Iowa Housing Price Prediction

Zhuoya Li, Ruyi Lu, Mingyan Xia and Jinghong Zou

December 1, 2022

Abstract

The goal of our project is to predict the house sale prices using statistical regression models based on the Iowa housing data, and this report provides a clear and detailed description on how we build our regression models from start to the end, including introduction, data cleaning, exploratory data analysis, feature selection, model construction, corresponding analysis, and discussing limitations of the model. The optimal model based on our project is the Support Vector Regression Model, using 17 predictors including "LotFrontage", "OverallQual", "Overall-Cond", "YearBuilt", "YearRemodAdd", etc.

1 Introduction

In recent decades, we have observed an increasing tendency of the real estate prices across the entire country. Together with the impact of COVID-19 on the global economy, it appears that a vast majority of people have been impacted by the economic distress, and they might need some additional information to help them make the optimized decision under such circumstances. The essential goal of our project is to serve as a tool for people to fill in an information gap and make potential adjustments to their household purchase plans, while contributing to assessing the current and future economical status from the perspective of the real estate industry.

To better understand our contructed models, we decide to apply them to the Iowa Housing Dataset, which contains more than 1400 household observations and 80 predictors measuring the interior and exterior features of the particular household. For example, our dataset takes into account the impact of different parts of the house, such as the size of garage and basement, on our response variables, the housing price.

The techniques we used for house price prediction include multiple regression models such as Lasso Regression, Random Forest Regression, Support Vector Regression, and XGBoost Models. For each model, we conduct a series of corresponding analyses to reach our conclusion. It turns out that SVR slightly out performs other models, while most of the models we applied generate a similar level of accuracy.

1.1 Notation

To clarify, in this report, the dataset of our features is X, and the predictor is y. For our model building purposes, we split the dataset into X_{train}, X_{test}, y_{train}, y_{test}. In mathematical equations, we use w as the parameters.

1.2 Organization

Our project is divided into several stages. First, we familiarize ourselves with this dataset by understanding the meaning of each variable, eyeballing whether abnormal data exists, etc. Then, we proceed to the second stage: data preparation. In this stage, we aims to make ample preparations for the upcoming model construction by manipulating and cleaning the data. In this stage, we conduct feature selection by using PCA, Random Forest, and Lasso. Our next stage is model construction and analysis. In this phase, we determine to construct several models for our dataset and evaluate which one has the highest accuracy regarding the test dataset. In the final stage, we analyze the limitations and implications of our models and our basic conclusions drawn from this project.

2 Mathematical Concepts

In this section, we will introduce mathematical proof and equations for machine learning techniques like principal concept analysis, lasso regression, random forest, support vector regression, and XGBoost. The detailed explanation of the methods are included under Section 3

2.1 Principal Component Analysis

Principle Component Analysis is a technique to reduce dimension. The goal of the algorithm is to maximize the variance of the data or minimize the mean squared distance between data points and their projections onto principal subspace. To conduct PCA, we need to first scale and center each columns.
The algorithm works as following:
1. Form the matrix $S = \frac{1}{n}\sum_{n=1}^{N}(\vec{x_n} - \bar{x})(\vec{x_n} - \bar{x})^T$
2. Eigendecompose the matrix S to find the first M eigenvectors $\vec{v_i}$ for i from 1 to M corresponding to the M largest eigenvalues.
3. Note the variance captured by the first M principal components is $\sum_{j=1}^{M}\lambda_j$, and the missed variance is $\sum_{j=M+1}^{D}\lambda_j$ where D is the column dimension of the matrix S.

2.2 Lasso Regression

$$\min_w f(w) = \frac{1}{2n_{samples}}\|Xw - y\|_2^2 + \alpha\|w\|_1 \quad (1)$$

2.3 Random Forest

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(fi - yi)^2 \quad (2)$$

2.4 Support Vector Regression

$$\begin{aligned} \min_w \quad & \frac{1}{2}w^T \cdot w \\ \text{s.t.} \quad & y_i - (w^T \cdot \phi(x) + b) \le \epsilon \\ & (w^T \cdot \phi(x) + b) - y_i \le \epsilon \end{aligned} \quad (3)$$

$$f(x) \equiv \sum_{i=1}^{l}\theta_i \phi(x, x_i) \quad (4)$$

2.5 XGBoost

$$[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \gamma T + \frac{1}{2}\lambda O_{value}^2] \quad (5)$$

3 Applications

3.1 Data Cleaning

To prepare our dataset for machine learning modeling, we first need to clean the dataset by replacing NAs, normalizing certain variables, and converting categorical variables into dummy variables.

3.1.1 Handling NAs

After further analyzing our dataset, we find out that out of the 80 predictors on housing prices, 17 of them contained missing values, and the total number entries that contain missing values are over 6000. Therefore, due to the limited size of our dataset, we decide to replace the missing values with special values instead of simply removing them, ensuring a lower variance for our models. Essentially, we replace all missing values based on different predictor types; for instance, we replace the NAs in numerical predictors with the mean of all values in that column: we have previously normalized our data, making the mean value a good representation of the predictor. Similarly, for categorical predictors, we replace the missing values with the most frequent value of the predictor.

3.1.2 Variable Normalization

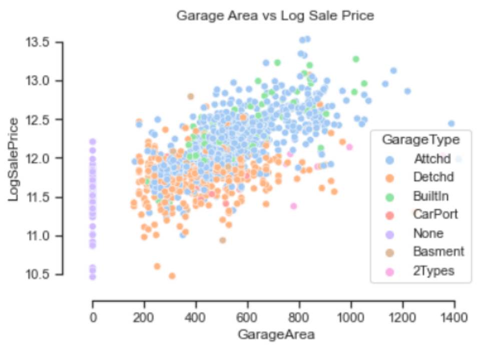
To satisfy assumptions of several regression models, we have checked the normality of our numerical variables. From the histograms of these features, we find that the graph of "SalePrice" and "LotArea" are right skewed. Thus, we apply a log transformation to these 2 variables, and the graphs look more like a Normal distribution. Since "SalePrice" is our predictor y, we decide to change our predictor to "LogSalePrice", and we still denote it with y.

3.1.3 One Hot Encoding

Some machine learning models, such as Support Vector Regression that we are implementing in our project, are based on Euclidean distances and can only take in numerical predictors, yet almost half of our predictors are categorical. If we simply exclude all categorical predictors from these models, we would result in issues with predictions and poor performance of the models. Therefore, we decide to perform One Hot Encoding, converting categorical data to dummy variables to prepare it for these algorithms and generate better predictions. We use the $get\_dummies()$ function from Pandas to assist us with the predictor conversion.

3.2 Exploratory Data Analysis

After we are done with all transformations on the dataset, we decided to perform Exploratory Data Analysis to serve as the “primitive feature selection”. Specifically, we generate a heatmap with the $heatmap()$ function in the seaborn library, which automatically find the correlations between all numerical predictors in our dataset. Based on the correlation levels between each predictor, we remove some of the predictors with a correlation score higher than 0.8 to eliminate the collinearity issues among predictors. Together with the heatmap, we also visualize the relationship between our response variables, log of housing sale price, and some predictors that are of greater importance. For example, for the scatter plot of log sale price and garage areas, we observe a general upward trend in the graph, indicating a positive correlation between the two variables; meanwhile, by labeling different garage type with different colors, we also get a sense of the distribution of each garage type with regard to garage area and log sale price respectively. Specifically, we can see that detached garage is the type of garage with the lowest overall sale price.



3.3 Dimension Reduction

After removing highly correlated columns in our dataset, we still have 286 features left, which might lead to overfitting in regression models. Therefore, we have tried 3 techniques to reduce the dimension.

The first method is principal component analysis. We first scale each column to meet the assumptions of PCA. After applying PCA to our model, we find that the first 50 components do not explain the dataset very well: they only contribute to less than 50% of the variance. The reason behind this poor performance is that our dataset contains a lot of dummy variables which contains only 0's and 1's, and our columns are not highly correlated. Thus, PCA could not successfully reduce the dimension.

Second, we use lasso feature selection method. Since lasso method adds $l1$ norm constraint to the regression, it would push as many features to 0 as possible. After we apply lasso feature selection, there are only 5 features left, which are "OverallQual", "YearBuilt", "TotalBsmtSF", "GrLivArea", and "GarageCars". We decide to disregard this selection since 5 variables would make our modeling overly simplified.

The last method we used is random forest feature selection which selects features with highest importance score, and we finally chose 17 features, which are "LotFrontage", "OverallQual", "OverallCond", "YearBuilt", "YearRemodAdd", "BsmtFinSF1", "BsmtUnfSF", "TotalBsmtSF", "2ndFlrSF", "GrLivArea", "Fireplaces", "GarageYrBlt", "GarageCars", "OpenPorchSF", "LogLotArea", "BsmtQualGd", and "CentralAirN". Now the dimension of our dataset X is 1460 rows by 17 columns.

3.4 Machine Learning Modeling

3.4.1 Train-Test Split

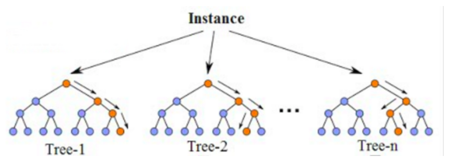
Before applying any machine learning regression methods, we first use Python's sklearn package to split our dataset into training set and test set. We will use X_{train} and y_{train} for training purpose and test our model accuracy by using X_{test} and y_{test}.

3.4.2 Lasso Regression Model

$argmin\ w\ f(w) = \frac{1}{2n_{samples}} \|Xw - y\|_2^2 + \alpha\|w\|_1$
Building upon a linear regression model, the Lasso regression model takes in data values and shrinks towards a central point as the mean, which could avoid the problem of overfitting. As we can see in the mathematical equation of Lasso regression, the first term is the sum of squared residuals, just as in the case of linear regression. The second term is called the L1 regularization term, also referred to as the Lasso regression penalty term. Such a term includes a real number(α) in $\mathbb{R}$, ranging from 0 to ∞, by multiplying the absolute value of the weight (the slope). We determined the hyperparameter α through cross validation. Compared to Ridge, which regularizes the equation in L2, Lasso can exclude useless variables from equations. Since our dataset has a relatively large number of features, considering there might be a higher possibility that some of them are unrelated to predicting the target value, we chose Lasso over Ridge to eliminate the effect of these variables.
Lasso Regression model, it resulted in a Root Mean Square Error of 0.1366 and R^2 of 0.8878 in predicting our test dataset.

3.4.3 Random Forest Model

Random forest is one of the most accurate learning algorithms available and can produce a high predicting accuracy in a large dataset. Random forest considers all possible splits on a random sample of a small subset of all variables.
Using Random Forest Algorithm to solve regression problems, we use mean squared error (MSE) to see how data branches from each node. The equation for MSE is $MSE = \frac{1}{N}\sum_{i=1}^{N}(fi - yi)^2$. This formula helps us calculate the distance of each node from the predicted actual value, letting us make the decision of which branch is the better decision for this whole forest.
Hence, the random forest is able to perform a stronger prediction accuracy. Specifically, it avoids the stronger biased predictor impact. Besides, it is easy to implement the model of random forest and it can estimate the importance of the variables in the regression analysis process.
Before fitting the model to our dataset, the first step is to tune the model for the best hyperparameters $n_{estimators}$, which gives us the best cross-validation score. To avoid overfitting, we run a for-loop for $n_{estimators}$ value from 1 to 50 with the predetermined $maxdepth = 5$. In each iteration, we record the current value of the cross-validation score and check whether this current value is the best cross-validation score. This loop helps us find the best $nestimators$ value for this data, which is 38. Finally, we are able to train our Random Forest model with the hyperparameters value($n_{estimators} = 32$, $maxdepth = 5$) and fit test data into this model. Finally, we obtain 0.8975 R^2 in the train data set and 0.8381 R^2 in the test data set.



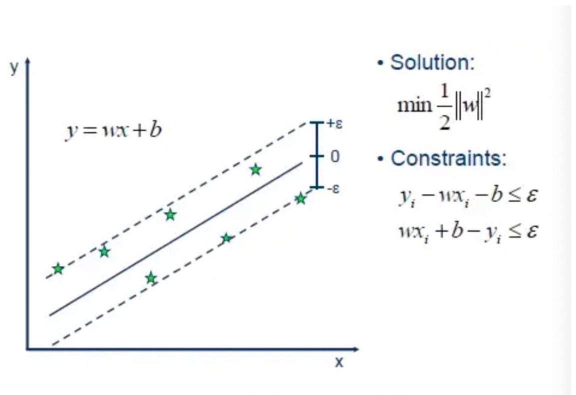
3.4.4 Support Vector Regression Model

Support Vector Regression is one of the supervised machine learning techniques. It aims to find an optimal hyperplane such that this plane can capture most of the data points. For other regression models, most of them are trying to minimize the difference between the actual value and predicted value. However, SVR is trying to find an optimal hyperplane in a given threshold, whose value is given by the distance between boundary line and hyperplane. In SVR model, we basically want to solve the following optimization problem:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}w^T \cdot w \\ \text{s.t.} \quad & y_i - (w^T \cdot \phi(x) + b) \le \epsilon, \\ & (w^T \cdot \phi(x) + b) - y_i \le \epsilon \end{aligned}$$

After training the the model, we use the following equation $f(x) \equiv \sum_{i=1}^{l} \theta_i \phi(x, x_i)$ to

generate predicted value for our test set.
In the construction of our SVR model, the first step we do is utilizing cross validation to optimize the hyperparameters. In our case, we assume the kernel in our SVR model to be Radial Basis Function, and we want to optimize the gamma value. Hence, we run gamma values from 0.01 to 5 in increments of 0.01 in a for loop. In each iteration, we record the current value of the cross-validation score and check whether this current value is the best cross-validation score. This loop helps us find the best gamma value for this data, which is 0.1. Finally, we are able to train our SVR model with the hyperparameters value($\gamma = 0.1$, $kernel = "rbf"$) and fit test data into this model. We obtained 0.9135 R^2 in the training set and 0.8940 R^2 in the test set.


y
$y = wx + b$
$+\varepsilon$
0
$-\varepsilon$
x
• Solution:
$\min \frac{1}{2}\|w\|^2$
• Constraints:
$y_i - wx_i - b \leq \varepsilon$
$wx_i + b - y_i \leq \varepsilon$

Support Vector Regression (SVR)

3.4.5 XGBoost Model

The last model we used to train our dataset is the XGBoost model. The logistics behind this model is as follows: first, we need to initialize the model with a constant value, which is the mean of the housing sales price. Next, we calculate the error for each observation based on this model, which is so-called the pseudo-residuals. The third step is actually to fit a decision tree regressor base estimator to predict the errors. And then we add the prediction from this model to the ensemble of models until we result in the final predictor. The mathematical equation behind XGBoost model:

$$[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \gamma T + \frac{1}{2}\lambda O_{value}^2]$$

where γ is a user definable penalty meant to encourage pruning and T is the number of Terminal nodes, or leaves, in the decision tree.

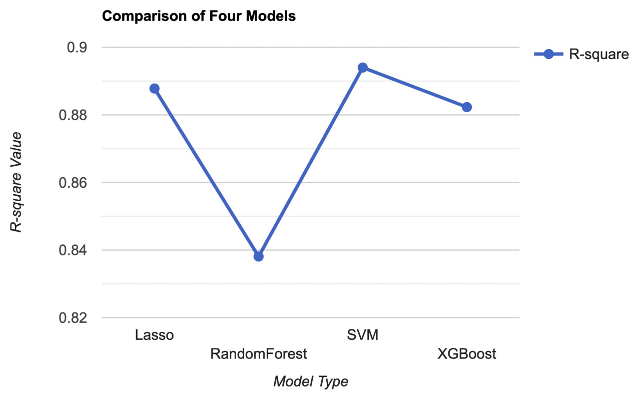
To interpret this equation, the first part is the loss function where it just represents the distance between the actual value and prediction plus the errors. The second term can be omitted as it does not affect the optimal output value. And the last term is the regularization term similar to the one in Ridge (L2 dimension). XGBoost uses the loss function for each model to build update the trees by minimizing the above equation. The goal is always to find an output value that would minimize the whole equation. And if we use the Second Order Taylor Approximation to derive the output value, we would result in a specific formula for the O_value for a leaf when using XGBoost for regression:

$$O_{value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

In general, the fundamental idea behind XGBoost is that it is a Ensemble learning technique where each tree aims to improve the result of the previous tree, by optimizing the difference between the output value and its predictions.
Applying XGBoost model, it resulted in a Root Mean Square Error of 0.14 and R^2 of 0.8823 in predicting our test dataset.

4 Conclusions

4.1 Final Result



Combining the result of the four models, we can see that the best performing model to predict the housing price in Iowa based on our dataset is Support Vector Regression Model. Among Lasso, SVR, and XGBoost, the three resulted in close value of both RMSE and R-squared. And they all avoided the problem of overfitting or underfitting, so they could all be potential valid candidates if we want to further apply the model in a larger housing market or across a longer time frame.

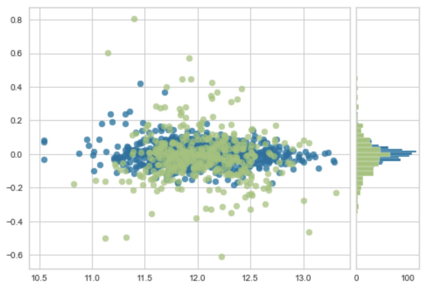


Figure 1: *Visualization of Residuals with XGBoost*

4.2 Limitations

Although the model performance statistics like R^2 and mean squared error have both shown that our model has achieved an accuracy of 90%, there are still many limitations of our project.
The first limitation comes from the housing price dataset we chose. Since the dataset contains less than 2000 rows, we could not generalize our findings to a larger scale. Also, because of the small input size, it is very likely that we have overfitted our models.
Secondly, the dataset only contains information of housing in Iowa State. It would be better if we can access to a larger dataset which record information for all the states in United States. Even if we can run our models on states like California, which represents almost half of the GDP in the US, our result can better reflect and be applied to the housing market in general. In such cases, we could generate a more thorough view on United State's housing market, and we could conduct research on comparing the differences in different regions.
Moreover, we could try different methods when handling missing values. For now we simply imputed numerical NAs with the mean of the column and categorical NAs with the mode of the column. Instead, we could examine more closely to each feature, and figure out the best way of handling the missing values. For example, for categorical variables like quality score of the housing material, we can use mode to replace NAs. However, for variables like years of house built, we should try a different approach.
When we build our models, we have ignored the time effect in the housing market. In fact, housing price could change due to historical events and inflation. If we want to make our models better matches the real life scenario, we could use additional dataset which tells us how the value of money has changed over years so that we can reduce the impact of inflation in house sale prices.

References

[1] [Chazzer]. ([2022]). [Kaggle]. Retrieved [12/1/2022] from https://www.kaggle.com/code/chazzer/ml-grandmaster-gradient-boosting-xgboost/notebookF09F9A80-Gradient-Boosting-Algorithm.

[2] [Dean De Cock]. ([2019]). [Ames Housing dataset]. Retrieved [11/2/2022] from [https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data.

[3] [Dinesh Kumar]. ([2022]). [Great Learning]. Retrieved [12/1/2022] from [https://www.mygreatlearning.com/blog/-understanding-of-lasso-regression/.

[4] [Madison Schott]. ([2019]). [Capital One Tech]. Retrieved [12/1/2022] from https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb.

[5] [StatQuest]. ([2020]). [Josh Starmer]. Retrieved [12/1/2022] from https://www.youtube.com/watch?v=ZVFeW798-2It=358s.

[6] [Statista Research Department]. ([2022]). [Average sales price of new homes sold in the U.S. 1965-2021]. Retrived [2022] from https://www.statista.com/statistics/240991/-average-sales-prices-of-new-homes-sold-in-the-us/.

[7] [Support Vector Machine-Regression]. ([2020]). [R Reference]. Retrieved [12/1/2022] from https://www.saedsayad.com/supportvector-machinereg.html.

In [1]:
```
import numpy as np
import pandas as pd
import numpy.linalg as la
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from scipy import stats as st
import seaborn as sns
import math
```

In [2]:
```
train_new = pd.read_csv("train.csv")
```

Handling Missing NAs

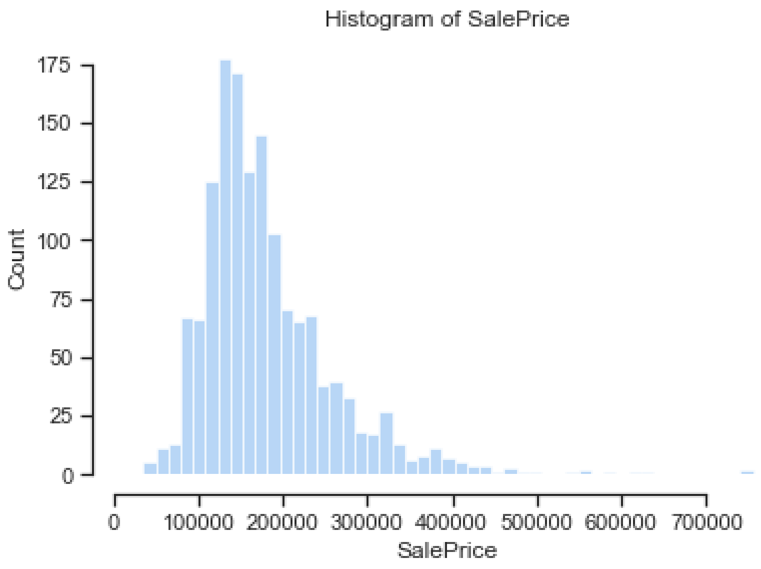
In [3]:
```
def handle_missing(train_new):
    train_new['Functional'] = train_new['Functional'].fillna('Typ')
    train_new['Electrical'] = train_new['Electrical'].fillna("SBrkr")
    train_new['KitchenQual'] = train_new['KitchenQual'].fillna("TA")
    train_new['Exterior1st'] = train_new['Exterior1st'].fillna(train_new['Exteri
    train_new['Exterior2nd'] = train_new['Exterior2nd'].fillna(train_new['Exteri
    train_new['SaleType'] = train_new['SaleType'].fillna(train_new['SaleType'].m
    train_new['MSZoning'] = train_new.groupby('MSSubClass')['MSZoning'].transfor
    train_new["PoolQC"] = train_new["PoolQC"].fillna("None")
    for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
        train_new[col] = train_new[col].fillna(0)
    for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
        train_new[col] = train_new[col].fillna('None')
    for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFin
        train_new[col] = train_new[col].fillna('None')

    train_new['LotFrontage'] = train_new.groupby('Neighborhood')['LotFrontage'].
    objects = []
    for i in train_new.columns:
        if train_new[i].dtype == object:
            objects.append(i)
    train_new.update(train_new[objects].fillna('None'))

    numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64'
    numeric = []
    for i in train_new.columns:
        if train_new[i].dtype in numeric_dtypes:
            numeric.append(i)
    train_new.update(train_new[numeric].fillna(0))
    return train_new

train_new = handle_missing(train_new)
```

Take Log of Predictor and Area

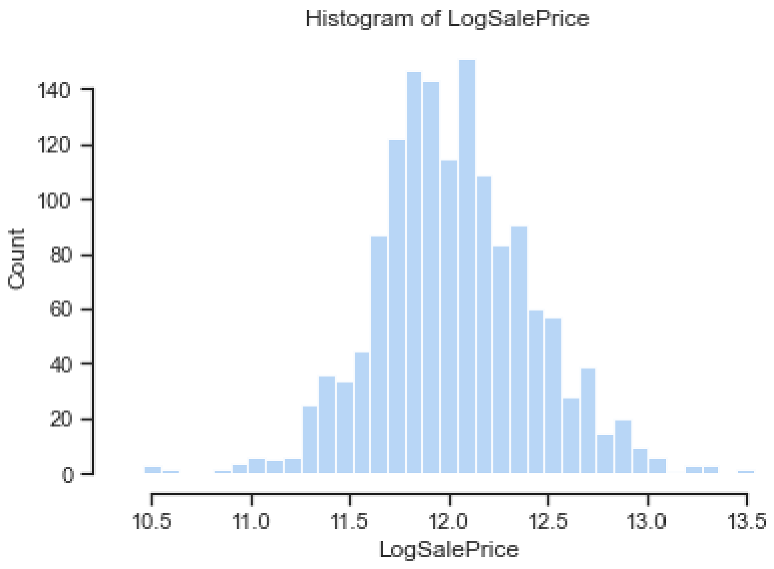
In [4]:
```
train_new['LogSalePrice'] = np.log(train_new['SalePrice'])
```

```
train_new['LogLotArea'] = np.log(train_new['LotArea'])
```

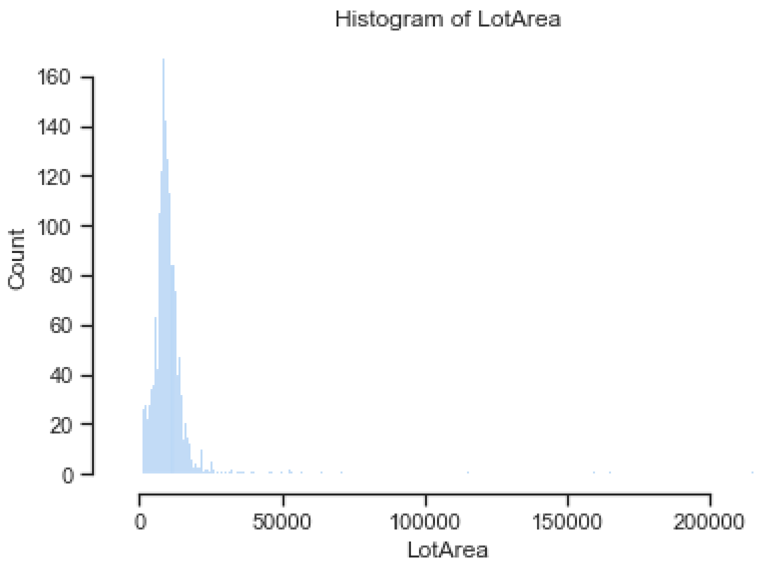
In [5]:
```
# fix the skewness of sales price
sns.set_theme(style="ticks", palette="pastel")
sns.histplot(data=train_new, x="SalePrice").set(title='Histogram of SalePrice')
sns.despine(offset=10, trim=True)
```



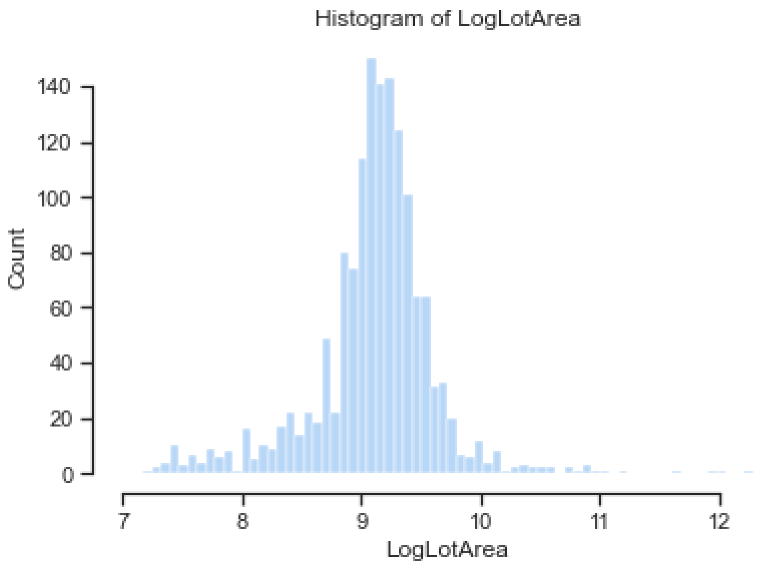
In [6]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.histplot(data=train_new, x="LogSalePrice").set(title='Histogram of LogSalePr
sns.despine(offset=10, trim=True)
```



In [7]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.histplot(data=train_new, x="LotArea").set(title='Histogram of LotArea')
sns.despine(offset=10, trim=True)
```



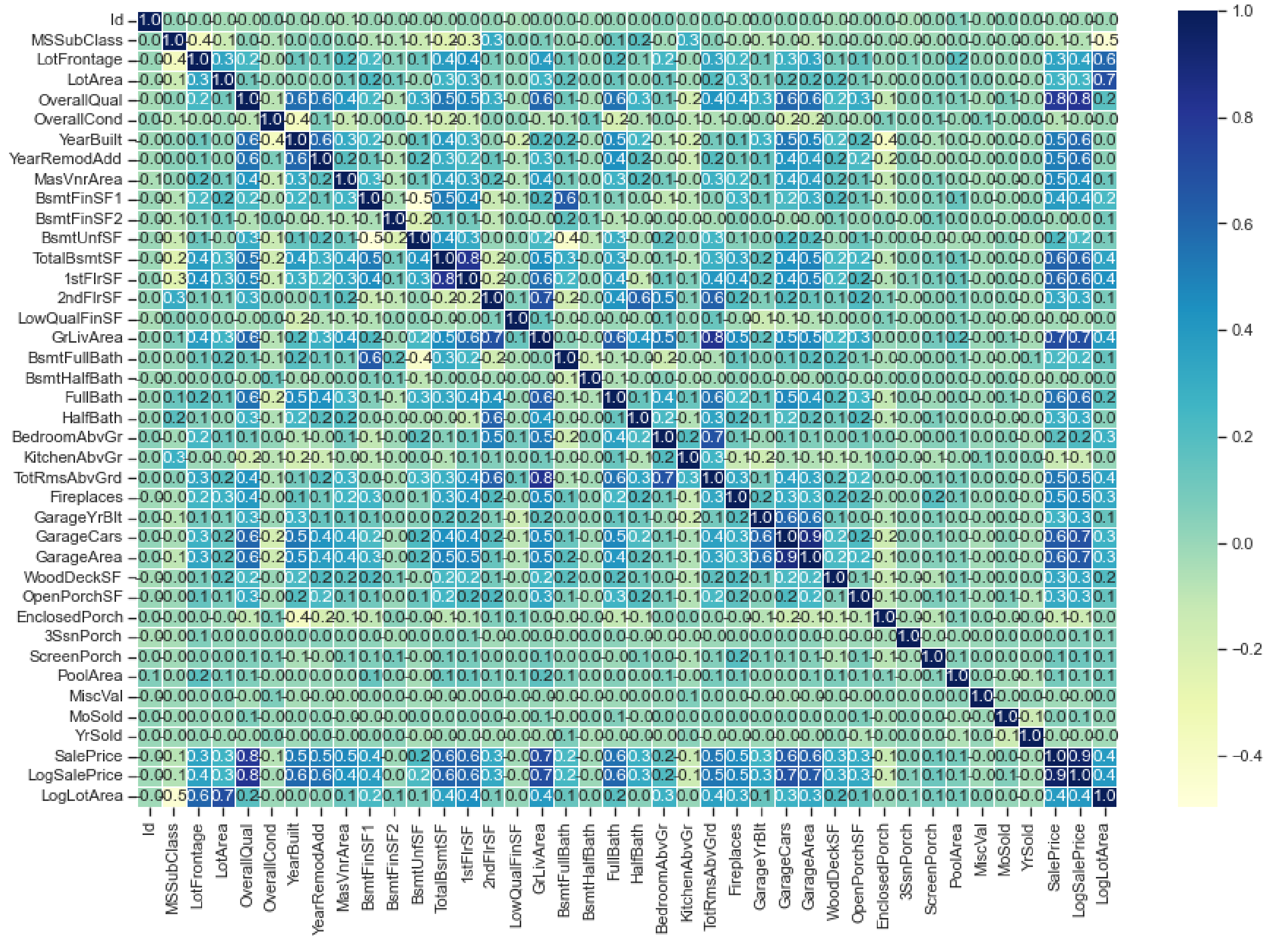
```
In [8]: sns.set_theme(style="ticks", palette="pastel")
        sns.histplot(data=train_new, x="LogLotArea").set(title='Histogram of LogLotArea'
        sns.despine(offset=10, trim=True)
```



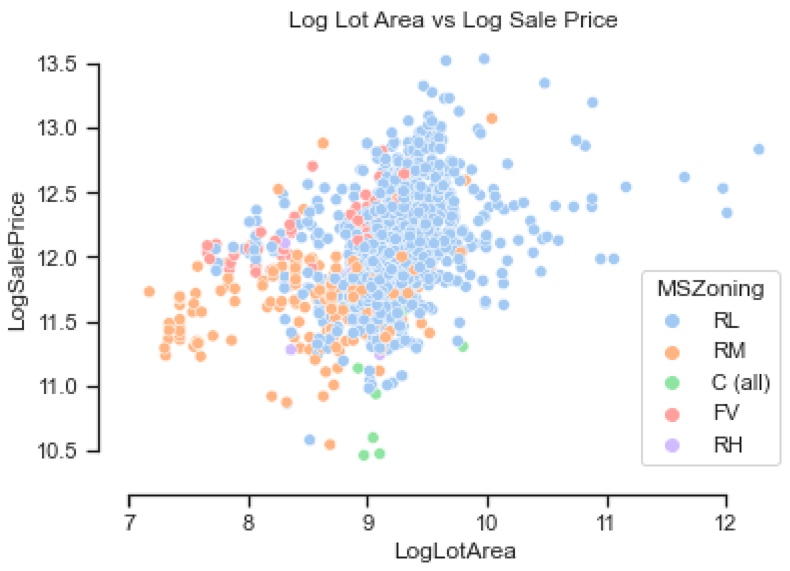
EDA

```
In [9]: f,ax = plt.subplots(figsize=(15, 10))
        sns.heatmap(train_new.corr(), cmap="YlGnBu", linecolor="white", annot=True, line
```

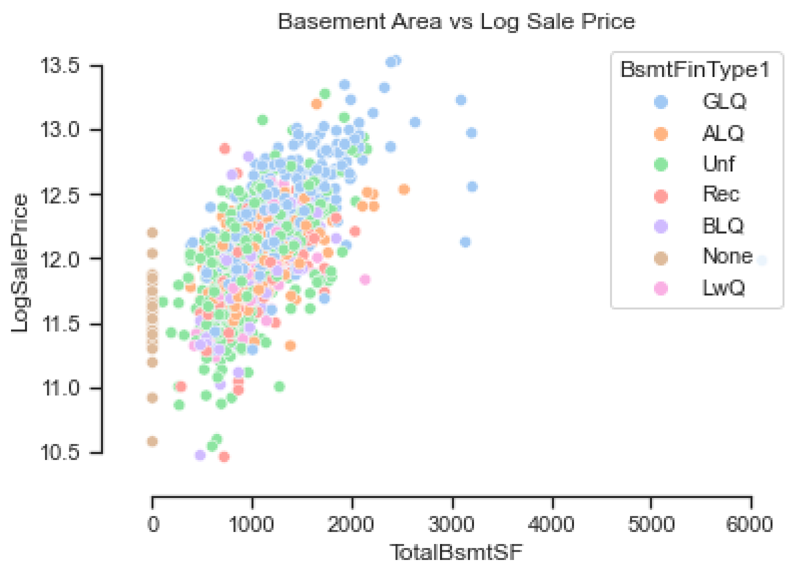
```
Out[9]: <AxesSubplot:>
```



In [10]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.scatterplot(x="LogLotArea", y="LogSalePrice",
                hue="MSZoning",
                data=train_new).set(title='Log Lot Area vs Log Sale Price')
sns.despine(offset=10, trim=True)
```

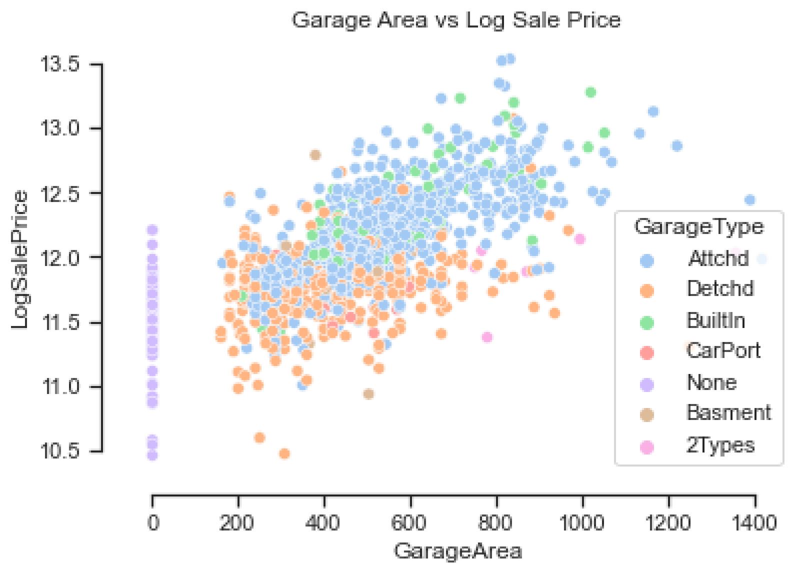


In [11]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.scatterplot(x="TotalBsmtSF", y="LogSalePrice",
                hue="BsmtFinType1",
                data=train_new).set(title='Basement Area vs Log Sale Price')
sns.despine(offset=10, trim=True)
```



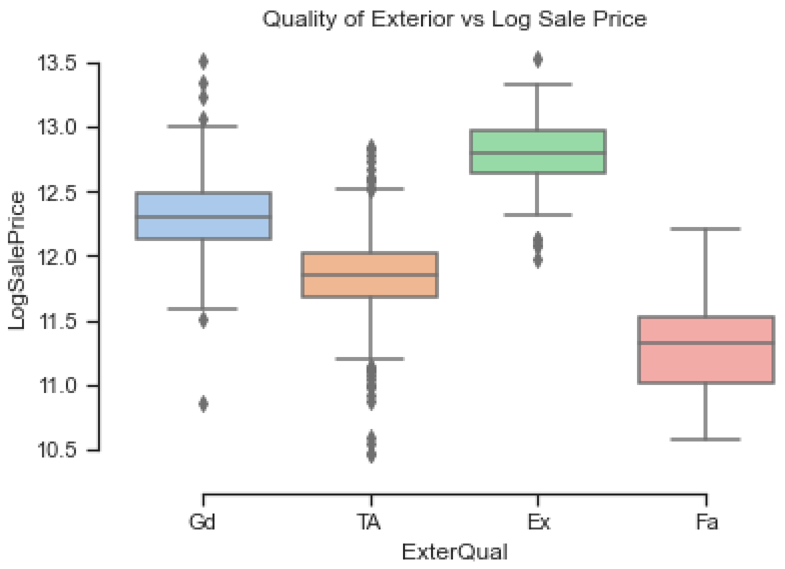
In [12]:

```
sns.set_theme(style="ticks", palette="pastel")
sns.scatterplot(x="GarageArea", y="LogSalePrice",
                hue='GarageType',
                data=train_new).set(title='Garage Area vs Log Sale Price')
sns.despine(offset=10, trim=True)
```

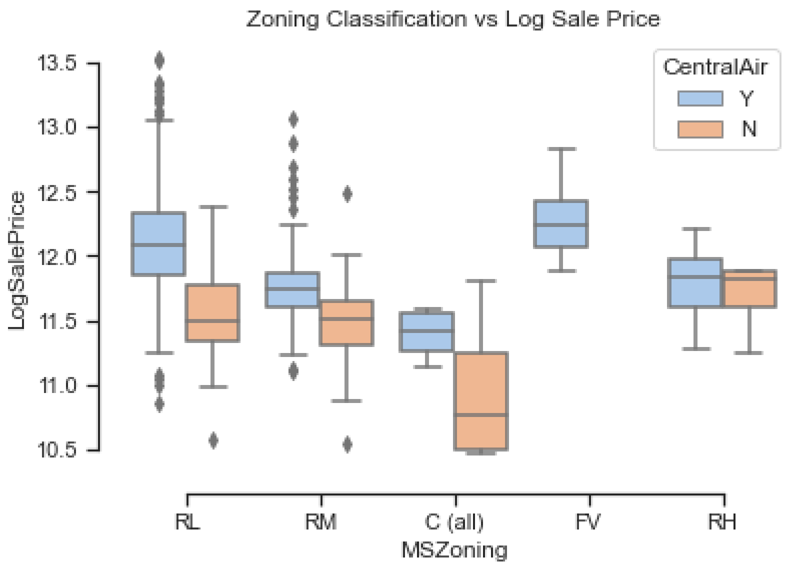


In [13]:

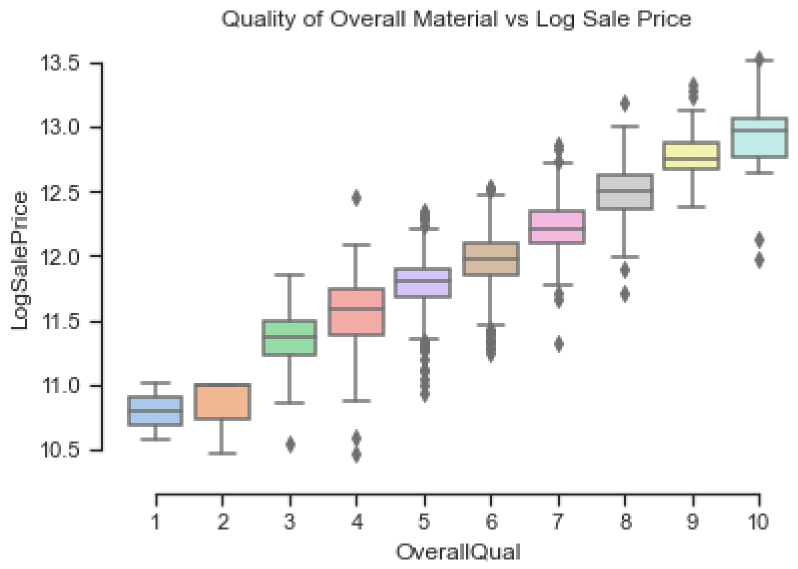
```
sns.set_theme(style="ticks", palette="pastel")
sns.boxplot(x="ExterQual", y="LogSalePrice",
            data=train_new).set(title='Quality of Exterior vs Log Sale Price')
sns.despine(offset=10, trim=True)
```



In [14]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.boxplot(x="MSZoning", y="LogSalePrice",
            hue = "CentralAir",
            data=train_new).set(title='Zoning Classification vs Log Sale Price')
sns.despine(offset=10, trim=True)
```



In [15]:
```
sns.set_theme(style="ticks", palette="pastel")
sns.boxplot(x='OverallQual', y="LogSalePrice",
            data=train_new).set(title='Quality of Overall Material vs Log Sale P
sns.despine(offset=10, trim=True)
```



Data Processing

In [16]:

```
# convert non-numeric predictors into strings
train_new['MSSubClass'] = train_new['MSSubClass'].apply(str)
train_new['YrSold'] = train_new['YrSold'].astype(str)
train_new['MoSold'] = train_new['MoSold'].astype(str)
```

In [17]:

```
train_new = pd.get_dummies(train_new).reset_index(drop=True)
```

In [18]:

```
train_new.drop(['Id','SalePrice','LotArea'], axis=1, inplace=True)
train_new = train_new.loc[:,~train_new.columns.duplicated()]
```

PCA

In [19]:

```
cor_matrix = train_new.corr().abs()

upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(bool)
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.8
train_new1 = train_new.drop(train_new[to_drop], axis=1)
train_new1['LogSalePrice'] = train_new['LogSalePrice']
train_new1
```

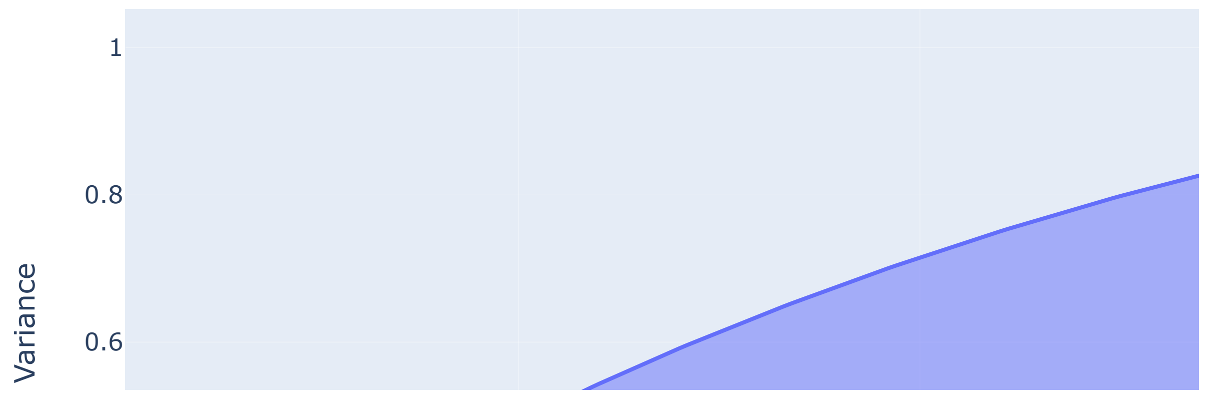
Out[19]:

	LotFrontage	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
0	65.0	7	5	2003	2003	196.0	706
1	80.0	6	8	1976	1976	0.0	978
2	68.0	7	5	2001	2002	162.0	486
3	60.0	7	5	1915	1970	0.0	216
4	84.0	8	5	2000	2000	350.0	655
...	...	...	...	...	...	...	...

	LotFrontage	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1
1455	62.0	6	5	1999	2000	0.0	0
1456	85.0	6	6	1978	1988	119.0	790
1457	66.0	7	9	1941	2006	0.0	275
1458	68.0	5	6	1950	1996	0.0	49
1459	75.0	5	6	1965	1965	0.0	830

1460 rows × 286 columns

In [20]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import plotly.express as px

scale_df = StandardScaler().fit_transform(train_new1.loc[:, train_new1.columns !
pca = PCA()

pca.fit(scale_df)
exp_var_cumul = np.cumsum(pca.explained_variance_ratio_)

px.area(
    x=range(1, exp_var_cumul.shape[0] + 1),
    y=exp_var_cumul,
    labels={"x": "# Components", "y": "Explained Variance"}
)
```



Because the correlation between each feature is not highly correlated, PCA does not work very well in this case. The reason is that we use dummy variables for categorical variables. Instead, we will try to use random forest feature selection to reduce dimension.

In [21]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split, GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(train_new1.loc[:, train_new1
                                                    train_new1['LogSalePrice'],
                                                    test_size = 0.3)

sel = SelectFromModel(estimator=RandomForestRegressor(n_estimators = 50,
                                                     random_state = 22))
sel.fit(X_train, y_train)
sel.get_support()
selected_feat = X_train.columns[(sel.get_support())]
print(len(selected_feat))
selected_feat
```

```
17
```

Out[21]:
```
Index(['LotFrontage', 'OverallQual', 'OverallCond', 'YearBuilt',
       'YearRemodAdd', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '2ndFlrSF',
       'GrLivArea', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'OpenPorchSF',
       'LogLotArea', 'BsmtQual_Gd', 'CentralAir_N'],
      dtype='object')
```

In [34]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
features = X_train.columns
pipeline = Pipeline([('scaler',StandardScaler()),
                     ('model',Lasso())])
search = GridSearchCV(pipeline,
                      {'model__alpha':np.arange(0.1,10,0.1)},
                      cv = 10, scoring="neg_mean_squared_error",verbose=0)
search.fit(X_train,y_train)
search.best_params_
coefficients = search.best_estimator_.named_steps['model'].coef_
importance = np.abs(coefficients)
np.array(features)[importance > 0]
```

Out[34]:
```
array(['OverallQual', 'YearBuilt', 'YearRemodAdd', 'TotalBsmtSF',
       'GrLivArea', 'GarageCars'], dtype=object)
```

After using Lasso feature selection, there are only 5 columns left. We decide to use random forest feature selection because it includes more features.

In [23]:
```
train_new2 = train_new1[selected_feat]
train_new2['LogSalePrice'] = train_new1['LogSalePrice']
```

```
/var/folders/yk/b7cscbxx2wn6l9yslyk7bn0w0000gn/T/ipykernel_23921/2805958101.py:
```

```
2: 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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Modeling

In [24]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(train_new2.loc[:, train_new2
                                                    train_new2['LogSalePrice'],
                                                    test_size = 0.3)
```

In [25]:

```
# train test split - version2
X_train1, X_test1, y_train1, y_test1 = train_test_split(train_new1.loc[:, train_
                                                    train_new1['LogSalePrice'],
                                                    test_size = 0.3)
```

In [26]:

```
# Some definitions prior to training models
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def cross_val(model):
    pred = cross_val_score(model, X, y, cv=10)
    return pred.mean()

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('__________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

In [27]:

```
import matplotlib.pyplot as plt
import numpy as np

import sklearn.datasets
from sklearn import linear_model
alphas = np.logspace(-4, -1, 10)
```

```
scores = np.empty_like(alphas)
for i,a in enumerate(alphas):
    lasso = linear_model.Lasso()
    lasso.set_params(alpha=a)
    lasso.fit(X_train, y_train)
    scores[i] = lasso.score(X_test, y_test)
    print(a, lasso.coef_)
```

```
0.0001 [-1.25092691e-03  9.10625217e-02  4.78221928e-02  2.75676705e-03
  1.58687332e-03 -2.11737956e-05 -6.62664276e-05  1.20423458e-04
 -3.25007059e-06  2.01216846e-04  4.61751615e-02  2.75272677e-06
  7.77618247e-02 -7.71668287e-05  1.26904949e-01  0.00000000e+00
 -6.58104146e-02]
0.00021544346900318845 [-1.24064160e-03  9.09677154e-02  4.78335840e-02  2.76522
259e-03
  1.59285769e-03 -2.12268030e-05 -6.64138849e-05  1.20722427e-04
 -3.47593914e-06  2.01827541e-04  4.60105261e-02  3.54509974e-06
  7.71890134e-02 -7.75071079e-05  1.26150477e-01  0.00000000e+00
 -6.33907151e-02]
0.00046415888336127773 [-1.21848358e-03  9.07634224e-02  4.78581371e-02  2.78343
991e-03
  1.60574984e-03 -2.13389187e-05 -6.67296043e-05  1.21364522e-04
 -3.96272136e-06  2.03143405e-04  4.56558351e-02  5.25227447e-06
  7.59548853e-02 -7.82399859e-05  1.24525051e-01  0.00000000e+00
 -5.81777007e-02]
0.001 [-1.17074684e-03  9.03232364e-02  4.79110501e-02  2.82268849e-03
  1.63352408e-03 -2.15777836e-05 -6.74072707e-05  1.22745276e-04
 -5.01168142e-06  2.05978563e-04  4.48916827e-02  8.93034720e-06
  7.32959803e-02 -7.98186228e-05  1.21023221e-01  0.00000000e+00
 -4.69466919e-02]
0.002154434690031882 [-1.06790453e-03  8.93747512e-02  4.80250900e-02  2.9072488
1e-03
  1.69335898e-03 -2.20851103e-05 -6.88603707e-05  1.25712866e-04
 -7.27229598e-06  2.12087395e-04  4.32453809e-02  1.68547273e-05
  6.75673832e-02 -8.32188359e-05  1.13478870e-01  0.00000000e+00
 -2.27504633e-02]
0.004641588833612777 [-8.36909878e-04  8.74341670e-02  4.64615453e-02  2.9968662
4e-03
  1.79660824e-03 -2.17434936e-05 -7.02685141e-05  1.28287892e-04
 -1.34460259e-05  2.26741060e-04  3.81932921e-02  3.11359574e-05
  5.60605125e-02 -8.09352030e-05  9.69297448e-02  0.00000000e+00
 -0.00000000e+00]
0.01 [-3.23515384e-04  8.34250183e-02  4.00744827e-02  3.03551084e-03
  1.97624412e-03 -1.86226833e-05 -7.04367367e-05  1.27473840e-04
 -2.89202720e-05  2.60800554e-04  2.47976164e-02  5.72467130e-05
  3.26630638e-02 -5.99847588e-05  6.07829235e-02  0.00000000e+00
 -0.00000000e+00]
0.021544346900318822 [ 3.94971288e-04  7.28496768e-02  2.73518389e-02  3.0534114
3e-03
  2.33767128e-03 -1.22184397e-05 -7.11268793e-05  1.30115387e-04
 -5.33481545e-05  3.20533464e-04  0.00000000e+00  9.84638923e-05
  0.00000000e+00 -2.10486677e-05  0.00000000e+00  0.00000000e+00
 -0.00000000e+00]
0.046415888336127774 [ 2.27364098e-04  4.15643146e-02  2.20293621e-03  2.8342559
9e-03
  3.29649827e-03  3.66053139e-08 -6.28868624e-05  1.49203930e-04
 -3.57321033e-05  3.34735177e-04  0.00000000e+00  1.15826413e-04
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00]
0.1 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  3.30499568e-03
```

```
 3.73238769e-03  2.22832598e-05 -3.46028199e-05  1.60518520e-04
-2.22751511e-05  3.67574194e-04  0.00000000e+00  1.28294707e-04
 0.00000000e+00  1.60386439e-05  0.00000000e+00  0.00000000e+00
-0.00000000e+00]
```

In [28]:

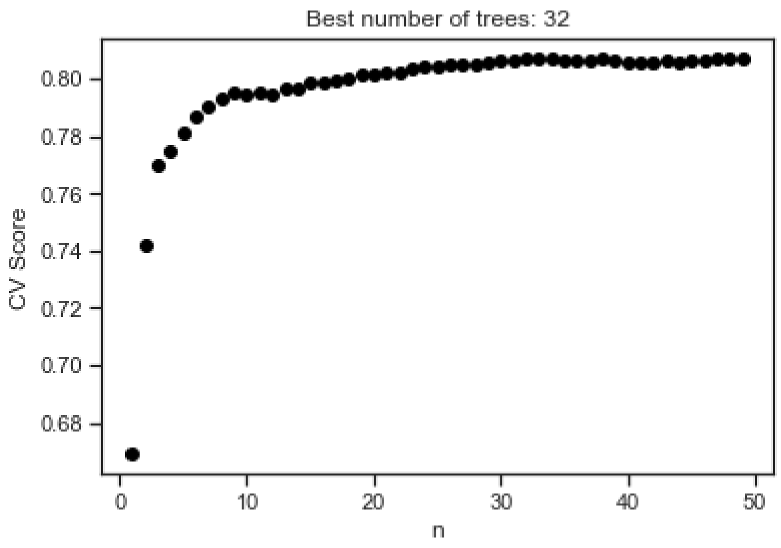
```
from sklearn.linear_model import Lasso

model = Lasso(alpha=0.00046415888336127773,
              selection='random',
              random_state=42)
model.fit(X_train, y_train)

test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Lasso Regression", *evaluate(y_test, test_pr
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square',
```

```
Test set evaluation:
_____________________________________
MAE: 0.0993597322114872
MSE: 0.018667065684144085
RMSE: 0.1366274704594361
R2 Square 0.8878048137059382
__________________________________
====================================
Train set evaluation:
_____________________________________
MAE: 0.09813009463434778
MSE: 0.022934210983218576
RMSE: 0.15144045358892244
R2 Square 0.8532746985020372
__________________________________
```

In [29]:

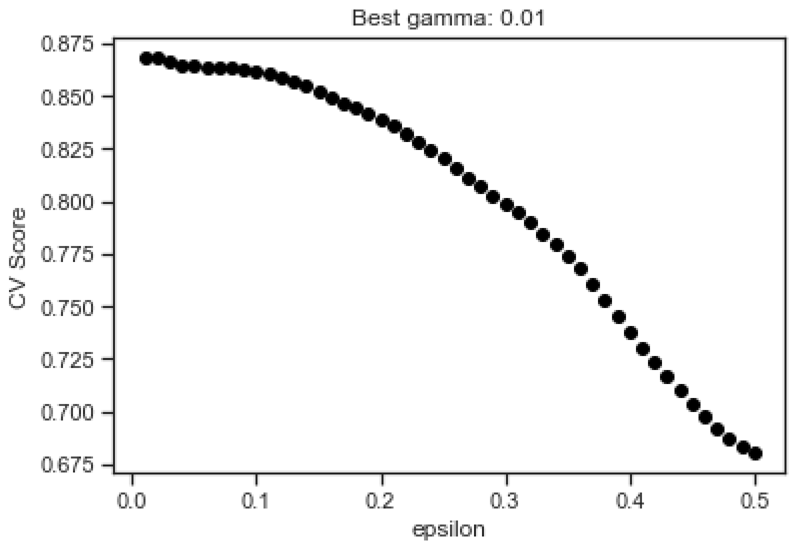
```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

fig, ax = plt.subplots(1)

best_score = 0

#runs through every integer from 1 to 200
for d in range(1,50):
    #create the random forests model with the current value
    RF = RandomForestRegressor(n_estimators=d, max_depth = 5, random_state = 25)

    #check the cross-validation score for this model
    cv_score = cross_val_score(RF, X_train1, y_train1).mean()

    #add this value for d and its cross-validation score to the scatterplot
    ax.scatter(d, cv_score, color = "black")
    #update best_score and best_c if current cross-validation score is new best
    if cv_score > best_score:
        best_n = d
        best_score = cv_score

#label the scatterplot appropriately
l = ax.set(title = "Best number of trees: " + str(best_n),
       xlabel = "n",
       ylabel = "CV Score")

RF = RandomForestRegressor(n_estimators = best_n, max_depth = 5, random_state =
#Fit the model to the training data
RF.fit(X_train1.values, y_train1.values)

#Score the model against the training data and the test data
#to check for accuracy and potential overfitting
#RF.score(X_train.values, y_train.values), RF.score(X_test.values, y_test.values

test_pred = RF.predict(X_test1)
train_pred = RF.predict(X_train1)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test1, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train1, train_pred)

results_df_22 = pd.DataFrame(data=[["Random Forest Model", *evaluate(y_test1, te
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square',
```

```
/Users/hijulia/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:443: Us
erWarning:

X has feature names, but RandomForestRegressor was fitted without feature names

/Users/hijulia/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:443: Us
erWarning:

X has feature names, but RandomForestRegressor was fitted without feature names

Test set evaluation:
_____________________________________
MAE: 0.11686475104404224
MSE: 0.02720384311839455
RMSE: 0.1649358757772079
R2 Square 0.8364959818416875
__________________________________
====================================
Train set evaluation:
_____________________________________
MAE: 0.09339938588948105
MSE: 0.01628379698755422
RMSE: 0.12760798167651669
R2 Square 0.8958217911102863
__________________________________
```



In [30]:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, A
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR


fig, ax = plt.subplots(1)

best_score = 0

#runs from 0.1 to 5 in increments of 0.1
gammas_range = [k/100 for k in range(1, 51)]
for d in gammas_range:
    #create the svc model with the current value
    svr = make_pipeline(StandardScaler(), SVR(epsilon=d))

    #check the cross-validation score for this model
    cv_score = cross_val_score(svr, X_train, y_train).mean()

    #add this value for d and its cross-validation score to the scatterplot
    ax.scatter(d, cv_score, color = "black")

    #update best_score and best_c if current cross-validation score is new best
    if cv_score > best_score:
        best_ep = d
        best_score = cv_score

#label the scatterplot appropriately
l = ax.set(title = "Best gamma: " + str(best_ep),
       xlabel = "epsilon",
       ylabel = "CV Score")
```



In [31]:

```
#SVR --- train_new2
svr = make_pipeline(StandardScaler(), SVR(gamma=best_ep))

#Fit the model to the training data
svr.fit(X_train.values, y_train.values)

#Score the model using the training data and the test data
#to ensure the accuracy of the model and check for potential overfitting
#svr.score(X_train.values, y_train.values), svr.score(X_test.values, y_test.valu
test_pred = svr.predict(X_test)
train_pred = svr.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_222 = pd.DataFrame(data=[["Support Vector Regression Model", *evaluat
                              columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square',
```

```
Test set evaluation:
_____________________________________
MAE: 0.09153899838958994
MSE: 0.017642656656313555
RMSE: 0.13282566264210224
R2 Square 0.8939618478999299
__________________________________
====================================
Train set evaluation:
_____________________________________
MAE: 0.08026416375780822
MSE: 0.013525104039059287
RMSE: 0.11629748079412248
R2 Square 0.9134709727151996
__________________________________
/Users/hijulia/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:443: Us
erWarning:

X has feature names, but StandardScaler was fitted without feature names
```

```
/Users/hijulia/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:443: Us
erWarning:

X has feature names, but StandardScaler was fitted without feature names
```

In [32]:

```
from xgboost import XGBRegressor
XGB = XGBRegressor(random_state=52,max_depth=1,learning_rate=0.1,n_estimators=10
XGB.fit(X_train,y_train)
```

Out[32]:

```
▾                              XGBRegressor

XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=
1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
             grow_policy='depthwise', importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_bin=256,
             max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=
0,
             max_depth=1, max_leaves=0, min_child_weight=5, missing=nan,
             monotone_constraints='()', n_estimators=1000, n_jobs=-1,
```

In [33]:

```
test_pred = XGB.predict(X_test)
train_pred = XGB.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["XGBoost Regression", *evaluate(y_test, test_
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square',
```

```
Test set evaluation:
_____________________________________
MAE: 0.09781401345898895
MSE: 0.019587398915034353
RMSE: 0.13995498888940813
R2 Square 0.8822733091813649
__________________________________
====================================
Train set evaluation:
_____________________________________
MAE: 0.07710990487101813
MSE: 0.012184145870621766
RMSE: 0.11038181856910026
R2 Square 0.9220499681602192
__________________________________
```