Import Libraries and Data

```
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from google.colab import drive
import gdown
gdown.download('https://drive.google.com/uc?id=1OB_GMPQwlIGFpn6KudUTf-Siq85cWU4c',
               'train.csv')
gdown.download('https://drive.google.com/uc?id=1euXRCik-3y-Ip9Z0EO6WIK7gKBFexBAL',
               'test.csv')
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureW
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
Downloading...
From: https://drive.google.com/uc?id=1OB_GMPQwlIGFpn6KudUTf-Siq85cWU4c
To: /content/train.csv
100%|██████████| 461k/461k [00:00<00:00, 51.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=1euXRCik-3y-Ip9Z0EO6WIK7gKBFexBAL
To: /content/test.csv
100%|██████████| 451k/451k [00:00<00:00, 47.4MB/s]
'test.csv'
```

```
# Load datasets

train_df = pd.read_csv("/content/train.csv", delimiter=',')
test_df = pd.read_csv("/content/test.csv", delimiter=',')
```

Cleaning

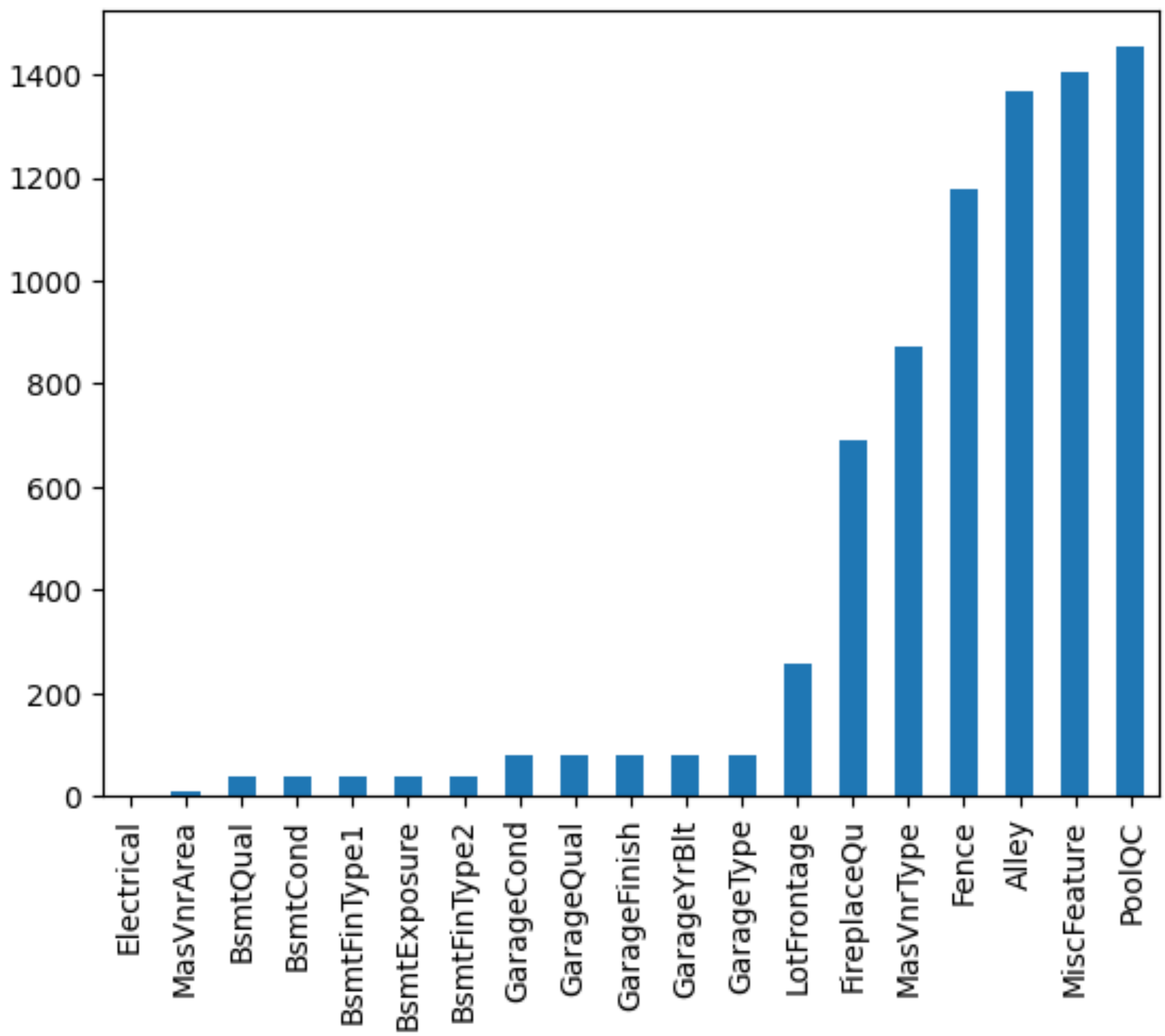
```
# Understand missing values

train_df.describe(include="all")

na_counts = train_df.isnull().sum()

missing = train_df.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

```
<Axes: >
```




```
##############################
# ----Data Cleaning Fcn ---- #
##############################

def data_cleaner(dt):
  dt['SalePrice'] = np.log(dt['SalePrice'])
```

```
dt = dt.drop('Id', axis=1)

### Lot Frontage
dt["LotFrontage"] = dt.groupby(["Street","Neighborhood"])["LotFrontage"].transf
### Alley
dt["Alley"]= dt["Alley"].fillna("None")
### MasVnrType & MasVnrArea
dt["MasVnrArea"] = dt.groupby(["Exterior1st", "Exterior2nd"])["MasVnrArea"].tra
dt["MasVnrType"] = np.where(dt["MasVnrArea"] == 0, "None",dt["MasVnrType"])
dt["MasVnrType"] = dt.groupby(["MasVnrArea", "Exterior1st", "Exterior2nd"])["Ma
### BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
dt[["BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2"]]=dt
### Electrical
dt["Electrical"] = dt["Electrical"].transform(lambda x: x.fillna(x.mode()[0] if
# filling missing fireplace column with None
dt['FireplaceQu'] = dt['FireplaceQu'].fillna('None')

### GarageType, GarageYrBlt, GarageFinish, GarageQual, GarageCond
dt[["GarageType", "GarageYrBlt", "GarageFinish", "GarageQual", "GarageCond"]] =
### PoolQC
dt["PoolQC"] = dt["PoolQC"].fillna("None")
### Fence
dt["Fence"] = dt["Fence"].fillna("None")
### MiscFeature
dt["MiscFeature"] = dt["MiscFeature"].fillna("None")

## Convert Time variables to Integer Age

dt["Age"] = 2010 - dt["YearBuilt"]
dt = dt.drop(columns=['YearBuilt'])
#dt['Age'] = dt.fillna(0)

dt["YearsSinceRemodeled"] = 2010 - dt["YearRemodAdd"]
dt = dt.drop(columns=['YearRemodAdd'])
#dt["YearsSinceRemodeled"] = dt["YearsSinceRemodeled"].fillna(0)

dt["GarageYrBlt"] = pd.to_numeric(dt["GarageYrBlt"], errors='coerce')
dt["GarageAge"] = 2010 - dt["GarageYrBlt"]
dt = dt.drop(columns=['GarageYrBlt'])
dt['GarageAge'] = dt['GarageAge'].fillna(0)


dt["TimeSinceSold"] = 2010 - dt["YrSold"] + ((12 - dt["MoSold"]) / 12)
#dt['TimeSinceSold'] = dt['TimeSinceSold'].fillna(0)
dt = dt.drop(columns=['YrSold'])
dt = dt.drop(columns=['MoSold'])
```

```
  # Create Dummies

  encoded_dt = pd.get_dummies(dt, columns=dt.select_dtypes(include=['object', 'ca

  return encoded_dt

########################
########################


# Checking the fcn

cleaned_train = data_cleaner(train_df)


cleaned_train
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	Bs
0	60	65.0	8450	7	5	196.0	
1	20	80.0	9600	6	8	0.0	
2	60	68.0	11250	7	5	162.0	
3	70	60.0	9550	7	5	0.0	
4	60	84.0	14260	8	5	350.0	
...	...	...	...	...	...	...	
1455	60	62.0	7917	6	5	0.0	
1456	20	85.0	13175	6	6	119.0	
1457	70	66.0	9042	7	9	0.0	
1458	20	68.0	9717	5	6	0.0	
1459	20	75.0	9937	5	6	0.0	

1460 rows × 302 columns

Testing Feature Selection Algorithms

In this analysis, we will evaluate the effectiveness of various feature selection algorithms in two distinct scenarios based on the relationship between the number of rows (n) and columns (m) in our dataset. Specifically, we will assess:

1. **Evaluating Feature Selection Algorithms when (n > m)**
2. **Evaluating Feature Selection Algorithms when (m > n)**

Where:

- (n) is the number of rows (observations).
- (m) is the number of columns (features).

Objective

The primary goal is to evaluate the **Root Mean Squared Error (RMSE)** of a standard linear regression model across three different feature selection algorithms, utilizing **5-fold cross-validation** and **Booststrapping** to ensure robust results.

Feature Selection Algorithms Under Evaluation

1. **Select K Best**
   - Uses mutual information regression to select the top k features based on their individual performance in predicting the target variable.
2. **Variance Threshold**
   - Eliminates features with low variance, retaining only those that contribute meaningful variability to the dataset.
3. **Recursive Feature Elimination (RFE) with Lasso Regression**
   - This method recursively removes features and builds a model on the remaining attributes, leveraging Lasso regression to promote sparsity in feature selection.

Methodology

To create a scenario where (m > n), we will utilize a random number generator to obtain a consistent set of random index numbers, with a length of (m - 25). This method ensures that the random indices remain constant across each feature selection algorithm during the experiment, thus minimizing variance and bias in the results. For the scenerio where (n > m), we utilize the entire provided training dataset. We use bootstrapping in a few ways:

1. To calculate robust RMSE values and confidence intervals for the baseline model.

2. To calculate robust RMSE values and confidence intervals for the full data partition.
3. To calculate robust RMSE and Percent Features in Common measures and confidence intervals for the subset data partition - not necessarily because of variability in cross-validation fold randomization, but because of the variation in features selected from the high-dimensionality scenerio.

```
############################################################
# ---------- Testing Feature Selection Algorithms ----------
############################################################

from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import Normalizer
from sklearn.feature_selection import SelectKBest, RFE, VarianceThreshold, Select
from sklearn.pipeline import Pipeline
from sklearn.metrics import root_mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_predict, KFold


feature_selection_algos = [
    ('SelectKBest', SelectKBest(mutual_info_regression,
                                k = 25)), # mutual_info_regression, r_regression,
    ('RFE', RFE(estimator = Lasso(),
                n_features_to_select = 25,
                step = 1)),
    ('VarianceThreshold', VarianceThreshold(threshold = 1.9)), # about 25 feature
    ('SelectFromModel_Lasso', SelectFromModel(estimator = Lasso(),
                                        max_features = 25)),
     ('SelectFromModel_Tree', SelectFromModel(estimator = DecisionTreeRegressor()
                                        max_features = 25))
]

model = LinearRegression()
cv = KFold(n_splits=5, shuffle=True)

y, X = cleaned_train['SalePrice'], cleaned_train.drop('SalePrice', axis=1)



######################
## Running All Tests
######################

# BOOTSTRAP CONTROL:
n_bootstrap = 100
```

```
####################
# BASELINE MODEL

lasso_rmse = []
print("Running control model")
for iter in range(0, n_bootstrap):
  lasso_pred = lasso_pred = cross_val_predict(Lasso(), X, y, cv=cv)
  lasso_rmse.append(np.sqrt(np.mean((np.exp(y) - np.exp(lasso_pred)) ** 2)))


lasso_cv_rmse = np.mean(lasso_rmse)

# Bootstraping Begins
bootstrap_rmse_results = {name: [] for name, _ in feature_selection_algos}
bootstrap_common_feature_percentages = {name: [] for name, _ in feature_selection_

####################
# FULL PARTITION
full_rmse_results = {}
full_rmse_ci = {}
full_rmse_std = {}
selected_features_full = {}

for name, selector in feature_selection_algos:
    print(f'Running "full" partition for {name}')
    selector.fit(X, y)
    X_full_transformed = selector.transform(X)

    # Store selected features for 'full'
    if hasattr(selector, 'get_support'):
        selected_features_full[name] = X.columns[selector.get_support()]
    elif hasattr(selector, 'get_feature_names_out'):
        selected_features_full[name] = selector.get_feature_names_out(input_featu
    else:
        selected_features_full[name] = []

    temp_rmse_results = []

    for iter in range(0, n_bootstrap):

      y_pred = cross_val_predict(model, X_full_transformed, y, cv=cv)
      temp_rmse_results.append(np.sqrt(np.mean((np.exp(y) - np.exp(y_pred)) ** 2)

    full_rmse_results[name] = np.mean(temp_rmse_results)
    full_rmse_std[name] = np.std(temp_rmse_results)
    ci_low, ci_high = np.percentile(temp_rmse_results, [2.5, 97.5])
    full_rmse_ci[name] = (ci_low, ci_high)
```

```
####################
# SUBSET PARTITION
for bootstrap_iteration in range(n_bootstrap):
    print(f'Running bootstrap sample {bootstrap_iteration + 1}/{n_bootstrap}')

    # Create the random subset for the bootstrap sample
    subset_indices = np.random.randint(0, len(cleaned_train), size=len(cleaned_tr

    for name, selector in feature_selection_algos:
        # Fit selector to the 'subset'
        X_subset = X.iloc[subset_indices]
        y_subset = y.iloc[subset_indices]
        selector.fit(X_subset, y_subset)
        X_subset_transformed = selector.transform(X)

        # Calculate RMSE for the 'subset'

        subset_y_pred = cross_val_predict(model, X_subset_transformed, y, cv=cv)
        bootstrap_rmse_results[name].append(np.sqrt(np.mean((np.exp(y) - np.exp(s


        # Percentage of feature commonality
        common_feat_list = [
            feature for feature in selected_features_full[name]
            if feature in X.columns[selector.get_support()]
        ]
        common_percentage = len(common_feat_list) / len(selected_features_full[na

        # Store the common feature percentage
        bootstrap_common_feature_percentages[name].append(common_percentage)

# Calculate confidence intervals for percentage of feature commonality for each a
for name in bootstrap_rmse_results:
    common_feat_array = np.array(bootstrap_common_feature_percentages[name])

    # Common feature percentage confidence intervals
    common_ci_low, common_ci_high = np.percentile(common_feat_array, [2.5, 97.5])
```

```
Running control model
Running "full" partition for SelectKBest
Running "full" partition for RFE
Running "full" partition for VarianceThreshold
Running "full" partition for SelectFromModel_Lasso
Running "full" partition for SelectFromModel_Tree
Running bootstrap sample 1/100
Running bootstrap sample 2/100
Running bootstrap sample 3/100
Running bootstrap sample 4/100
```

```
    Running bootstrap sample 5/100
    Running bootstrap sample 6/100
    Running bootstrap sample 7/100
    Running bootstrap sample 8/100
    Running bootstrap sample 9/100
    Running bootstrap sample 10/100
    Running bootstrap sample 11/100
    Running bootstrap sample 12/100
    Running bootstrap sample 13/100
    Running bootstrap sample 14/100
    Running bootstrap sample 15/100
    Running bootstrap sample 16/100
    Running bootstrap sample 17/100
    Running bootstrap sample 18/100
    Running bootstrap sample 19/100
    Running bootstrap sample 20/100
    Running bootstrap sample 21/100
    Running bootstrap sample 22/100
    Running bootstrap sample 23/100
    Running bootstrap sample 24/100
    Running bootstrap sample 25/100
    Running bootstrap sample 26/100
    Running bootstrap sample 27/100
    Running bootstrap sample 28/100
    Running bootstrap sample 29/100
    Running bootstrap sample 30/100
    Running bootstrap sample 31/100
    Running bootstrap sample 32/100
    Running bootstrap sample 33/100
    Running bootstrap sample 34/100
    Running bootstrap sample 35/100
    Running bootstrap sample 36/100
    Running bootstrap sample 37/100
    Running bootstrap sample 38/100
    Running bootstrap sample 39/100
    Running bootstrap sample 40/100
    Running bootstrap sample 41/100
    Running bootstrap sample 42/100
    Running bootstrap sample 43/100
    Running bootstrap sample 44/100
    Running bootstrap sample 45/100
    Running bootstrap sample 46/100
    Running bootstrap sample 47/100
    Running bootstrap sample 48/100
    Running bootstrap sample 49/100
    Running bootstrap sample 50/100
    Running bootstrap sample 51/100
    Running bootstrap sample 52/100
    Running bootstrap sample 53/100
    Running bootstrap sample 54/100
```

```
# Create a dictionary to store the formatted RMSE results
formatted_rmse_results = {
    'Full': [],
```

```
    'Full 95% CI Lower': [],
    'Full 95% CI Upper': [],
    'Subset': [],
    'Subset 95% CI Lower': [],
    'Subset 95% CI Upper': []
}


for name in bootstrap_rmse_results:
    # Get the full RMSE value
    full_rmse = full_rmse_results[name]
    # Confidence Int for full subset
    full_rmse_ci_low, full_rmse_ci_high = full_rmse_ci[name]

    # Calculate the mean and confidence interval for the subset RMSE
    subset_rmse_array = np.array(bootstrap_rmse_results[name])
    subset_rmse_mean = subset_rmse_array.mean()
    subset_rmse_ci_low, subset_rmse_ci_high = np.percentile(subset_rmse_array, [2

    formatted_rmse_results['Full'].append(full_rmse)
    formatted_rmse_results['Full 95% CI Lower'].append(full_rmse_ci_low)
    formatted_rmse_results['Full 95% CI Upper'].append(full_rmse_ci_high)
    formatted_rmse_results['Subset'].append(subset_rmse_mean)
    formatted_rmse_results['Subset 95% CI Lower'].append(subset_rmse_ci_low)
    formatted_rmse_results['Subset 95% CI Upper'].append(subset_rmse_ci_high)

rmse_df = pd.DataFrame(formatted_rmse_results, index=[name for name, _ in feature

rmse_df = np.round(rmse_df, 2)

control_lasso_rmse = np.round(lasso_cv_rmse, 2)
lasso_rmse_ci_low, lasso_rmse_ci_high = np.percentile(lasso_rmse, [2.5, 97.5])

# Append the Control Lasso row to the DataFrame
control_row = pd.DataFrame({
    'Full': [control_lasso_rmse],
    'Full 95% CI Lower':[lasso_rmse_ci_low],
    'Full 95% CI Upper': [lasso_rmse_ci_high],
    'Subset': [' '],
    'Subset 95% CI Lower': [' '],
    'Subset 95% CI Upper': [' ']
}, index=['Control Lasso'])

# Combine the original DataFrame with the control row
rmse_df = pd.concat([rmse_df, control_row])
```

```
rmse_df
```

	Full	Full 95% CI Lower	Full 95% CI Upper	Subset	Subset 95% CI Lower	
SelectKBest	69838.90	62898.870000	84738.730000	75437.46	63035.86	
RFE	107954.10	85511.670000	134999.420000	302574.49	77774.09	2[illegible]
VarianceThreshold	83253.75	72062.080000	100890.790000	97356.36	69991.35	[illegible]
SelectFromModel_Lasso	116037.21	106580.530000	136815.180000	118236.3	82876.55	[illegible]
SelectFromModel_Tree	90812.73	80617.430000	109767.010000	81865.78	62075.79	[illegible]

```
# Standard Deviation between Full and Subset RMSE Values
rmse_stdevs = {
    'Full': [],
    'Subset': [],
}

for name in bootstrap_rmse_results:
    # FULL RMSE STD
    full_std = full_rmse_std[name]

    # SUBSET RMSE STD
    subset_rmse_std = np.std(bootstrap_rmse_results[name])

    rmse_stdevs['Full'].append(full_std)
    rmse_stdevs['Subset'].append(subset_rmse_std)

rmse_std_df = pd.DataFrame(rmse_stdevs, index=[name for name, _ in feature_select

np.round(rmse_std_df, 2)
```

	Full	Subset
SelectKBest	6416.26	8096.52
RFE	12330.15	1156413.07
VarianceThreshold	8885.44	20898.27
SelectFromModel_Lasso	8535.57	16537.02
SelectFromModel_Tree	9275.00	12992.92

```
formatted_percentage_results = {
    'Percentage': [],
    'Percentage 95% CI Lower': [],
    'Percentage 95% CI Upper': []
}

# Iterate over each algorithm to fill in the DataFrame
for name in bootstrap_common_feature_percentages:
    # Get the array of common feature percentages
    percentage_array = np.array(bootstrap_common_feature_percentages[name])

    # Calculate the mean and confidence interval for the percentages
    percentage_mean = percentage_array.mean()
    percentage_ci_low, percentage_ci_high = np.percentile(percentage_array, [2.5,

    # Add the values to the dictionary
    formatted_percentage_results['Percentage'].append(percentage_mean)
    formatted_percentage_results['Percentage 95% CI Lower'].append(percentage_ci_
    formatted_percentage_results['Percentage 95% CI Upper'].append(percentage_ci_

percentage_df = pd.DataFrame(formatted_percentage_results, index=[name for name,
```

```
np.round(percentage_df, 3)
```

	Percentage	Percentage 95% CI Lower	Percentage 95% CI Upper
SelectKBest	0.862	0.80	0.96
RFE	0.890	0.72	0.96
VarianceThreshold	0.970	0.92	1.00
SelectFromModel_Lasso	0.854	0.70	1.00
SelectFromModel_Tree	0.561	0.45	0.75

```
# What are the common features across full subset?

select_algorithms = ['SelectKBest', 'RFE', 'VarianceThreshold', 'SelectFromModel_


# FULL
common_features_full = set(selected_features_full[select_algorithms[0]])

for algo in select_algorithms[1:]:

  features_algo = set(selected_features_full[algo])

  common_features_full &= features_algo

common_feat_list_full = list(common_features_full)


from scipy.stats import pearsonr

unsure_features = []

for feature in common_feat_list_full:

    coef, p_value = pearsonr(cleaned_train['SalePrice'], cleaned_train[feature])

    if coef > 0.1 and p_value < 0.05:
        print(f"{feature} increases house prices [corr_coef: {coef:.2f}, p-value:
    elif coef < -0.1 and p_value < 0.05:
        print(f"{feature} decreases house prices [corr_coef: {coef:.2f}, p-value:
    else:
        unsure_features.append(feature)
        print(f"**Unsure about {feature}.** Correlation coefficient: {coef:.2f},
```

```
YearsSinceRemodeled decreases house prices [corr_coef: -0.57, p-value: 0.00].

TotalBsmtSF increases house prices [corr_coef: 0.61, p-value: 0.00].

GrLivArea increases house prices [corr_coef: 0.70, p-value: 0.00].

GarageArea increases house prices [corr_coef: 0.65, p-value: 0.00].

Age decreases house prices [corr_coef: -0.59, p-value: 0.00].

2ndFlrSF increases house prices [corr_coef: 0.32, p-value: 0.00].
```

```
# Lasso Comparison
from sklearn.linear_model import LassoCV, Lasso
```

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

alpha = 0.01

lasso = Lasso(alpha=alpha).fit(X_scaled, y)

# Count non-zero coefficients
non_zero_count = np.sum(lasso.coef_ != 0)
print(f'Number of non-zero coefficients: {non_zero_count}')

# If not within the desired range, adjust alpha
while non_zero_count < 10 or non_zero_count > 15:
    if non_zero_count > 15:
        alpha *= 1.1  # Increase alpha to increase regularization (fewer features
    elif non_zero_count < 10:
        alpha /= 1.1  # Decrease alpha to decrease regularization (more features)

    lasso = Lasso(alpha=alpha).fit(X_scaled, y)
    non_zero_count = np.sum(lasso.coef_ != 0)
    print(f'Adjusted alpha: {alpha}, Non-zero coefficients: {non_zero_count}')

selected_features = np.array(X.columns)[lasso.coef_ != 0]
selected_coefficients = lasso.coef_[lasso.coef_ != 0]

# Create a DataFrame to display features and coefficients
ls_mod_features = pd.DataFrame({
    'Feature': selected_features,
    'Coefficient': selected_coefficients
}).sort_values(by='Coefficient', ascending=False).reset_index(drop=True)
```

```
Number of non-zero coefficients: 57
Adjusted alpha: 0.011000000000000001, Non-zero coefficients: 53
Adjusted alpha: 0.012100000000000001, Non-zero coefficients: 51
Adjusted alpha: 0.013310000000000002, Non-zero coefficients: 48
Adjusted alpha: 0.014641000000000003, Non-zero coefficients: 44
Adjusted alpha: 0.016105100000000004, Non-zero coefficients: 39
Adjusted alpha: 0.017715610000000007, Non-zero coefficients: 36
Adjusted alpha: 0.019487171000000008, Non-zero coefficients: 31
Adjusted alpha: 0.021435888100000012, Non-zero coefficients: 30
Adjusted alpha: 0.023579476910000015, Non-zero coefficients: 28
Adjusted alpha: 0.025937424601000018, Non-zero coefficients: 25
Adjusted alpha: 0.02853116706110002, Non-zero coefficients: 23
Adjusted alpha: 0.031384283767210024, Non-zero coefficients: 20
Adjusted alpha: 0.03452271214393103, Non-zero coefficients: 21
Adjusted alpha: 0.03797498335832414, Non-zero coefficients: 18
Adjusted alpha: 0.04177248169415655, Non-zero coefficients: 17
Adjusted alpha: 0.04594972986357221, Non-zero coefficients: 15
```

```
np.round(ls_mod_features, 4).set_index('Feature')
```

	Coefficient
Feature	
OverallQual	0.1404
GrLivArea	0.0888
GarageCars	0.0504
TotalBsmtSF	0.0267
Fireplaces	0.0111
BsmtFinSF1	0.0063
1stFlrSF	0.0034
MSZoning_RL	0.0030
GarageArea	0.0029
CentralAir_Y	0.0000
FireplaceQu_None	-0.0039
CentralAir_N	-0.0062
MSZoning_RM	-0.0115
Age	-0.0234
YearsSinceRemodeled	-0.0257