Module 3 Assignment 1: Advanced Regression Techniques

Aaron Chen, Nikhil Prabhu, Elvis Matos, Reese Mayer

Introduction:

In our project, we utilized two datasets: the primary train dataset and the secondary test dataset to explore multiple regression models for effective prediction. We started with comprehensive exploratory data analysis and cross-validation to enhance understanding and ensure model reliability. Then, we constructed Lasso, Ridge, and ElasticNet regression models. Through iterative refinement and expanded data exploration, we developed functional predictive models, surpassing our previous project's scope.

Cross Validation:

The cross-validation helped to ensure that the model predictions were accurate and reliable, taking into account the variability in the data and avoiding overfitting. Initially, the dataset was divided into k folds, where each fold served as a testing set, while the remaining folds were used for training. Once the model was trained on the training data and evaluated it resulted in a mean squared error of 1513951629, which is considered particularly high. A high mean squared error could occur due to various factors such as inadequate feature selection, non-linear relationships between features and target variables, or outliers influencing the model performance. Moreover, the chosen model may be too simplistic or overly complex for the underlying data structure. In the ElasticNet model, for example, despite utilizing cross-validation and incorporating 19 variables, the resulting mean square error (MSE) of 1410194382 suggests that there might have been inadequate feature selection.

EDA:

Within the Exploratory Data Analysis, the first graph that was created was a scatter plot where we compared the variables Sales Price and GrLiveArea. A bar chart shows the year when the house was built where it was compared to the sales price and it can be seen that around the year 1990 had the highest spike in Sales Price which was evaluated at around $700,000. Furthermore, a histogram was conducted to compare the overall quality. When looking at the data it can be seen that the values range from 5 through 7. A heatmap was then created to see all the columns and their correlation. A top correlation variable was created to see what columns correlated greater than 0.2, leading to another heat map where the correlation was compared to the Sales Price. This heat map helped to pick the columns for the next steps.

Models:

In this section, the columns from the top correlated variables were chosen and placed into a New_train variable. The first model used was the linear regression model; it can be seen that the values have a mix of negative and positive coefficients. The negative coefficients represent that it doesn't have a great relationship to the target variable which could reduce the predictive model and vice versa for the positive coefficient. The polynomial model was used where this relationship can help capture more complex relationships between the features and target variable. Through feature engineering techniques we were able to ensure our data integrity, by addressing the missing values by filling all the NA or null values with zeros. Additionally, we engineered new features such as 'TotalSF' to represent the total square footage by summing different square footage measurements that were split up. With these transformations, we were able to improve the completeness and accuracy of our datasets and also provide insights that we believe enhanced our predictive models.

Regression Models:

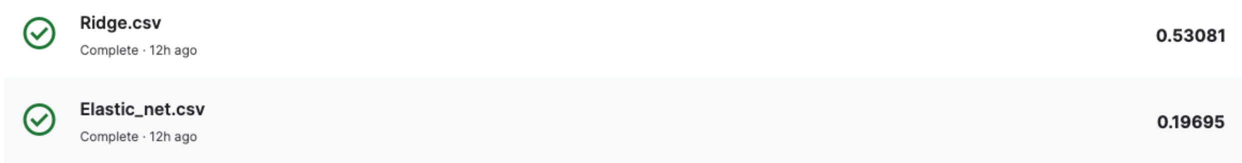
We created three different models using Lasso, Ridge, and ElasticNet techniques. The Lasso regression model is created and trained on the training data along with the other two regression techniques. The mean squared error we achieved for the Lasso model was very high indicating the average squared difference between the predicted and actual home prices may have not been very accurate. Our results for the ElasticNet and Ridge regressions both returned high mse as well indicating that we may have over-complicated our models in the process due to the possibility of too many variables to create accurate predictions in a possible nonlinear relationship.

Hyperparameter Tuning:

Hyperparameter tuning for an ElasticNet model is implemented using search with cross-validation. The dataset is split into training and testing sets, and an ElasticNet model is defined. Hyperparameters such as alpha and l1_ratio are specified for tuning. Grid search is performed with 5-fold cross-validation to find the best combination of hyperparameters. The optimal alpha and l1_ratio values are determined to be 0.1 and 0.5, respectively. Subsequently, the Elastic model is initialized with these best hyperparameters and trained on the training set. Predictions are made on the test set using the best model, resulting in a mean squared error of 1410194382. This approach showcases the optimization of the ElasticNet Model's hyperparameter, although the obtained mean squared error suggests potential issues with model performance that may warrant further investigation or refinement.

Appendix:


Nikhilprabhu1999


Ridge.csv
Complete · 12h ago
0.53081
Elastic_net.csv
Complete · 12h ago
0.19695

Module 3 Assignment 1 : House Prices: Advanced Regression Techniques

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_fac
from sklearn.metrics import r2_score, mean_absolute_error, mean_squarec
import scipy.stats as stats
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

from sklearn.linear_model import Lasso, Ridge, ElasticNet
from sklearn.preprocessing import StandardScaler
```

Research Question

Our goal is to establish and research the variables of house square footage and lot size that will impact the final price of the home.

In [2]:
```
# Data
train = pd.read_csv('train.csv')
test= pd.read_csv('test.csv')

# replace NA to 0
train.fillna(0, inplace=True)
test.fillna(0, inplace=True)
```

In [3]:
```
X_train = train.drop('SalePrice', axis=1)  # Features for training
y_train = train['SalePrice']               # Target variable for traini
X_test = test    # Features for testing (assuming no target variable in
```

In [4]:
```
X_train = X_train.select_dtypes(include=[np.number])
X_train = X_train.astype(int)
```

1.

Conduct your analysis using a cross-validation design.

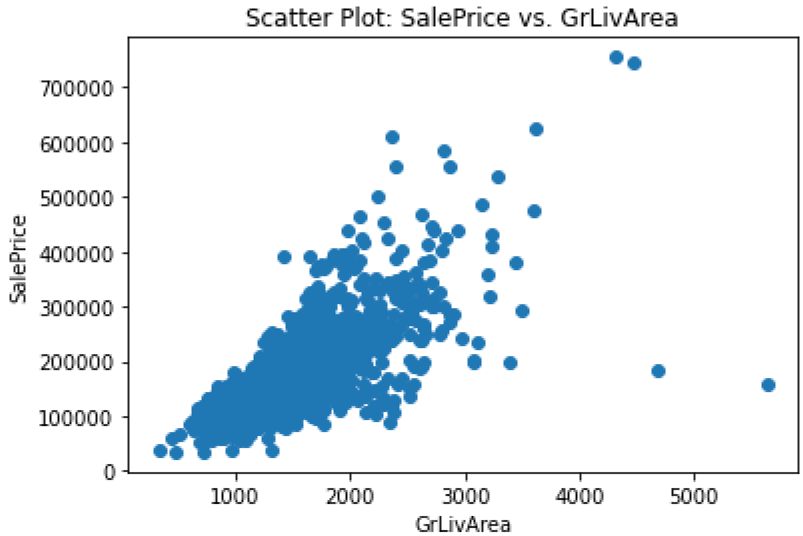
In [5]:
```
# Perform Cross-Validation
# Initialize Linear Regression model
linear_model = LinearRegression()

# Initialize k-fold cross-validator
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Initialize list to store MSE scores
mse_scores = []

# Perform k-fold cross-validation
for train_index, val_index in kfold.split(X_train):
    # Split data into training and validation sets for current fold
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc

    # Fit the linear regression model on the training data for the curr
    linear_model.fit(X_train_fold, y_train_fold)

    # Make predictions on the validation data for the current fold
    y_val_pred = linear_model.predict(X_val_fold)

    # Calculate the Mean Squared Error for the current fold
    mse_fold = mean_squared_error(y_val_fold, y_val_pred)

    # Append the MSE to the list of MSE scores
    mse_scores.append(mse_fold)

# Calculate the average MSE across all folds
avg_mse = np.mean(mse_scores)
print("Average Mean Squared Error:", avg_mse)
```

```
Average Mean Squared Error: 1513951629.5258148
```

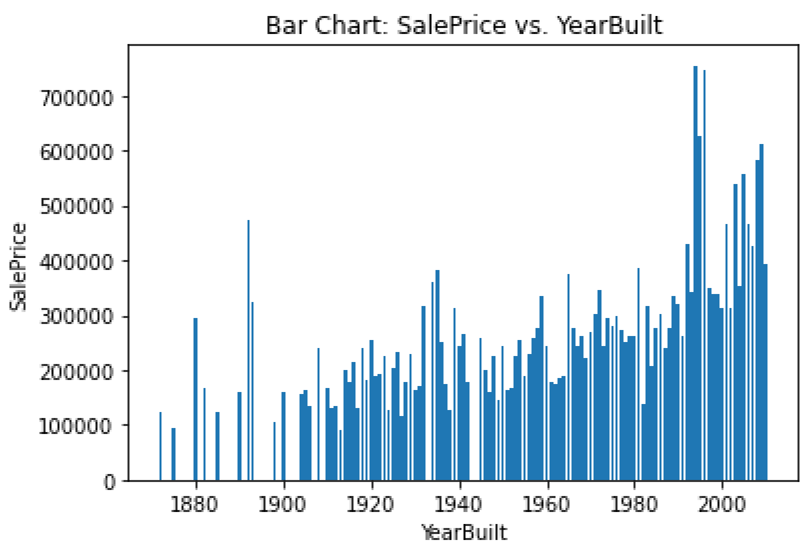
2.

Conduct / improve upon previous EDA.

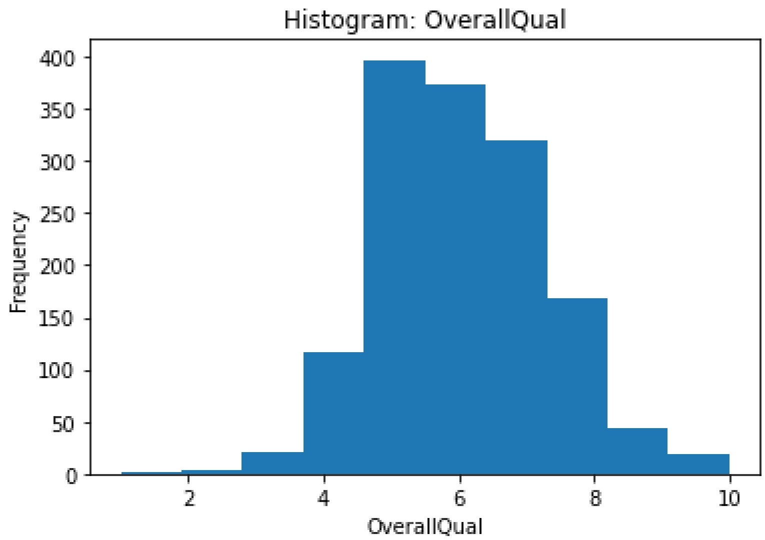
In [6]:
```
# Scatterplot
plt.scatter(X_train['GrLivArea'], y_train)
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.title('Scatter Plot: SalePrice vs. GrLivArea')
plt.show()
```



In [7]:
```
# Bar chart of YearBuilt vs. SalePrice
plt.bar(X_train['YearBuilt'],  y_train)
plt.xlabel('YearBuilt')
plt.ylabel('SalePrice')
plt.title('Bar Chart: SalePrice vs. YearBuilt')
plt.show()
```

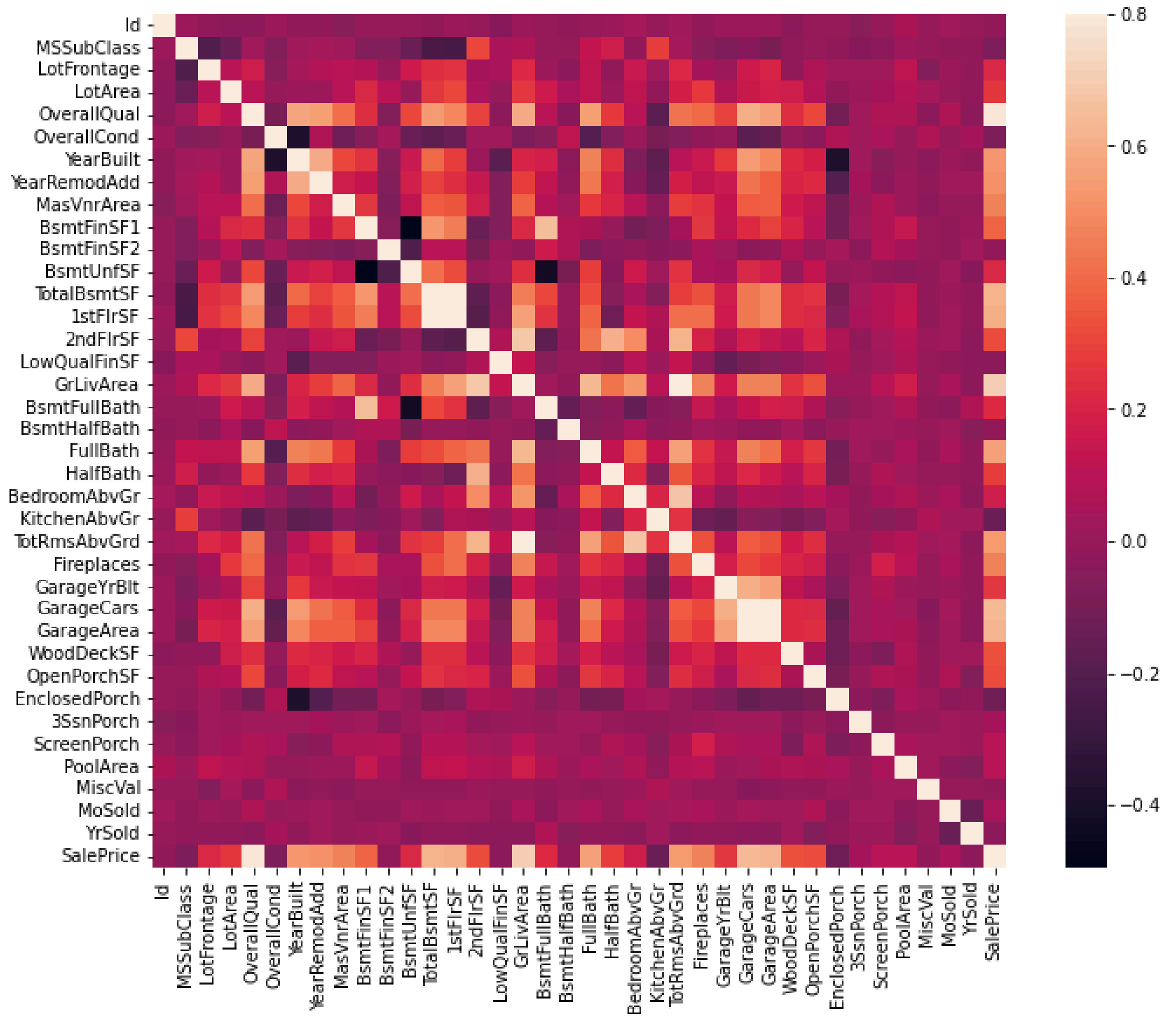


In [8]:
```
# Histogram of OverallQual
plt.hist(X_train['OverallQual'])
plt.xlabel('OverallQual')
plt.ylabel('Frequency')
plt.title('Histogram: OverallQual')
plt.show()
```



In [9]:
```python
# variable correlation Heatmap

corr_mat = train.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr_mat, vmax=0.8, square=True)
```

Out[9]: <AxesSubplot:>



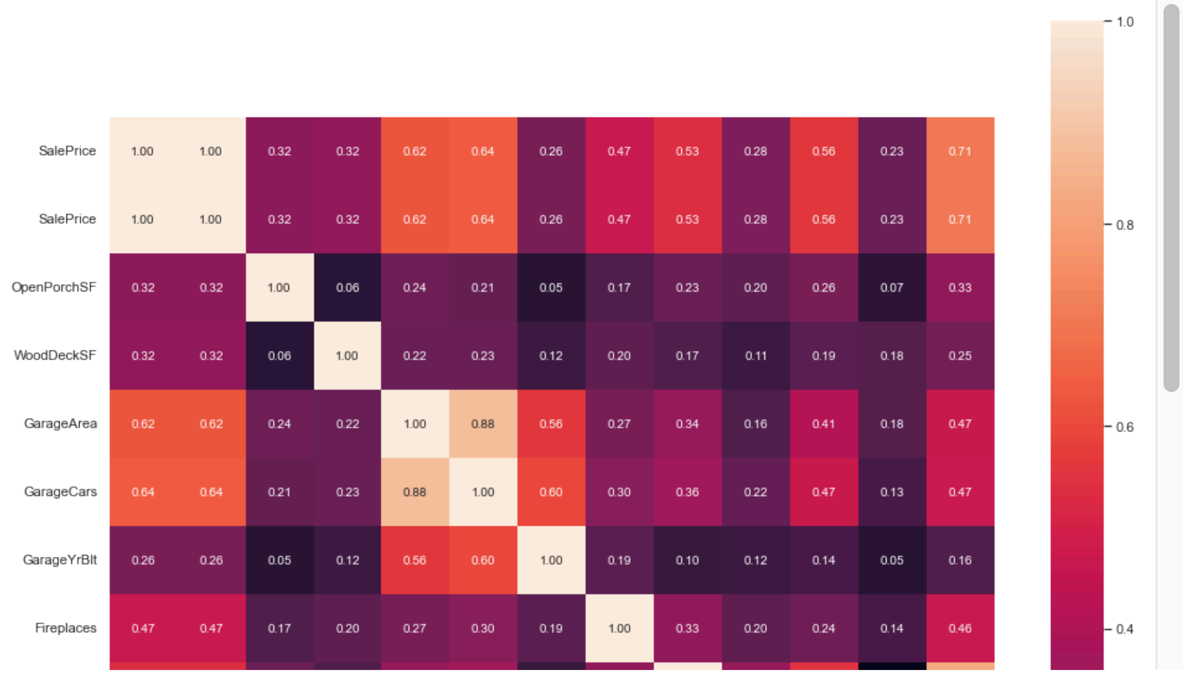
In [10]:
```python
# top correlated variable with saleprice

top_correlations = train.corr()
top_feature_columns = top_correlations['SalePrice'][top_correlations['S
top_feature_columns
```

Out[10]:
```
array(['LotFrontage', 'LotArea', 'OverallQual', 'YearBuilt',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullB
ath',
       'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
       'OpenPorchSF', 'SalePrice'], dtype=object)
```

In [11]:
```
len(top_feature_columns)
```

Out[11]: 23

In [12]:
```
# Heatmap with top correlated variable

heat_map_with_top_correlated_features = np.append(top_feature_columns[-
pearson_correlation_coefficients = np.corrcoef(train[heat_map_with_top_
plt.figure(figsize=(16,16))
sns.set(font_scale=1)
with sns.axes_style('white'):
    sns.heatmap(pearson_correlation_coefficients, yticklabels=heat_map_
                xticklabels=heat_map_with_top_correlated_features[::-1]
                annot=True, square=True, cmap=None)
```



3.

Build models with many variables.

In [13]:
```
New_train = train[['Id','LotFrontage', 'LotArea', 'OverallQual', 'YearE
        'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnfSF',
        'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBat
        'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces',
        'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF',
        'OpenPorchSF', 'SalePrice']]

X_New_train = New_train.drop('SalePrice', axis=1)  # Features for trair
y_New_train = New_train['SalePrice']
```

In [14]:
```
# Linear Regressin model

linear_regressor = LinearRegression()
linear_regressor.fit(X_New_train, y_New_train)

print("Coefficients:", linear_regressor.coef_)
print("Intercept:", linear_regressor.intercept_)
```

```
Coefficients: [-1.16424828e+00  2.99798694e+01  4.56563344e-01  1.959
93654e+04
  1.78476292e+02  3.27472893e+02  2.83261353e+01  7.70527521e+00
 -5.53797552e+00  1.48311533e+01  3.26490915e+01  2.73611910e+01
  1.13606219e+01  4.98280819e+03 -2.60124403e+03 -1.14317448e+03
  1.45310773e+03  7.14176977e+03 -1.09979732e+01  1.37121712e+04
  1.37463288e+01  2.75306976e+01  6.08767394e+00]
Intercept: -1054183.5340628612
```

In [15]:
```
# Polynomial model

degree = 2
poly_features = PolynomialFeatures(degree=degree)
X_poly_train = poly_features.fit_transform(X_New_train)
X_poly_train
```

Out[15]:
```
array([[1.00000e+00, 1.00000e+00, 6.50000e+01, ..., 0.00000e+00,
        0.00000e+00, 3.72100e+03],
       [1.00000e+00, 2.00000e+00, 8.00000e+01, ..., 8.88040e+04,
        0.00000e+00, 0.00000e+00],
       [1.00000e+00, 3.00000e+00, 6.80000e+01, ..., 0.00000e+00,
        0.00000e+00, 1.76400e+03],
       ...,
       [1.00000e+00, 1.45800e+03, 6.60000e+01, ..., 0.00000e+00,
        0.00000e+00, 3.60000e+03],
       [1.00000e+00, 1.45900e+03, 6.80000e+01, ..., 1.33956e+05,
        0.00000e+00, 0.00000e+00],
       [1.00000e+00, 1.46000e+03, 7.50000e+01, ..., 5.41696e+05,
        5.00480e+04, 4.62400e+03]])
```

In [16]:
```
categorical_cols = train[['MSZoning', 'BldgType']]

# Dichotomous Variables

dichotomous_var = pd.get_dummies(categorical_cols)
dichotomous_var
```

Out[16]:

	MSZoning_C (all)	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	BldgType_1Fam
0	0	0	0	1	0	1
1	0	0	0	1	0	1
2	0	0	0	1	0	1
3	0	0	0	1	0	1
4	0	0	0	1	0	1
...	...	...	...	...	...	...
1455	0	0	0	1	0	1
1456	0	0	0	1	0	1
1457	0	0	0	1	0	1
1458	0	0	0	1	0	1
1459	0	0	0	1	0	1

1460 rows × 10 columns

4.

Transform and feature engineer as appropriate.

In [20]:
```
New_train
```

Out[20]:

	Id	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
0	1	65.0	8450	7	2003	2003	196.0	
1	2	80.0	9600	6	1976	1976	0.0	
2	3	68.0	11250	7	2001	2002	162.0	
3	4	60.0	9550	7	1915	1970	0.0	
4	5	84.0	14260	8	2000	2000	350.0	
...	...	...	...	...	...	...	...	
1455	1456	62.0	7917	6	1999	2000	0.0	
1456	1457	85.0	13175	6	1978	1988	119.0	
1457	1458	66.0	9042	7	1941	2006	0.0	
1458	1459	68.0	9717	5	1950	1996	0.0	
1459	1460	75.0	9937	5	1965	1965	0.0	

1460 rows × 26 columns

In [21]:
```
New_train['TotalSF'] = New_train['BsmtFinSF1'] + New_train['BsmtUnfSF']
New_train['TotalArea'] = New_train['LotArea'] + New_train['LotFrontage'
New_train
```

```
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/408751010.py:1: Set
tingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  New_train['TotalSF'] = New_train['BsmtFinSF1'] + New_train['BsmtUnf
SF'] + New_train['TotalBsmtSF'] + New_train['1stFlrSF'] + New_train
['2ndFlrSF']
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/408751010.py:2: Set
tingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  New_train['TotalArea'] = New_train['LotArea'] + New_train['LotFront
age']
```

Out[21]:

	Id	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
0	1	65.0	8450	7	2003	2003	196.0	
1	2	80.0	9600	6	1976	1976	0.0	
2	3	68.0	11250	7	2001	2002	162.0	
3	4	60.0	9550	7	1915	1970	0.0	
4	5	84.0	14260	8	2000	2000	350.0	
...	...	...	...	...	...	...	...	
1455	1456	62.0	7917	6	1999	2000	0.0	
1456	1457	85.0	13175	6	1978	1988	119.0	
1457	1458	66.0	9042	7	1941	2006	0.0	
1458	1459	68.0	9717	5	1950	1996	0.0	
1459	1460	75.0	9937	5	1965	1965	0.0	

1460 rows × 26 columns

5

Build at a minimum the following regression models. 1.Lasso 2.Ridge 3.ElasticNet

In [23]:
```
X1 = New_train.drop('SalePrice', axis=1)
y1 = New_train['SalePrice']

X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_s
```

In [24]:
```
# Lasso Linear Regression

# Standardize the features
scaler = StandardScaler()
X1_train_scaled = scaler.fit_transform(X1_train)
X1_test_scaled = scaler.transform(X1_test)

# Create and fit the Lasso Regression model
lasso = Lasso(alpha=0.1)  # Alpha is the regularization strength
lasso.fit(X1_train_scaled, y1_train)

# Predict on the testing set
y1_pred = lasso.predict(X1_test_scaled)

# Evaluate the model
mse = mean_squared_error(y1_test, y1_pred)
print("Mean Squared Error:", mse)

# Print the coefficients
print("Coefficients:", lasso.coef_)
```

```
Mean Squared Error: 1425840686.8004577
Coefficients: [  -842.98698198    255.43400238  20358.43932237  2721
0.42276863
   6346.65404207   6729.71011536   3455.64890725  14978.95739356
   9510.30366665  14896.89961183  16731.86708495  16712.14771003
  11846.84368776   4056.65801627  -1520.57933304   -590.66148712
   3014.34450982   5007.05481991  -4958.51114206  10993.26864885
   3000.31216925   3334.75709428    298.99602185 -30026.62126131
 -15619.57681094]

C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coor
dinate_descent.py:530: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations. Duality gap:
303403845660.52374, tolerance: 696659484.3571944
  model = cd_fast.enet_coordinate_descent(
```

In [25]:
```
# Ridge Linear Regression

X2_train, X2_test, y2_train, y2_test = train_test_split(X1, y1, test_s:

# Standardize the features

scaler = StandardScaler()
X2_train_scaled = scaler.fit_transform(X2_train)
X2_test_scaled = scaler.transform(X2_test)

# Create and fit the Ridge Regression model
ridge = Ridge(alpha=1.0)  # Alpha is the regularization strength
ridge.fit(X2_train_scaled, y2_train)

# Predict on the testing set
y2_pred = ridge.predict(X2_test_scaled)

# Evaluate the model
mse = mean_squared_error(y2_test, y2_pred)
print("Mean Squared Error:", mse)

# Print the coefficients
print("Coefficients:", ridge.coef_)
```

```
Mean Squared Error: 1425624259.553772
Coefficients: [ -844.40906553   199.94867447  2370.83010795 27165.210
45521
  6335.08788583  6738.56862173  3458.79285302  2047.1274246
 -3057.53228253  2447.15719345  6047.47811044  4573.14965867
 11495.07766755  4058.21509249 -1494.45571473  -575.38681356
  3029.96363171  5013.8929805  -4943.39854583 10951.49223602
  3034.25468935  3335.76550981   301.86668706  4134.13965213
  2370.78921318]
```

In [26]:
```
# ElasticNet Regression model

X3_train, X3_test, y3_train, y3_test = train_test_split(X1, y1, test_s

# Standardize the features

scaler = StandardScaler()
X3_train_scaled = scaler.fit_transform(X3_train)
X3_test_scaled = scaler.transform(X3_test)

# Create and fit the ElasticNet Regression model
elastic_net = ElasticNet(alpha=1.0, l1_ratio=0.5)  # Alpha is the overa
elastic_net.fit(X3_train_scaled, y3_train)

# Predict on the testing set
y3_pred = elastic_net.predict(X3_test_scaled)

# Evaluate the model
mse = mean_squared_error(y3_test, y3_pred)
print("Mean Squared Error:", mse)

# Print the coefficients
print("Coefficients:", elastic_net.coef_)
```

```
Mean Squared Error: 1530108367.3009803
Coefficients: [ -642.30174356  1012.00953853  2053.9856474  15298.832
49807
  5716.34610016  7004.21994282  4265.17924464  3614.25692782
  -133.26083418  3913.39489625  4897.21836289  4166.25160158
  7196.03565932  3530.09147475  3283.01534927  2088.74969152
  4209.36923384  5419.00969719 -1004.5752495   6691.3003821
  5244.45107277  3333.16348148  1350.69845946  5826.15483416
  2056.70417625]
```

Question 6: Conduct hyperparameter tuning for the ElasticNet.

In [27]:
```
import numpy as np
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error


# Split the data into training and testing sets
X4_train, X4_test, y4_train, y4_test = train_test_split(X1, y1, test_s

# Define the ElasticNet model
model2 = ElasticNet()

# Define hyperparameters to tune
param_grid = {'alpha': [0.1, 1.0, 10.0], 'l1_ratio': [0.1, 0.5, 0.9]}

# Perform grid search with cross-validation (5 folds)
grid_search = GridSearchCV(estimator=model2,
param_grid=param_grid, cv=5)
grid_search.fit(X4_train, y4_train)

# Get the best hyperparameters from grid search
best_alpha = grid_search.best_params_['alpha']
best_l1_ratio = grid_search.best_params_['l1_ratio']

# Initialize the ElasticNet model with best hyperparameters
best_model = ElasticNet(alpha=best_alpha,
l1_ratio=best_l1_ratio)
best_model.fit(X4_train, y4_train)

# Make predictions on the test set using the best model
y4_pred = best_model.predict(X4_test)

# Calculate Mean Squared Error (MSE) as a performance metric
mse = mean_squared_error(y4_test, y4_pred)
print(f"Best Alpha: {best_alpha}, Best L1 Ratio: {best_l1_ratio}")
print(f"Mean Squared Error: {mse}")
```

```
C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coo
rdinate_descent.py:530: ConvergenceWarning: Objective did not conver
ge. You might want to increase the number of iterations. Duality ga
p: 608488135555.6132, tolerance: 537415025.1745832
  model = cd_fast.enet_coordinate_descent(
C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coo
rdinate_descent.py:530: ConvergenceWarning: Objective did not conver
ge. You might want to increase the number of iterations. Duality ga
p: 593837261672.4115, tolerance: 572016182.2224231
  model = cd_fast.enet_coordinate_descent(
C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coo
rdinate_descent.py:530: ConvergenceWarning: Objective did not conver
ge. You might want to increase the number of iterations. Duality ga
p: 444480271076.4716, tolerance: 525605188.80204767
  model = cd_fast.enet_coordinate_descent(
C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coo
rdinate_descent.py:530: ConvergenceWarning: Objective did not conver
ge. You might want to increase the number of iterations. Duality ga
p: 666218471738.508, tolerance: 571614666.2949619
```

7

Evaluate performance of the model using the Kaggle metric upon which your scores are evaluated.

In [28]:
```
# Fit the model
model2.fit(X1, y1)
```

```
C:\Users\elvis\anaconda3\lib\site-packages\sklearn\linear_model\_coor
dinate_descent.py:530: ConvergenceWarning: Objective did not converg
e. You might want to increase the number of iterations. Duality gap:
1015820966255.721, tolerance: 920791133.4609977
  model = cd_fast.enet_coordinate_descent(
```

Out[28]:
```
ElasticNet()
```

In [29]:
```
ridge.fit(X1, y1)
```

Out[29]:
```
Ridge()
```

In [30]:
```
x_testing = test[['Id','LotFrontage', 'LotArea', 'OverallQual', 'YearBu
                  'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnf
                  'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',
                  'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces'
                  'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeck
                  'OpenPorchSF']]

x_testing['TotalSF'] = x_testing['BsmtFinSF1'] + x_testing['BsmtUnfSF']
x_testing['TotalArea'] = x_testing['LotArea'] + x_testing['LotFrontage'
x_testing
```

```
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/1068965064.py:8: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  x_testing['TotalSF'] = x_testing['BsmtFinSF1'] + x_testing['BsmtUnf
SF'] + x_testing['TotalBsmtSF'] + x_testing['1stFlrSF'] + x_testing
['2ndFlrSF']
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/1068965064.py:9: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  x_testing['TotalArea'] = x_testing['LotArea'] + x_testing['LotFront
age']
```

Out[30]:

	Id	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
0	1461	80.0	11622	5	1961	1961	0.0	4
1	1462	81.0	14267	6	1958	1958	108.0	9
2	1463	74.0	13830	5	1997	1998	0.0	7
3	1464	78.0	9978	6	1998	1998	20.0	6
4	1465	43.0	5005	8	1992	1992	0.0	2
...	...	...	...	...	...	...	...	
1454	2915	21.0	1936	4	1970	1970	0.0	
1455	2916	21.0	1894	4	1970	1970	0.0	2
1456	2917	160.0	20000	5	1960	1996	0.0	12
1457	2918	62.0	10441	5	1992	1992	0.0	3
1458	2919	74.0	9627	7	1993	1994	94.0	7

1459 rows × 25 columns

In [31]:

```
enp = model2.predict(x_testing)
enp
```

Out[31]:

```
array([123861.26676314, 164175.41318605, 189762.68798893, ...,
       187913.92313866, 121595.99245545, 239878.03662574])
```

In [32]:
```
x_testing["SalePrice"] = enp.round(2)
Elastic_Net_Submission = x_testing.drop(['LotFrontage', 'LotArea', 'Ove
                    'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnf
                    'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',
                    'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces'
                    'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeck
                    'OpenPorchSF','TotalSF','TotalArea'], axis=1)
Elastic_Net_Submission
```

```
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/2165622178.py:1: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  x_testing["SalePrice"] = enp.round(2)
```

Out[32]:

	Id	SalePrice
0	1461	123861.27
1	1462	164175.41
2	1463	189762.69
3	1464	201588.12
4	1465	189663.99
...	...	...
1454	2915	78988.32
1455	2916	86591.38
1456	2917	187913.92
1457	2918	121595.99
1458	2919	239878.04

1459 rows × 2 columns

In [33]:
```
ridge_test = test[['Id','LotFrontage', 'LotArea', 'OverallQual', 'YearB
                   'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnf
                   'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',
                   'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces'
                   'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeck
                   'OpenPorchSF']]

ridge_test['TotalSF'] = ridge_test['BsmtFinSF1'] + ridge_test['BsmtUnfS
ridge_test['TotalArea'] = ridge_test['LotArea'] + ridge_test['LotFronta
ridge_test
```

```
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/2421134651.py:8: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  ridge_test['TotalSF'] = ridge_test['BsmtFinSF1'] + ridge_test['Bsmt
UnfSF'] + ridge_test['TotalBsmtSF'] + ridge_test['1stFlrSF'] + ridge_
test['2ndFlrSF']
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/2421134651.py:9: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  ridge_test['TotalArea'] = ridge_test['LotArea'] + ridge_test['LotFr
ontage']
```

Out[33]:

	Id	LotFrontage	LotArea	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
0	1461	80.0	11622	5	1961	1961	0.0	4
1	1462	81.0	14267	6	1958	1958	108.0	9
2	1463	74.0	13830	5	1997	1998	0.0	7
3	1464	78.0	9978	6	1998	1998	20.0	6
4	1465	43.0	5005	8	1992	1992	0.0	2
...	...	...	...	...	...	...	...	
1454	2915	21.0	1936	4	1970	1970	0.0	
1455	2916	21.0	1894	4	1970	1970	0.0	2
1456	2917	160.0	20000	5	1960	1996	0.0	12
1457	2918	62.0	10441	5	1992	1992	0.0	3
1458	2919	74.0	9627	7	1993	1994	94.0	7

1459 rows × 25 columns

In [34]:

```
rp = ridge.predict(ridge_test)
rp
```

Out[34]:

```
array([107431.77494381, 159949.89108526, 176118.44264359, ...,
       185775.00515035, 119043.1552401 , 247079.28454426])
```

In [35]:
```
ridge_test["SalePrice"] = rp.round(2)
Submission_ridge=ridge_test.drop(['LotFrontage', 'LotArea', 'OverallQua
                    'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtUnf
                    'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',
                    'FullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces'
                    'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeck
                    'OpenPorchSF','TotalSF','TotalArea'], axis=1)
Submission_ridge
```

```
C:\Users\elvis\AppData\Local\Temp/ipykernel_18780/1670168838.py:1: Se
ttingWithCopyWarning:
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/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  ridge_test["SalePrice"] = rp.round(2)
```

Out[35]:

	Id	SalePrice
0	1461	107431.77
1	1462	159949.89
2	1463	176118.44
3	1464	196668.44
4	1465	199060.63
...	...	...
1454	2915	72838.20
1455	2916	73762.95
1456	2917	185775.01
1457	2918	119043.16
1458	2919	247079.28

1459 rows × 2 columns

In [36]:
```
Submission_ridge
```

Out[36]:

	Id	SalePrice
0	1461	107431.77
1	1462	159949.89
2	1463	176118.44
3	1464	196668.44
4	1465	199060.63
...	...	...
1454	2915	72838.20
1455	2916	73762.95
1456	2917	185775.01
1457	2918	119043.16
1458	2919	247079.28

1459 rows × 2 columns

In [37]:
```
Elastic_Net_Submission
```

Out[37]:

	Id	SalePrice
0	1461	123861.27
1	1462	164175.41
2	1463	189762.69
3	1464	201588.12
4	1465	189663.99
...	...	...
1454	2915	78988.32
1455	2916	86591.38
1456	2917	187913.92
1457	2918	121595.99
1458	2919	239878.04

1459 rows × 2 columns

In [38]:
```
Elastic_Net_Submission.to_csv('Elastic_net.csv', index=False)
Submission_ridge.to_csv('Ridge.csv', index=False)
```