.61808229e-02  2.42243790e-02  6.95349451e-02
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  2.51643279e-03
  0.00000000e+00 -1.86622997e-05  1.84006566e-01  2.46860056e-02
  0.00000000e+00  1.01755101e-02  1.72673876e-02 -0.00000000e+00
 -1.65936356e-02  3.32505813e-02  4.93259348e-03  2.61716317e-02
  1.52140017e-02  0.00000000e+00  5.21161155e-02  4.10254337e-02
  2.70558643e-02  0.00000000e+00  2.30652808e-02  1.34023067e-02
  1.67813390e-03  1.34991981e-03  2.39006708e-02  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  9.41176671e-04
  3.21902052e-02  6.91241163e-03  2.64275356e-02  2.00981720e-01
 -4.01255451e-02 -7.54500592e-02  3.79073812e-03  0.00000000e+00
  0.00000000e+00 -5.01522928e-02  0.00000000e+00  3.22639563e-03
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.84472801e-02 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -3.97772115e-03 -0.00000000e+00
 -0.00000000e+00 -4.91541755e-04  1.13170266e-02  8.77678553e-03
 -0.00000000e+00  4.85363737e-02 -1.17670065e-02  0.00000000e+00
 -0.00000000e+00 -2.15960818e-02 -1.17832513e-03 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  3.80998842e-03  2.74290398e-02
 -2.23187875e-03 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  1.38466880e-02  2.84149704e-02  0.00000000e+00  3.71488597e-03
 -1.27563708e-02 -0.00000000e+00  2.59726671e-02 -0.00000000e+00
  0.00000000e+00 -9.13520856e-03  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  2.51263310e-02  0.00000000e+00
 -6.15091942e-04 -1.55519981e-02 -0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00  6.55827650e-04  0.00000000e+00 -1.29595794e-04
 -0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  7.28270473e-04
 -0.00000000e+00  0.00000000e+00 -1.33082845e-02  2.41896524e-02
 -0.00000000e+00  0.00000000e+00 -6.11469164e-03  5.43587585e-03
 -0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.15170483e-02
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -2.85742997e-03  0.00000000e+00 -0.00000000e+00  1.11322540e-02
  4.43071426e-04 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  6.20371380e-03 -0.00000000e+00 -0.00000000e+00  2.62451044e-02
  0.00000000e+00  6.22903279e-03 -4.77455788e-03 -7.79837630e-03
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  3.04113582e-02  0.00000000e+00 -1.07545621e-02
  0.00000000e+00  0.00000000e+00  0.00000000e+00  8.97144497e-03
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -2.46339358e-04
  3.10139060e-03 -3.50975469e-03  4.64976308e-03 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  7.71655985e-03 -1.56874171e-02  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -7.55102047e-03
 -2.44924883e-02  0.00000000e+00  0.00000000e+00 -9.78471378e-03
 -1.30260125e-02  2.53278389e-02 -5.09277918e-03  3.68371423e-03
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  3.26300960e-03 -0.00000000e+00  0.00000000e+00
 -6.50114630e-04 -0.00000000e+00  0.00000000e+00  8.86416409e-04
  6.24797758e-03 -0.00000000e+00  0.00000000e+00  2.68387209e-02
  0.00000000e+00 -8.86138386e-03 -4.00799608e-02  0.00000000e+00
```

```
 -0.00000000e+00 -1.30035001e-02  0.00000000e+00  0.00000000e+00]
[-2.18349378e-15]
```

Out[101]: <AxesSubplot:>

Out[101]: <matplotlib.collections.PathCollection at 0x7fee8a0ef850>

Out[101]: Text(0.5, 0, 'Sale Price (Standard Scaled)')

Out[101]: Text(0, 0.5, 'Predicted Sale Price')

Out[101]: Text(0.5, 1.0, 'ElasticNet: Sale Price vs Predicted Sale Price')

Out[101]: <AxesSubplot:>

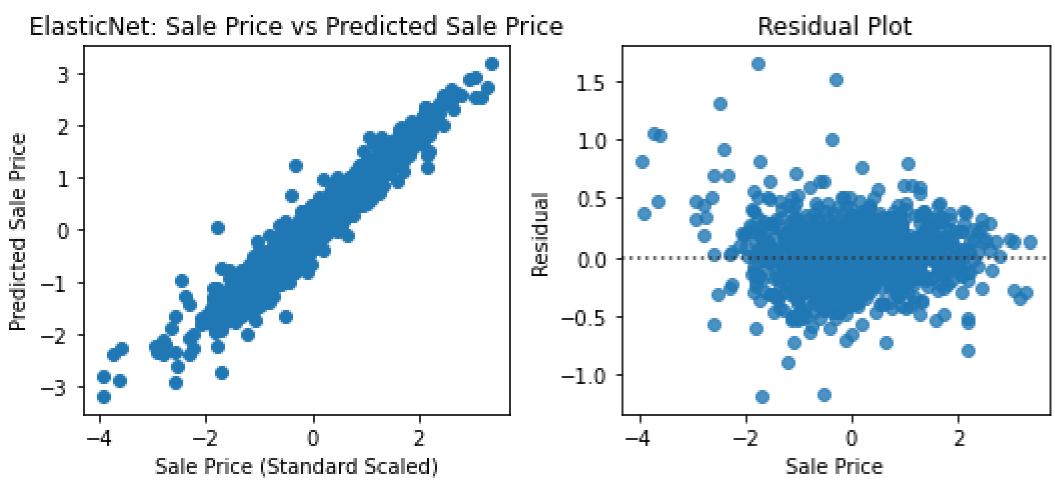
Out[101]: <AxesSubplot:>

Out[101]: Text(0.5, 0, 'Sale Price')

Out[101]: Text(0, 0.5, 'Residual')

Out[101]: Text(0.5, 1.0, 'Residual Plot')

```
MSE: 1.8975655358794232
R_sq: 0.93603420818627
```



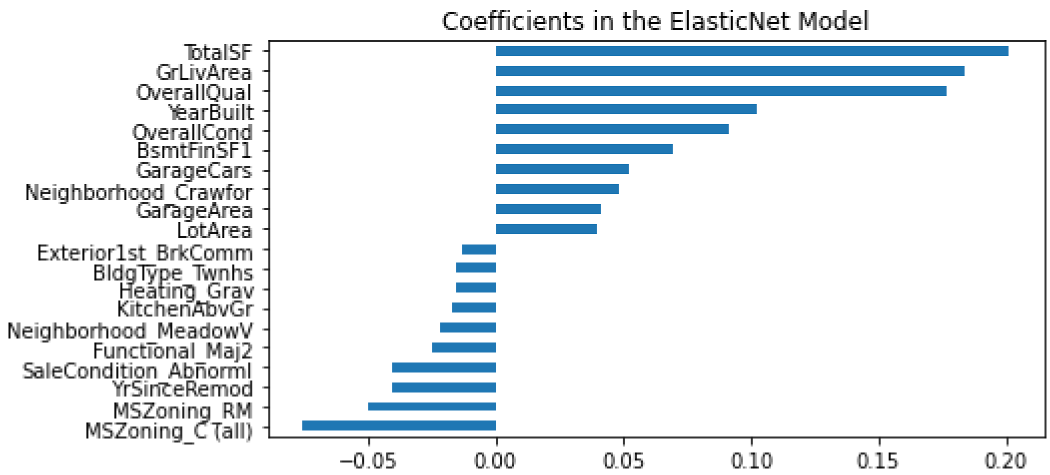
Let's check the coefficients to determine the important predictors in the model

In [102…

```
coef = pd.Series(elastic_model.coef_, index = x_raw.columns)
important_features = pd.concat([coef.sort_values().head(10),
                     coef.sort_values().tail(10)])
important_features.plot(kind = "barh")
plt.title("Coefficients in the ElasticNet Model")
```

Out[102]: <AxesSubplot:>

Out[102]: Text(0.5, 1.0, 'Coefficients in the ElasticNet Model')



TotalSF, GrLivArea, and Overall Quality appear to have strong, positive effects on Sales Price.

Try using LassoCV

In [103…

```
from sklearn.linear_model import LassoCV
from sklearn.datasets import make_regression
reg = LassoCV(cv=5, random_state=0).fit(x_scale, y_scale)
reg.score(x_scale, y_scale)
y_pred = reg.predict(x_scale)
print(reg.alpha_)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklear
n/linear_model/_coordinate_descent.py:1088: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[103]:

```
0.9346873914793059

0.009417743973449105
```

Running Models on Test Data for Kaggle Predictions

Scale x-values and transform using PCA, then use those values in the ridge model we created above to predict values of Sales Price

In [104…

```
# select numeric variables in the test data for x
x_test_raw = housing_testing_data.select_dtypes(exclude=['object'])
# scale x-values
x_test_scale = xscaler.fit_transform(x_test_raw)
# transform x-values using PCA
x_test_pca_scale = pca.fit_transform(x_test_scale)
# use x-values to predict y values
scaled_y_predtest = ridgemodel.predict(x_test_pca_scale)

# Apply inverse scaling transformation to predicted y-values
y_predtest = yscaler.inverse_transform(scaled_y_predtest)
# Exponential transform the predictions (since the model output log transformed y-valu
y_pred_final = np.exp(y_predtest)
```

Create dataframe with Id and Sales Price predictions

In [105…
```
# Create a dataframe with the y predictions
predictiondf=pd.DataFrame(y_pred_final, columns=['SalePrice'])
# Add the Id column to the front of the dataframe
predictiondf.insert(0, 'Id', housing_testing_data['Id'])

predictiondf.head()
```

Out[105]:

	Id	SalePrice
0	1461	256784.333872
1	1462	203132.211028
2	1463	161976.051358
3	1464	145088.338480
4	1465	148360.496704

In [106…
```
#output predictions to csv
predictiondf.to_csv('test_salespriceridge_v6_ridge.csv', index=False)
```

Kaggle Results - Ridge Regression

Upon submission of our home price predictions from the ridge regression model into Kaggle using Claire Markey's username, the team achieved a RMSE (as calculated using the log of the predicted and actual home prices) of 0.16954 for the testing dataset as displayed in the screenshot from the Kaggle leaderboard below.

In [107…
```
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Kaggle_RMSE_ridge.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[107]:
```
<Figure size 1080x1080 with 0 Axes>
```

Out[107]:
```
<matplotlib.image.AxesImage at 0x7fee7045d650>
```

Out[107]:
```
(-0.5, 1246.5, 126.5, -0.5)
```

Predicting Home Sale Prices In the Testing Dataset using the Polynomial Model

Let's predict Sales Price using our polynomial model

In [108…
```
poly_model = PolynomialFeatures(degree=degree)

# transform out polynomial features
poly_x_test_values = poly_model.fit_transform(x_test_pca_scale)

y_poly_pred = poly_regression_model.predict(poly_x_test_values)

# Apply inverse scaling transformation to predicted y-values
y_poly_predtest = ypolyscaler.inverse_transform(y_poly_pred)
```

In [109…
```
# Create a dataframe with the y predictions
predictionpolydf=pd.DataFrame(y_poly_predtest, columns=['SalePrice'])
# Add the Id column to the front of the dataframe
predictionpolydf.insert(0, 'Id', housing_testing_data['Id'])
predictionpolydf.head()
```

Out[109]:

	Id	SalePrice
0	1461	261902.092805
1	1462	205879.875005
2	1463	155206.325441
3	1464	135215.922537
4	1465	141244.457630

In [110…
```
#output predictions to csv

predictionpolydf.to_csv('test_salespricepoly_v5.csv', index=False)
```

Kaggle Results - Polynomial Regression

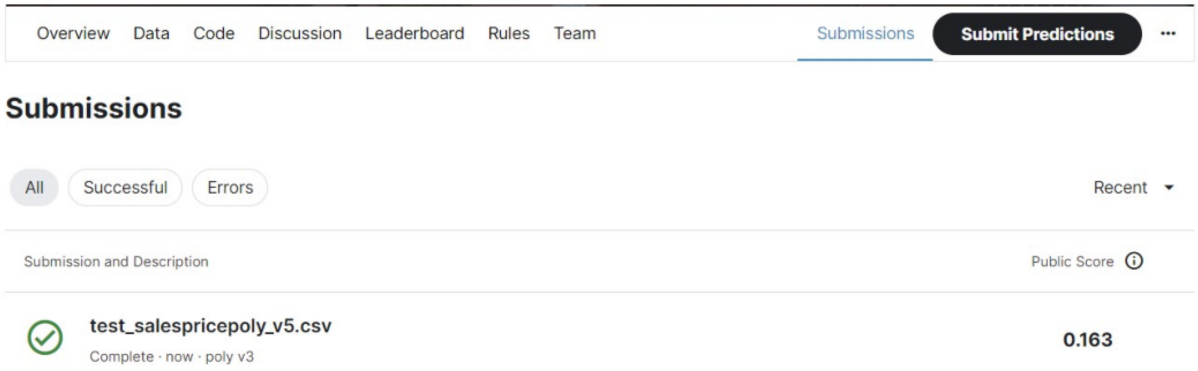
Upon submission of our home price predictions from the polynomial model into Kaggle using Claire Markey's username, the team achieved a RMSE (as calculated using the log of the predicted and actual home prices) of 0.163 for the testing dataset as displayed in the screenshot from the Kaggle leaderboard below.

In [111…
```
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Kaggle_RMSE_poly.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[111]:
```
<Figure size 1080x1080 with 0 Axes>
```

Out[111]:
```
<matplotlib.image.AxesImage at 0x7fee6f16a2d0>
```

Out[111]:
```
(-0.5, 1300.5, 420.5, -0.5)
```



Kaggle Results - Lasso Regression

In [112…

```
# Set up Test dataframe for Lasso Regression Analysis

lasso_X_test = housing_testing_data.copy(deep=True)



lasso_X_test = lasso_X_test[ ['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
                'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2
                'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', '
                'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF'
                'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal']]


from sklearn.preprocessing import StandardScaler

lasso_X_test[numerical_predictors] = lasso_scaler.transform(lasso_X_test[numerical_pre
```

In [113…

```
# Exponential transform the predictions (since the model output log transformed y-valu
lasso_y_predictions_test = lasso_model.predict(lasso_X_test)
y_predictions_final_lasso = np.exp(lasso_y_predictions_test)

# Create a dataframe with the y predictions
predictiondf_lasso=pd.DataFrame(y_predictions_final_lasso, columns=['SalePrice'])
# Add the Id column to the front of the dataframe
predictiondf_lasso.insert(0, 'Id', housing_testing_data['Id'])


#output predictions to csv
predictiondf_lasso.to_csv('test_sales_price_lasso_v1.csv', index=False)
```

In [114…

```
# Display the kaggle results associated with the Lasso Regression Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Kaggle_RMSE_lasso.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[114]:

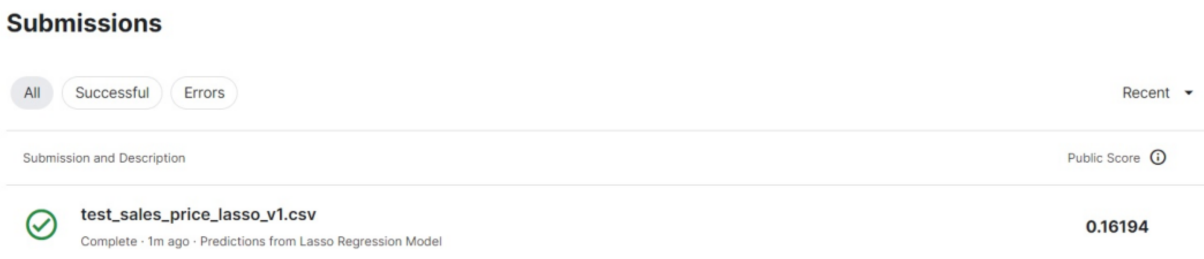
```
<Figure size 1080x1080 with 0 Axes>
```

Out[114]:

```
<matplotlib.image.AxesImage at 0x7fee6f4b4390>
```

Out[114]:

```
(-0.5, 1478.5, 335.5, -0.5)
```



Kaggle Results - Elastic Regression

In [115…

```
x_raw = housing_testing_data_large_common.select_dtypes(exclude=['object'])
x_scale = xscaler.fit_transform(x_raw)

# predict Sales Price
y_scale_elastic_pred = elastic_model.predict(x_scale)

# transform the predictions (since the model output scaled y-values)
y_log_elastic_pred = elasticscaler.inverse_transform(y_scale_elastic_pred)
y_elastic_pred = np.exp(y_log_elastic_pred)
y_elastic_pred
```

Out[115]:

```
array([124133.94005013, 160183.6995892 , 179493.67266664, ...,
       172452.63034495, 120158.64228585, 226165.83170092])
```

In [116…

```
# Create a dataframe with the y predictions
predictiondf_elastic=pd.DataFrame(y_elastic_pred, columns=['SalePrice'])
# Add the Id column to the front of the dataframe
predictiondf_elastic.insert(0, 'Id', housing_testing_data['Id'])


#output predictions to csv
predictiondf_elastic.to_csv('test_sales_price_elastic_v7.csv', index=False)
```

In [117…

```
# Display the kaggle results associated with the Lasso Regression Model
plt.figure(figsize = (15, 15))
kaggle_results = plt.imread('Kaggle_RMSE_elastic_full.jpg')
plt.imshow(kaggle_results)
plt.axis("off")
plt.show()
```

Out[117]:

```
<Figure size 1080x1080 with 0 Axes>
```

Out[117]:

```
<matplotlib.image.AxesImage at 0x7fee8b1a4390>
```

Out[117]:

```
(-0.5, 1187.5, 121.5, -0.5)
```

In [ ]: