## House Prices: Transformed Ridge, Lasso, and ElasticNet Linear Regression

The House Prices dataset contains 79 explanatory variables describing residential homes in Ames, Iowa, and their effects on the independent variable: SalePrice. While exploring the dataset of 1460 records, there are null values in the dependent variables, as seen in Figure 1. This is normal as these data are input by sellers and agents and could miss some information, or some variables like “PoolQC” can be null if there is no pool for the given house. To adjust, we will later replace the null value with the median for the numeric variables. Figure 2 shows a skewed distribution for the “SalePrice” after which we remove outliers in the data.

The histogram in Figure 3 supports this skewness with some values much higher than the others, indicating high variance. Checking the overall quality and sale price scatterplot, we also find the distribution is exponential for sale price and other independent variables. From this information we decided we need to perform logarithmic and exponential transformations to the data when performing lasso, ridge, and elastic net regression.

Prior to regression, we performed EDA on the dependent variable in Figures 4 and 5 exploring the correlation between “SalePrice” and quantitative dependent variables (14 have a >0.5 correlation coefficient) and category plots for each categorical variable.  It can be seen that some categories lead to higher sales prices, such as the “EX” category in “FireplaceQU”, meaning excellent numbers of fireplaces show a higher sales price.

First, we split the dataset into train and test datasets, with 80% training and 20% testing. We then encode every categorical feature into multiple boolean dummy variables using Sklearn one-hot encoding. This allows us to factor in categorical variables in our regression model. We then add two additional features that could be useful: 'Nonlivingarea' - The sum of GarageArea, PoolArea, WoodDeckSF. These are non-living spaces a house has. 'QualityCondition' - The sum of OveralQual and OverallCond. A house’s condition and quality are both important and worth evaluating together. Finally, we fill all null values with their respective variable median.

Lasso ("Least Absolute Shrinkage and Selection Operator") regression, a type of linear regression that incorporates regularization, is commonly used for feature selection and regularization to prevent overfitting in models with a large number of features. It can reduce the problem of multi-collinearity by forcing some features' coefficients to be 0 and removing the effect of those features. It minimizes overfitting by penalizing large coefficients, and automatically performs feature selection. First, we use untransformed sales prices with LassoCV to find the best hyperparameters, which did not converge. Then we applied a logarithmic transformation to normalize Saleprice, stabilize variance, and linearize relationships. The Lasso model converged with an optimum alpha level of 0.007 and R^2 Score of 0.9 on the test dataset (Figure 9 and 10). The lasso model has 56 features, a reduction from 286, whose absolute values of coefficients are greater than 0.001. We ran the model on the Kaggle test dataset and achieved a 0.15 score (Figure 13). Our Ridge regression model resulted in an R^2 score of around 0.92 (Figure 7) and a Kaggle score of 0.17.

Our Grid Search, Elastic Net model supported our finding that the Lasso model performed better. As seen in Figure 11, The ideal l1_ratio hyperparameter was 0.1 which is much closer to the Lasso end of the spectrum. This model scored a 0.15 on Kaggle as seen in Figure 13 as well. As compared to last week, we show that a logarithmic transformation paired with an ElasticNetCV regression model can perform very well in predicting housing sale prices in Ames, Iowa.

## Appendix

```
from google.colab import drive
```

```
drive.mount('/content/drive')
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
%cd /content/drive/My Drive/
df = pd.read_csv('Housing_Price/train.csv')
df.head(5)
```

| | Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | Alley | LotShape | LandContour | Utilities | ... | PoolArea | PoolQC | Fence | MiscFeature | MiscVal | MoSold | YrSold | Sa |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 60 | RL | 65.0 | 8450 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 2 | 2008 | |
| 1 | 2 | 20 | RL | 80.0 | 9600 | Pave | NaN | Reg | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 5 | 2007 | |
| 2 | 3 | 60 | RL | 68.0 | 11250 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 9 | 2008 | |
| 3 | 4 | 70 | RL | 60.0 | 9550 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 2 | 2006 | |
| 4 | 5 | 60 | RL | 84.0 | 14260 | Pave | NaN | IR1 | Lvl | AllPub | ... | 0 | NaN | NaN | NaN | 0 | 12 | 2008 | |

5 rows × 81 columns

```
#Null Value Columns
nullseries= df.isna().sum()
print(nullseries[nullseries > 0])
```

```
LotFrontage      259
Alley           1369
MasVnrType       872
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Electrical         1
FireplaceQu      690
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
GarageCond        81
PoolQC          1453
Fence           1179
MiscFeature     1406
dtype: int64
```

Figure 1

## Drop Outliers & EDA

```
x = df.drop(columns=['SalePrice'])
y = df['SalePrice']
```

```python
# Use K-Fold Later
from sklearn.model_selection import train_test_split
# Split the dataset into 80% train and 20% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=42)

# Histogram for SalePrice
plt.figure(figsize=(12,6))
sns.histplot(y,kde=True)
plt.title('Histogram of SalePrice')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')
```

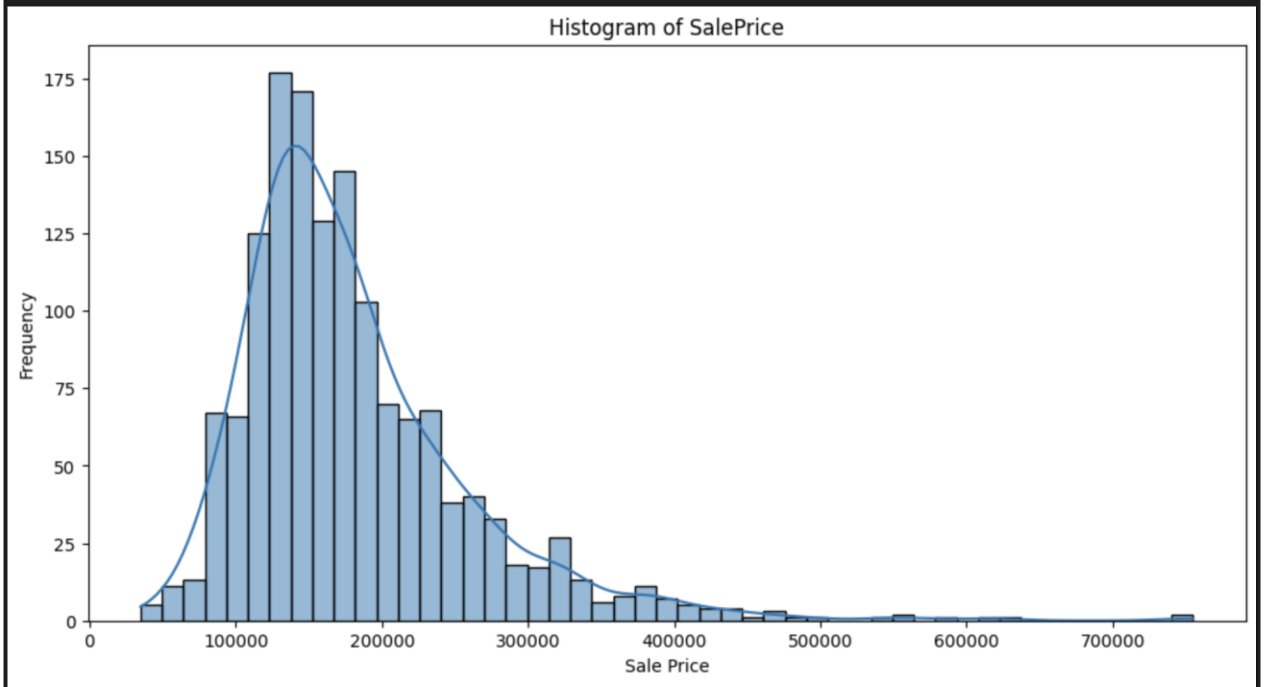



Figure 2

```python
plt.scatter(x['OverallQual'], y, color='blue', alpha=0.7)
```

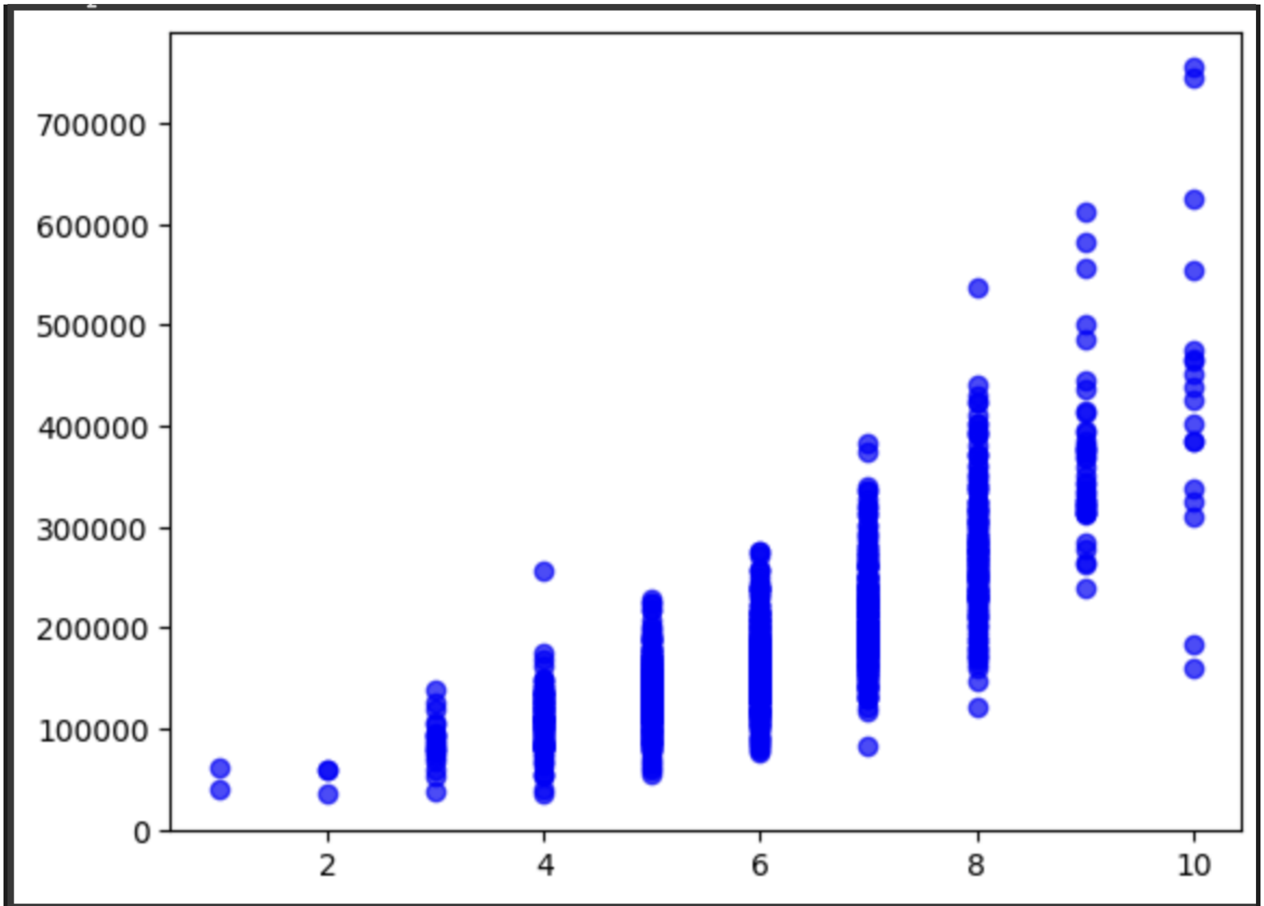



Figure 3

```
# drop outliers
Q1 = y.quantile(0.25)
Q3 = y.quantile(0.75)
IQR = Q3-Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = y_train[(y_train < lower_bound) | (y_train > upper_bound)]

# Remove outliers from the dataset
y_cleaned = y_train.drop(outliers.index)
x_cleaned = x_train.drop(outliers.index)

#Create heat map of all numeric variables
# Select only numeric variables
```

```python
x_cleaned_numeric = x_cleaned.select_dtypes(include=['int64',
'float64']).drop(columns=['Id'])

# Calculate the correlation matrix
train_data = pd.concat([x_cleaned_numeric, y_cleaned], axis=1)
correlation_matrix = train_data.corr()
correlation_matrix =
correlation_matrix['SalePrice'].drop('SalePrice').sort_values()
correlation_matrix=correlation_matrix.sort_values()

# Plot barplot for correlation
plt.figure(figsize=(12, 10))
bar_plot = sns.barplot(x=correlation_matrix.index, y=correlation_matrix,
palette='coolwarm')
plt.title('Correlation of Salesprice and other Numeric Variables')
bar_plot.set_xticklabels(bar_plot.get_xticklabels(), rotation=60,
ha='right') # Rotate x-axis labels
plt.xlabel('Numeric Factors')
plt.ylabel('Correlation with SalePrice')
for index, value in enumerate(correlation_matrix):
plt.text(index, value, f'{value:.2f}', ha='center', va='bottom')
plt.tight_layout() # Adjust layout to prevent clipping of labels
plt.show()

# I can see 13 variables has correlation greater than 0.5
```

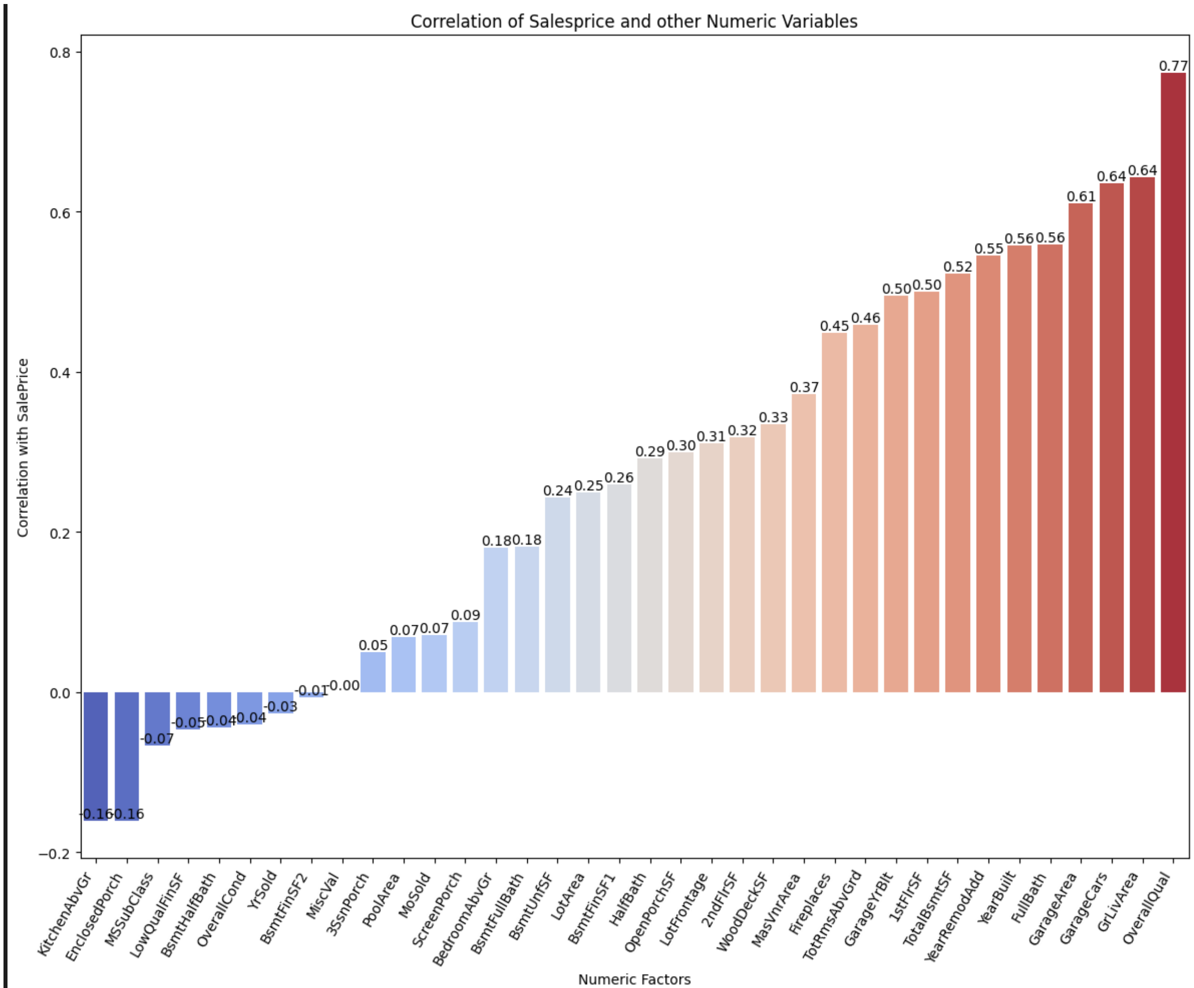



Figure 4

```
categorical_variables =
x_cleaned.select_dtypes(include=['object']).columns.tolist()
categorical_variables

['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
```

```
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']

#This is designed to run once
import numpy as np

# Create subplots
num_plots = len(categorical_variables)
cols_per_row = 4
rows = num_plots // cols_per_row + 1
```

```
fig, axes = plt.subplots(rows, cols_per_row, figsize=(18, 4*rows))

for idx, col in enumerate(categorical_variables):
row_idx = idx // cols_per_row
col_idx = idx % cols_per_row
ax = axes[row_idx, col_idx]
sns.barplot(x=x_cleaned[col], y=y_cleaned, estimator=np.median, ax=ax)
ax.set_title(f'{col} vs SalePrice (Median)')
ax.tick_params(axis='x', rotation=45) # Rotate x-axis labels for better
readability
ax.set_xlabel('') # Remove x-axis label for better clarity
ax.set_ylabel('SalePrice')
plt.tight_layout() # Adjust layout to prevent overlapping
plt.subplots_adjust(hspace=0.5) # Add vertical spacing between subplots

# Remove empty subplots if any
if num_plots % cols_per_row != 0:
for i in range(cols_per_row - (num_plots % cols_per_row)):
fig.delaxes(axes[-1, -(i+1)])

# # I can see MSZoning 'FV', Pave street, Pave Alley, IR2 lot shape, HLS
land contour, Allpub utility,
# # NoRidge/NridgHr/StroneBr Neighborhood, PosN/PosA Condition2,
1Fam/TwnhsE BldgType, 2stroy/SLvl HouseStyle,
# # WdShngl RoofMatl, Exterior CemntBd/Stone/Imstucc/VinylSd, EX BsmtQual,
PConc Foundation, Gd BsmtCond, GLQ BsmtFinType1, GasA Heating, Y Central
Air
# # SBrkr Electrical, Ex heatingQC, Ex KitchenQual, Ex FireplaceQu,
Builtin GarageType, Gd GarageQual, Y PavedDrive, Ex PoolQC, Tenc
MiscFeature, Con/New for SaleType
# # Partial SalesCondition
```


MSZoning vs SalePrice (Median)
Street vs SalePrice (Median)
Alley vs SalePrice (Median)
LotShape vs SalePrice (Median)
LandContour vs SalePrice (Median)
Utilities vs SalePrice (Median)
LotConfig vs SalePrice (Median)
LandSlope vs SalePrice (Median)
Neighborhood vs SalePrice (Median)
Condition1 vs SalePrice (Median)
Condition2 vs SalePrice (Median)
BldgType vs SalePrice (Median)
HouseStyle vs SalePrice (Median)
RoofStyle vs SalePrice (Median)
RoofMatl vs SalePrice (Median)
Exterior1st vs SalePrice (Median)
Exterior2nd vs SalePrice (Median)
MasVnrType vs SalePrice (Median)
ExterQual vs SalePrice (Median)
ExterCond vs SalePrice (Median)
Foundation vs SalePrice (Median)
BsmtQual vs SalePrice (Median)
BsmtCond vs SalePrice (Median)
BsmtExposure vs SalePrice (Median)
SalePrice

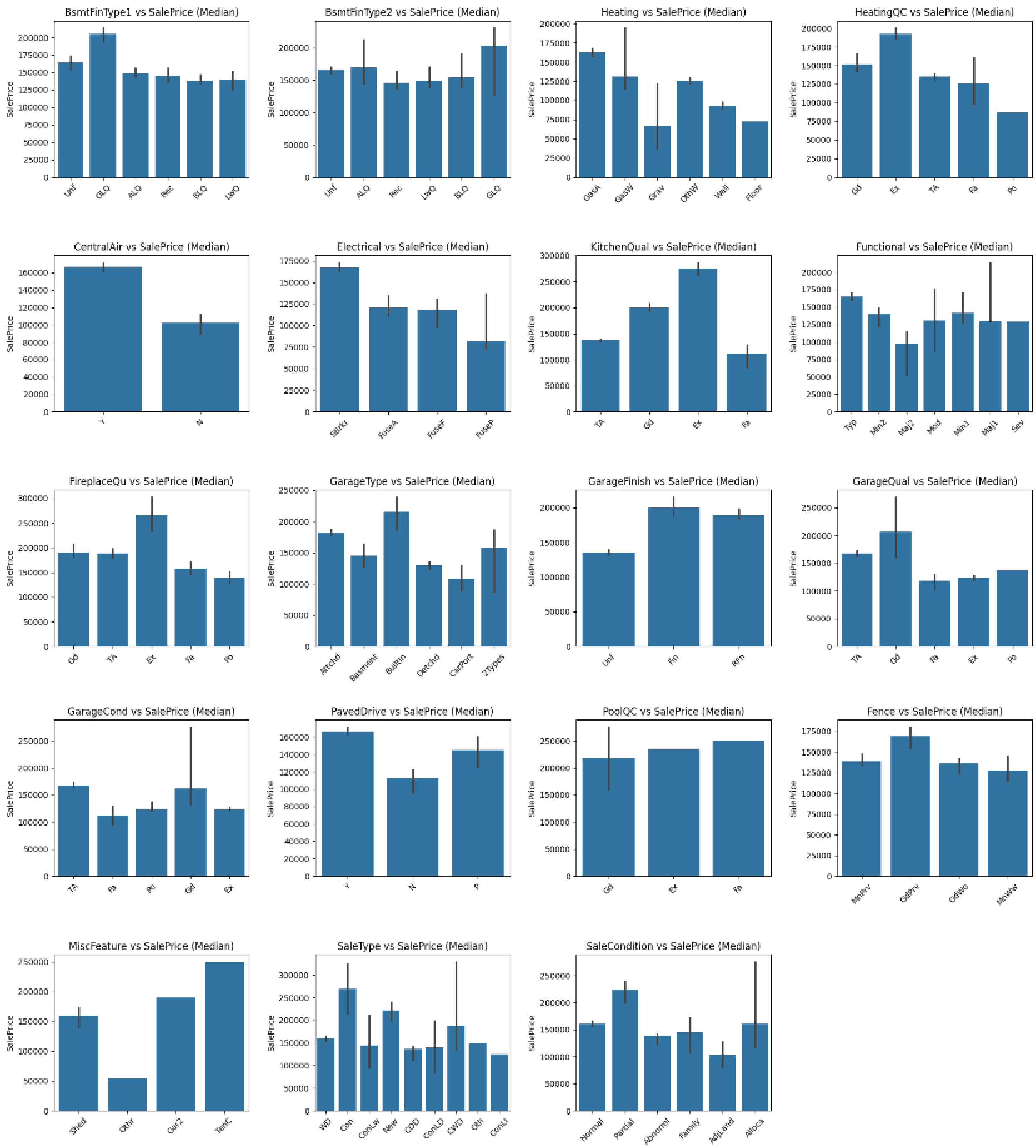



Figure 5

## Feature Engineering

```
#Feature Engineering: Step 1 One Key Encoding Categorical Variable to
boolean column
x_encoded = pd.get_dummies(x_cleaned)
```

```
x_encoded.head(5)
```

| | Id | MSSubClass | LotFrontage | LotArea | OverallQual | OverallCond | YearBuilt | YearRemodAdd | MasVnrArea | BsmtFinSF1 | ... | SaleType_ConLw | SaleType_New | SaleType_Oth |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **135** | 136 | 20 | 80.0 | 10400 | 7 | 6 | 1970 | 1970 | 288.0 | 0 | ... | False | False | False |
| **1452** | 1453 | 180 | 35.0 | 3675 | 5 | 5 | 2005 | 2005 | 80.0 | 547 | ... | False | False | False |
| **762** | 763 | 60 | 72.0 | 8640 | 7 | 5 | 2009 | 2009 | 0.0 | 24 | ... | False | False | False |
| **932** | 933 | 20 | 84.0 | 11670 | 9 | 5 | 2006 | 2006 | 302.0 | 0 | ... | False | False | False |
| **435** | 436 | 60 | 43.0 | 10667 | 7 | 6 | 1996 | 1996 | 0.0 | 385 | ... | True | False | False |

5 rows × 283 columns

```
#Feature Engineering: Step 2 Create Some Additional Variables
x_encoded['Nonlivingarea']=x_encoded['GarageArea']+x_encoded['PoolArea']+x
_encoded['WoodDeckSF']
x_encoded['QualityCondition']=x_encoded['OverallQual']+x_encoded['OverallC
ond']

#Feature Engineering: Step 3 Fill NaN with Median
x_encoded = x_encoded.fillna(x_encoded.median())
```

## Ridge Regression using Standard Scaling, log transform SalePrice

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV
import statsmodels.api as sm
from sklearn.model_selection import ShuffleSplit, cross_val_score

# Standard scaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_encoded) # fit it on the training data

x_scaled_with_const = sm.add_constant(x_scaled)

k_fold = ShuffleSplit(n_splits=10, random_state=0, test_size=0.25,
train_size=None)
ridge_model = RidgeCV(alphas=np.arange(0.01,1,.01), cv=k_fold)
ridge_model.fit(x_scaled_with_const,np.log(y_cleaned))

print(ridge_model.alpha_)
```

```
0.99
```

Figure 6

```
#One Key Encoding Categorical Variable to boolean column
x_test_cleaned_encoded = pd.get_dummies(x_test)

# Create Some Additional Variables
x_test_cleaned_encoded['Nonlivingarea']=x_test_cleaned_encoded['GarageArea
']+x_test_cleaned_encoded['PoolArea']+x_test_cleaned_encoded['WoodDeckSF']
x_test_cleaned_encoded['QualityCondition']=x_test_cleaned_encoded['Overall
Qual']+x_test_cleaned_encoded['OverallCond']

# Fill NaN with Median
x_test_cleaned_encoded =
x_test_cleaned_encoded.fillna(x_test_cleaned_encoded.median())

# Make sure missing dummy variables are added to test dataset and drop
those not in trained dataset
missing_cols = set(x_test_cleaned_encoded.columns) -
set(x_encoded.columns)
x_test_cleaned_encoded=x_test_cleaned_encoded.drop(columns=missing_cols)

missing_cols = set(x_encoded.columns)- set(x_test_cleaned_encoded.columns)
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
x_test_cleaned_encoded[c] = False
# Ensure the order of column in the test set is in the same order than in
train set
print(len(x_test_cleaned_encoded.columns))
print(len(x_encoded.columns))

# Ensure that the columns order is the same in both datasets
train_columns = x_encoded.columns

x_test_cleaned_encoded = x_test_cleaned_encoded[train_columns]

scaler = StandardScaler()
x_test_scaled = scaler.fit_transform(x_test_cleaned_encoded) # fit it on
the training data
```

```
# predict using trained model
x_test_scaled = sm.add_constant(x_test_scaled)

y_pred = ridge_model.predict(x_test_scaled)

#Performance Evaluation
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


mse = mean_squared_error(y_test, np.exp(y_pred))
print("Mean Squared Error:", mse)


# Calculate the R^2 score
r2 = r2_score(y_test, np.exp(y_pred))
print("R^2 Score:", r2)

Mean Squared Error: 521085126.4714137 R^2 Score: 0.925325576823251
```

Figure 7

## Make prediction of Kaggle Test Dataset

```
df=pd.read_csv('Housing_Price/test.csv')

#One Key Encoding Categorical Variable to boolean column
df_encoded = pd.get_dummies(df)

# Create Some Additional Variables
df_encoded['Nonlivingarea']=df_encoded['GarageArea']+df_encoded['PoolArea'
]+df_encoded['WoodDeckSF']
df_encoded['QualityCondition']=df_encoded['OverallQual']+df_encoded['Overa
llCond']

# Fill NaN with Median
df_encoded = df_encoded.fillna(df_encoded.median())
```

```
# Make sure missing dummy variables are added to test dataset and drop those not in trained dataset
missing_cols = set(df_encoded.columns) - set(x_encoded.columns)
df_encoded=df_encoded.drop(columns=missing_cols)

missing_cols = set(x_encoded.columns)- set(df_encoded)
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
df_encoded[c] = False
# Ensure the order of column in the test set is in the same order than in train set
print(len(df_encoded.columns))
print(len(x_encoded.columns))

# Ensure that the columns order is the same in both datasets
train_columns = x_encoded.columns

df_encoded = df_encoded[train_columns]

scaler = StandardScaler()
df_encoded_scaled = scaler.fit_transform(df_encoded) # fit it on the training data

# predict using trained model
df_encoded_scaled = sm.add_constant(df_encoded_scaled)

y_pred = ridge_model.predict(df_encoded_scaled)

np.exp(y_pred)
result=pd.concat([df['Id'],pd.DataFrame(np.exp(y_pred))],axis=1)
result.rename(columns={0:'SalePrice'},inplace=True)
result.to_csv('Housing_Price/ridge_model_prediction.csv',index=False)
```

## Regression Model 1 Lasso Regression Standard Scaling, no log transform

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV
```

```python
import statsmodels.api as sm
from sklearn.model_selection import ShuffleSplit, cross_val_score
import numpy as np

# Standard scaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_encoded) # fit it on the training data

x_scaled_with_const = sm.add_constant(x_scaled)

lasso_model1 = LassoCV(alphas=np.arange(0.01,1,0.1),cv=2)
lasso_model1.fit(x_scaled_with_const,y_cleaned)

print(lasso_model1.alpha_)

#Model Does not Converge, need log transform dependent Variable

Figure 8
```

## Regression Model 2 Lasso Regression Standard Scaling, log transform SalePrice

```python
# Create Linear Model with Standard Scaling of Independent Variable
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm


scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_encoded) # fit it on the training data


x_scaled_with_const = sm.add_constant(x_scaled)


k_fold = ShuffleSplit(n_splits=10, random_state=0, test_size=0.25,
train_size=None)
lasso_model2 = LassoCV(alphas=np.arange(0.001,1,.001),cv=k_fold)
```

```
#log transform y_cleaned
lasso_model2.fit(x_scaled_with_const,np.log(y_cleaned))


print(lasso_model2.alpha_)
```

```
0.007
```

Figure 9

```
#Performance Evaluation
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


y_pred2 = lasso_model2.predict(x_test_scaled)

mse = mean_squared_error(y_test, np.exp(y_pred2))
print("Mean Squared Error:", mse)


# Calculate the R^2 score
r2 = r2_score(y_test, np.exp(y_pred2))
print("R^2 Score:", r2)
```

```
Mean Squared Error: 732493053.8859528 R^2 Score: 0.8950296343127162
```

Figure 10

```
len([i for i in lasso_model2.coef_ if abs(i) > 0.001])
```

```
56
```

## Make prediction of Kaggle Test Dataset

```
df=pd.read_csv('Housing_Price/test.csv')
#One Key Encoding Categorical Variable to boolean column
df_encoded = pd.get_dummies(df)
```

```python
# Create Some Additional Variables
df_encoded['Nonlivingarea']=df_encoded['GarageArea']+df_encoded['PoolArea']+df_encoded['WoodDeckSF']
df_encoded['QualityCondition']=df_encoded['OverallQual']+df_encoded['OverallCond']

# Fill NaN with Median
df_encoded = df_encoded.fillna(df_encoded.median())


# Make sure missing dummy variables are added to test dataset and drop those not in trained dataset
missing_cols = set(df_encoded.columns) - set(x_encoded.columns)
df_encoded=df_encoded.drop(columns=missing_cols)

missing_cols = set(x_encoded.columns)- set(df_encoded)
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
df_encoded[c] = False
# Ensure the order of column in the test set is in the same order than in train set
print(len(df_encoded.columns))
print(len(x_encoded.columns))

# Ensure that the columns order is the same in both datasets
train_columns = x_encoded.columns

df_encoded = df_encoded[train_columns]

scaler = StandardScaler()
df_encoded_scaled = scaler.fit_transform(df_encoded) # fit it on the training data

# predict using trained model
df_encoded_scaled = sm.add_constant(df_encoded_scaled)

y_pred = lasso_model2.predict(df_encoded_scaled)
np.exp(y_pred)
```

```
result=pd.concat([df['Id'],pd.DataFrame(np.exp(y_pred))],axis=1)
result.rename(columns={0:'SalePrice'},inplace=True)
result.to_csv('Housing_Price/lasso_modelprediction.csv',index=False)
```

# ElasticNet Regression using Standard Scaling, log transform SalePrice

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
import statsmodels.api as sm
from sklearn.model_selection import ShuffleSplit, cross_val_score
import numpy as np

# Standard scaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_encoded) # fit it on the training data

x_scaled_with_const = sm.add_constant(x_scaled)

k_fold = ShuffleSplit(n_splits=10, random_state=0, test_size=0.25,
train_size=None)
param_grid = {'alpha': np.arange(0.01, 1.0, 0.01), 'l1_ratio': [0.1, 0.2,
0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}
e_net = ElasticNet(alpha=0.1, l1_ratio=0.5, max_iter=10000,
random_state=42)
grid_model = GridSearchCV(e_net, param_grid, cv=k_fold, n_jobs = -1)
grid_model.fit(x_scaled_with_const, np.log(y_cleaned))

print("Best parameters : {}".format(grid_model.best_params_))
print("Best cross validation score:
{:.2f}".format(grid_model.best_score_))
print("Best estimator: {}".format(grid_model.best_estimator_))

Best parameters : {'alpha': 0.06999999999999999, 'l1_ratio': 0.1} Best
cross validation score: 0.84 Best estimator:
ElasticNet(alpha=0.06999999999999999, l1_ratio=0.1, max_iter=10000,
random_state=42)
```

Figure 11

```
#One Key Encoding Categorical Variable to boolean column
x_test_cleaned_encoded = pd.get_dummies(x_test)

# Create Some Additional Variables
x_test_cleaned_encoded['Nonlivingarea']=x_test_cleaned_encoded['GarageArea
']+x_test_cleaned_encoded['PoolArea']+x_test_cleaned_encoded['WoodDeckSF']
x_test_cleaned_encoded['QualityCondition']=x_test_cleaned_encoded['Overall
Qual']+x_test_cleaned_encoded['OverallCond']

# Fill NaN with Median
x_test_cleaned_encoded =
x_test_cleaned_encoded.fillna(x_test_cleaned_encoded.median())

# Make sure missing dummy variables are added to test dataset and drop
those not in trained dataset
missing_cols = set(x_test_cleaned_encoded.columns) -
set(x_encoded.columns)
x_test_cleaned_encoded=x_test_cleaned_encoded.drop(columns=missing_cols)

missing_cols = set(x_encoded.columns)- set(x_test_cleaned_encoded.columns)
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
x_test_cleaned_encoded[c] = False
# Ensure the order of column in the test set is in the same order than in
train set
print(len(x_test_cleaned_encoded.columns))
print(len(x_encoded.columns))

# Ensure that the columns order is the same in both datasets
train_columns = x_encoded.columns

x_test_cleaned_encoded = x_test_cleaned_encoded[train_columns]

scaler = StandardScaler()
x_test_scaled = scaler.fit_transform(x_test_cleaned_encoded) # fit it on
the training data
```

```python
# predict using trained model
x_test_scaled = sm.add_constant(x_test_scaled)

y_pred = grid_model.predict(x_test_scaled)

#Performance Evaluation
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


mse = mean_squared_error(y_test, np.exp(y_pred))
print("Mean Squared Error:", mse)


# Calculate the R^2 score
r2 = r2_score(y_test, np.exp(y_pred))
print("R^2 Score:", r2)
```

```
Mean Squared Error: 842684381.5541793 R^2 Score: 0.879238598616831
```

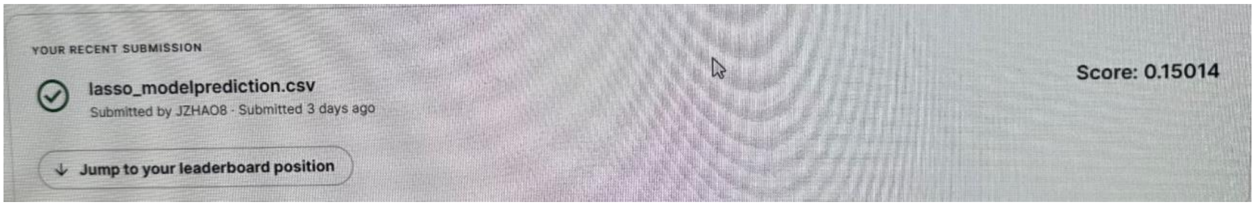
Figure 12

## Make prediction of Kaggle Test Dataset

```python
df=pd.read_csv('Housing_Price/test.csv')

#One Key Encoding Categorical Variable to boolean column
df_encoded = pd.get_dummies(df)

# Create Some Additional Variables
df_encoded['Nonlivingarea']=df_encoded['GarageArea']+df_encoded['PoolArea'
]+df_encoded['WoodDeckSF']
df_encoded['QualityCondition']=df_encoded['OverallQual']+df_encoded['Overa
llCond']

# Fill NaN with Median
df_encoded = df_encoded.fillna(df_encoded.median())
```

```
# Make sure missing dummy variables are added to test dataset and drop those not in trained dataset
missing_cols = set(df_encoded.columns) - set(x_encoded.columns)
df_encoded=df_encoded.drop(columns=missing_cols)

missing_cols = set(x_encoded.columns)- set(df_encoded)
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
df_encoded[c] = False
# Ensure the order of column in the test set is in the same order than in train set
print(len(df_encoded.columns))
print(len(x_encoded.columns))

# Ensure that the columns order is the same in both datasets
train_columns = x_encoded.columns

df_encoded = df_encoded[train_columns]

scaler = StandardScaler()
df_encoded_scaled = scaler.fit_transform(df_encoded) # fit it on the training data

# predict using trained model
df_encoded_scaled = sm.add_constant(df_encoded_scaled)

y_pred = grid_model.predict(df_encoded_scaled)

np.exp(y_pred)

result=pd.concat([df['Id'],pd.DataFrame(np.exp(y_pred))],axis=1)
result.rename(columns={0:'SalePrice'},inplace=True)
result.to_csv('Housing_Price/grid_model_prediction.csv',index=False)
```

## Kaggle Results


YOUR RECENT SUBMISSION
lasso_modelprediction.csv
Submitted by JZHAO8 · Submitted 3 days ago
Jump to your leaderboard position
Score: 0.15014


YOUR RECENT SUBMISSION
grid_model_prediction-3.csv
Submitted by Zachary Cmiel · Submitted 3 days ago
Jump to your leaderboard position
Score: 0.15133

Figure 13