Code for this assignment can be found here: LINK

All submissions and scores are provided in a table at the end.

Continuing from Module 1 & 2, I wanted to take a more systematic approach to comparing prediction outcomes for all the different regressions I wanted to run on the House Prices dataset. Like the last assignment, I took the same steps to merge and clean data. I used one-hot encoding to convert categorical variables into Booleans and finally split the dataset back to training and test. To get a baseline understanding how an OLS regression performs I ran two regressions (reg1 and reg2) without any engineered features or standardization. The first included only the numerical features while the second included both numerical and categorical features. I was surprised to see that just the numerical features did not create a great model. This showed that the categorical features were in fact critical to get a better prediction.

I then proceeded to create additional features inspired by previous assignments, features AvgSalePrice and NbhoodSpace are engineered features that really stand out as adding them seems to capture a lot of the interaction of all the categorical variables. Of course there were other engineered variables added too, but those captured interactions between numerical variables while the two mentioned above broke down neighborhood characteristics and assigned numerical values to them. It's interesting to see that reg3 with just numerical features and the engineered features beats reg2.

Once I had the baselines, I proceeded to standardize the numerical variables before moving into advanced regression techniques. As expected, standardization does not change the prediction power of OLS model regressions (reg3 vs reg4), however it does help with interpreting coefficients and comparing them with each other. Comparing reg2, reg3, reg4 and reg5 its easy to conclude that there is quite a lot of multicollinearities between categorical variables and also between categorical variables and numerical/engineered variables. Going into the first Lasso regression (reg6), I experimented with different alpha values to see how things change. When only considering the set of numerical features (including engineering features), adjusting alpha barely provides a better model. Generally, the predictions were better when alpha was higher but not by much. This could mean that the features already explain the target well, again even without using the categorical variables. Next, I added in all the categorical variables (reg7) and was able to see significant improvement in the model. This goes to show that Lasso does a great job to shrink the coefficients of variables that matter less, these happen to be mostly categorical variables which is consistent with our observations so far. This model performs better than reg6. The Ridge regression (reg8) also seems to be overfit with too many noisy variables, the model performs better as alpha increases and still has room to improve.

And finally, the Elastic Net. A baseline regression (reg9) was run before hyperparameter tuning which yielded that for this dataset the best alpha was 0.0152 and best l1 ratio was 0.5. Hyperparameter tuning for the Elastic Net was conducted using the ElasticNetCV() function that iterates through various values of alpha and the l1 ratio trying to identify the point which results in a model with the lowest mean square error across the dataset. The final regression (reg10) takes the identified optimal parameter values into account, however underperforms. I believe this could be due to too many irrelevant and noisy features. Next steps will be further EDA to eliminate more features and utilizing the Lasso regression to help filter some variables out.

Models that do not include categorical variables tend to poorly predict as house prices increase. The models tend to undervalue these houses. This divergence is particularly obvious past the $400k mark. This could be because higher priced, luxury houses have unique features and finishes that are harder to capture in a standard dataset. EDA was conducted to drill down into the variables with low coefficients as identified by the Lasso regressions. From EDA a next step could be to reduce levels (by merging several levels) for some categorical variables where the coefficients and p-values are low. For example, for "Electrical" two levels: SBrkr or not SBrkr. It also might make sense to combine some variables into composite variables for example Exterior1st and Exterior2nd can be combined, and low frequency combinations can be merged into a category called "other". Interestingly, the most common combinations have the same material for Exterior1st and Exterior2nd.

Model	Type	Score	Notes
reg1	OLS	0.33755	Only numeric, no standardization, no engineered features
reg2	OLS	0.18950	All features, no standardization, no engineered features
reg3	OLS	0.17434	Only numeric including engineered features, no standardization
reg4	OLS	0.17434	Only numeric including engineered features, standardized
reg5	OLS	0.18607	All features including engineered features, standardized
reg6	Lasso	0.17434	Only numeric including engineered features, standardized, $\alpha = 0.1$
reg6	Lasso	0.17433	Only numeric including engineered features, standardized, $\alpha = 1$
reg6	Lasso	0.17428	Only numeric including engineered features, standardized, $\alpha = 10$
reg6	Lasso	0.17425	Only numeric including engineered features, standardized, $\alpha = 100$
reg7	Lasso	0.17849	All features including engineered features, standardized, $\alpha = 1$
reg7	Lasso	0.16184	All features including engineered features, standardized, $\alpha = 10$
reg7	Lasso	0.15335	All features including engineered features, standardized, $\alpha = 100$
reg8	Ridge	0.16387	All features including engineered features, standardized, $\alpha = 1$
reg8	Ridge	0.15392	All features including engineered features, standardized, $\alpha = 10$
reg8	Ridge	0.15216	All features including engineered features, standardized, $\alpha = 100$

reg9	E-Net	0.14758	All features incl. engineered features, standardized, α = 1, l1 = 0.5
reg10	E-Net	0.15413	All features incl. engineered features, std., α = 0.0152, l1 = 0.5

2582 Vibhu Vanjari 0.14758 31 2m

Your Best Entry!
Your submission scored 0.15413, which is not an improvement of your previous score. Keep trying!

reg10_prediction.csv Complete · 2m ago · Elastic Net - All numeric and categorical features, standardized, includes engineered features, alpha = 0.0152, l1 ratio = 0.5	0.15413
reg9_prediction.csv Complete · 3m ago · Elastic Net - All numeric and categorical features, standardized, includes engineered features, alpha = 1, l1 ratio = 0.5	0.14758
reg8_prediction (2).csv Complete · 5m ago · Ridge - All numeric and categorical features, standardized, includes engineered features, alpha = 100	0.15216
reg8_prediction (1).csv Complete · 7m ago · Ridge - All numeric and categorical features, standardized, includes engineered features, alpha = 10	0.15392
reg8_prediction.csv Complete · 8m ago · Ridge - All numeric and categorical features, standardized, includes engineered features, alpha = 1	0.16387
reg7_prediction (3).csv Complete · 11m ago · Lasso - All numeric and categorical features, standardized, includes engineered features, alpha = 100	0.15335

Introduction

Link to access this code - https://colab.research.google.com/drive/1E01avkCRSyan6wbelTYzigPE5Fy33OQh

Earlier versions of this code include additional EDA, reasoning for cleanup and handling missing data.

- v0 - https://colab.research.google.com/drive/1gKRrXN0jYrhelwl3eefoSEj9gh3Fwewl
- v1 - https://colab.research.google.com/drive/1Cg5IznYQKiKiPJjcd3S_TltXvsYuW5Cw#scrollTo=WeDqFsFMadku
- v2 - https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn#scrollTo=9HWrWlpE3gFQ

Data taken from - https://www.kaggle.com/c/house-prices-advanced-regression-techniques/

Import modules and data files

[ ] ↳ 5 cells hidden

Merge and clean data

[ ] ↳ 23 cells hidden

Regression Analysis (without engineered features)

MLR (reg1)

All numeric features, no categorical features, no standardization, no engineered features

Kaggle Score: 0.33755

```
# Select the features and target variable
features = numeric_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features]
y = df_train[target]

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg1 = sm.OLS(y, X)
results = reg1.fit()

print(results.summary())
```

Show hidden output

```
# Add predictions to test dataset
df_test['reg1_SalePrice'] = results.predict(sm.add_constant(df_test[features]))


# Exporting prediction into a csv
df_test[['Id', 'reg1_SalePrice']].to_csv('reg1_prediction.csv', index=False)
```

MLR (reg2)

All numeric and categorical features, no standardization, no engineered features

Kaggle Score: 0.18950

[ ] ↳ 3 cells hidden

Creating additional features and refreshing test & train datasets

[ ] ↳ 12 cells hidden

Regression Analysis (with engineered features)

MLR (reg3)

All numeric features, no categorical features, no standardization, includes engineered features

Kaggle Score: 0.17434

TAKEAWAY: New engineered features are doing a great job, I suspect creating numerical features related to neighborhood characteristics has reduced the need to consider a lot of categorical variables.

```
# Select the features and target variable
features = numeric_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg3 = sm.OLS(y, X)
results = reg3.fit()

print(results.summary())
```

Show hidden output

```
# Add predictions to the training dataset
df_train['reg3_SalePrice'] = results.predict(sm.add_constant(df_train[features]))

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg3_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg3_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg3_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg3_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg3_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.show()
```

```
Correlation: 0.9247404508565895
RMSE: 30,225
MAE: 19,023
R² Score: 0.8551
```


```
# Add predictions to test dataset
df_test['reg3_SalePrice'] = results.predict(sm.add_constant(df_test[features]))
```

```
# Exporting prediction into a csv
df_test[['Id', 'reg3_SalePrice']].to_csv('reg3_prediction.csv', index=False)
```

Standardization before regularization techniques

Standardize all numeric variables

```
# Standardize numeric features in df_train and df_test datasets
scaler = StandardScaler()
df_train[numeric_features] = scaler.fit_transform(df_train[numeric_features])
df_test[numeric_features] = scaler.transform(df_test[numeric_features])
```

MLR (reg4)

All numeric features, no categorical features, numerical features are standardized, includes engineered features

Kaggle Score: 0.17434

TAKEAWAY: Standardization doesn't change OLS model regression, but it does help interpret weights of coefficients with respect to each other.

```
# Select the features and target variable
features = numeric_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg4 = sm.OLS(y, X)
results = reg4.fit()

print(results.summary())
```

Show hidden output

```
# Add predictions to test dataset
df_test['reg4_SalePrice'] = results.predict(sm.add_constant(df_test[features]))
```

```
# Exporting prediction into a csv
df_test[['Id', 'reg4_SalePrice']].to_csv('reg4_prediction.csv', index=False)
```

MLR (reg5)

All numeric and categorical features, numerical features are standardized, includes engineered features

Kaggle Score: 0.18607

TAKEAWAY: Including all the features and engineered features overfits the model.

```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Add constant term
X = sm.add_constant(X)

# Create and fit model
reg5 = sm.OLS(y, X)
results = reg5.fit()

print(results.summary())
```

Show hidden output

```
# Add predictions to test dataset
df_test['reg5_SalePrice'] = results.predict(sm.add_constant(df_test[features]))
```

```
# Exporting prediction into a csv
df_test[['Id', 'reg5_SalePrice']].to_csv('reg5_prediction.csv', index=False)
```

Lasso, Ridge and Elastic Net Regression

Lasso (reg6)

All numeric features, standardized, includes engineered features

Kaggle Score: 0.17425 (alpha = 100)

Kaggle Score: 0.17428 (alpha = 10)

Kaggle Score: 0.17433 (alpha = 1)

Kaggle Score: 0.17434 (alpha = 0.1)

TAKEAWAY: Since alpha doesn't impact the prediction much, I assume the features explain the target well.

[ ] ↳ 4 cells hidden

Lasso (reg7)

All numeric and categorical features, standardized, includes engineered features

Kaggle Score: 0.17849 (alpha = 1)

Kaggle Score: 0.16184 (alpha = 10)

Kaggle Score: 0.15335 (alpha = 100)

TAKEAWAY: Lasso does better when there are more variables that are correlated when compared to OLS. Lasso is able to shrink the redundant categorical variables that were previously overfitting the model and bring out the value of the other categorical variables as it this model performs better than the previous set of Lasso regressions.

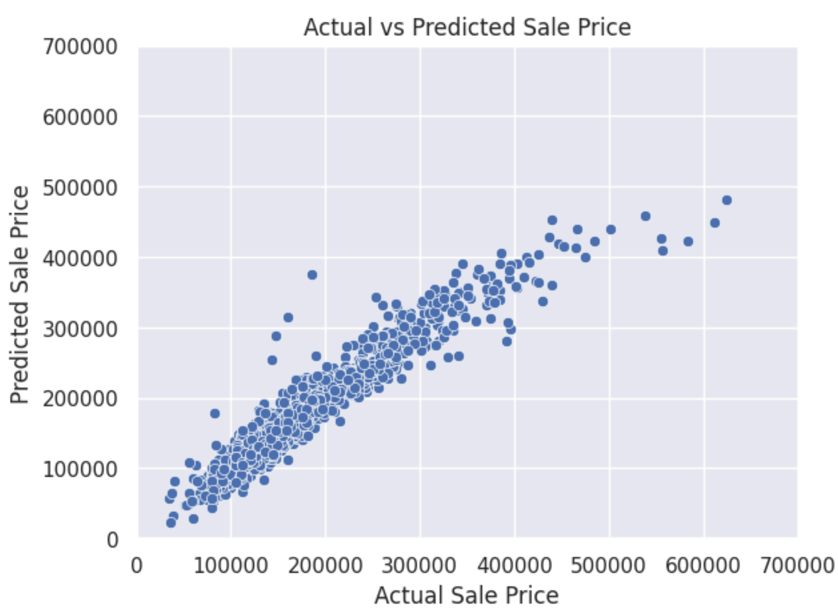
```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Create and fit model
reg7 = Lasso(alpha=100)
results = reg7.fit(X, y)

# Display coefficients with feature names
coef_df = pd.Series(results.coef_, index=X.columns)
sorted_coef = coef_df.round(2).sort_values(ascending=False)
print(sorted_coef)

# Display intercept
print(f"\nIntercept: {results.intercept_:.4f}")
```

```
LotConfig_CulDSac        6892.59
OpenSpaceRatio           6643.26
Exterior2nd_ImStucc      6530.72
LandContour_Lvl          6387.74
GarageCars               5891.99
MSSubClass_20            5375.23
MSSubClass_30            5283.02
BsmtFinSF1               5236.24
MasVnrType_None          5166.95
MasVnrArea               4345.50
BsmtFinType1_GLQ         4148.80
YearBuilt                4143.21
SaleCondition_Normal     3751.59
Exterior1st_CemntBd      3598.31
Neighborhood_Sawyer      3576.10
PoolArea                 2842.76
Fireplaces               2829.47
Foundation_PConc         2745.82
TotalBaths               2682.72
BsmtCond_TA              2631.97
FireplaceQu_NF           2437.01
MasVnrType_Stone         2408.52
LandSlope_Mod            2104.28
GarageType_Detchd        2059.32
ScreenPorch              1799.62
WoodDeckSF               1669.26
GarageType_BuiltIn       1647.07
LotArea                  1608.61
HouseStyle_1Story        1450.74
LotShape_IR2              882.30
RoofStyle_Hip             864.63
MSZoning_RM               739.83
3SsnPorch                 715.83
```

```
MSSubClass_60                 0.00
PavedDrive_Y                  0.00
SaleType_CWD                  0.00
SaleType_Con                  0.00
YrSold                       -0.00
MSSubClass_150                0.00
GarageCond_TA                 0.00
```

```
# Add predictions to the training dataset
df_train['reg7_SalePrice'] = results.predict(df_train[features])

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg7_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg7_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg7_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg7_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg7_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.xlim(0, 700000)
plt.ylim(0, 700000)
plt.show()
```

```
Correlation: 0.9528716769340638
RMSE: 24,113
MAE: 15,244
R² Score: 0.9078
```




```
# Add predictions to test dataset
df_test['reg7_SalePrice'] = results.predict(df_test[features])
```

```
# Exporting prediction into a csv
df_test[['Id', 'reg7_SalePrice']].to_csv('reg7_prediction.csv', index=False)
```

Ridge (reg8)

All numeric and categorical features, standardized, includes engineered features

Kaggle Score: 0.16387 (alpha = 1)

Kaggle Score: 0.15392 (alpha = 10)

Kaggle Score: 0.15216 (alpha = 100)

```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Create and fit Ridge model
reg8 = Ridge(alpha=100)
results = reg8.fit(X, y)

# Display coefficients with feature names
coef_df = pd.Series(results.coef_, index=X.columns)
sorted_coef = coef_df.round(2).sort_values(ascending=False)
print(sorted_coef)

# Display intercept
print(f"\nIntercept: {results.intercept_:.4f}")
```

```
Neighborhood_StoneBr      3947.74
SaleCondition_Partial     3803.35
LandContour_HLS           3650.19
Fireplaces                3627.76
MasVnrType_None           3588.25
Neighborhood_BrkSide      3534.89
AvgRmSize                 3139.81
HouseStyle_1Story         2891.09
Neighborhood_Crawfor      2720.35
WoodDeckSF                2679.99
TotalBaths                2670.32
LandContour_Lvl           2646.48
Foundation_PConc          2607.20
LandSlope_Mod             2570.18
Exterior1st_CemntBd       2520.98
ScreenPorch               2472.33
Exterior2nd_ImStucc       2355.51
LotShape_IR2              2275.91
Exterior2nd_BrkFace       2222.84
Exterior2nd_CmentBd       2197.20
MSSubClass_30             2084.03
Condition2_Norm           2080.51
GarageType_BuiltIn        2043.71
YearBuilt                 1989.21
FullBath                  1947.35
RoofStyle_Hip             1924.34
Neighborhood_Sawyer       1902.01
LotArea                   1887.98
BsmtCond_TA               1784.65
GarageArea                1481.99
Street_Pave               1470.96
PavedDrive_Y              1445.96
```

```
SaleType_CWD              594.67
BsmtFinType2_GLQ          561.17
Exterior2nd_Wd Sdng       544.45
```

```
# Add predictions to the training dataset
df_train['reg8_SalePrice'] = results.predict(df_train[features])

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg8_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg8_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg8_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg8_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg8_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.xlim(0, 700000)
plt.ylim(0, 700000)
plt.show()
```

```
Correlation: 0.9401514424786233
RMSE: 27,146
MAE: 15,961
R² Score: 0.8832
```


```
# Add predictions to test dataset
df_test['reg8_SalePrice'] = results.predict(df_test[features])
```

```
# Exporting prediction into a csv
df_test[['Id', 'reg8_SalePrice']].to_csv('reg8_prediction.csv', index=False)
```

Elastic Net (reg9)

All numeric and categorical features, standardized, includes engineered features

Kaggle Score: 0.14758 (alpha = 1, l1_ratio = 0.5)

TAKEAWAY: Elastic net uses the best features of Ridge and Lasso and provides the best prediction.

```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'
```

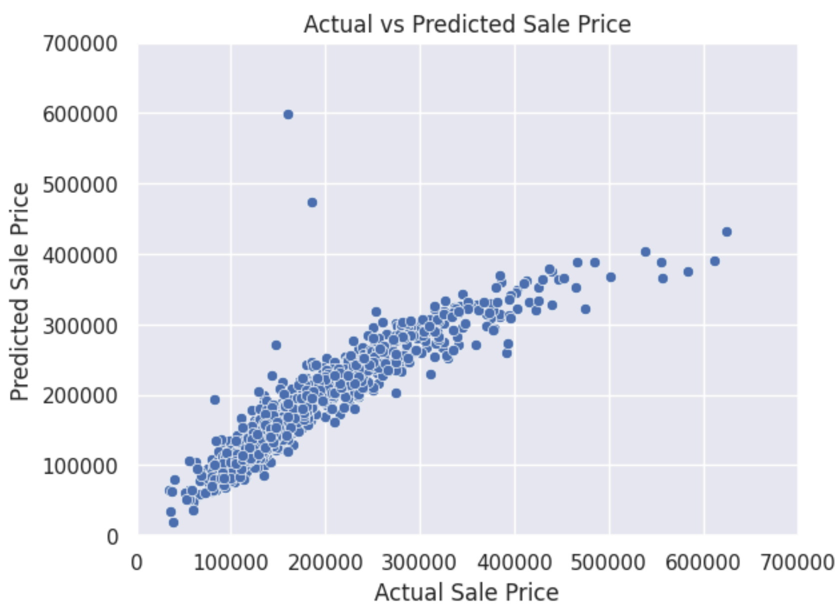
```
# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Create and fit Elastic Net model
reg9 = ElasticNet(alpha=1.0, l1_ratio=0.5)
results = reg9.fit(X, y)

# Display coefficients with feature names
coef_df = pd.Series(results.coef_, index=X.columns)
sorted_coef = coef_df.round(2).sort_values(ascending=False)
print(sorted_coef)

# Display intercept
print(f"\nIntercept: {results.intercept_:.4f}")
```

```
AvgSalePrice             10505.82
OverallQual               9950.91
GrLivArea                 6399.87
TotalLivArea              6017.38
TotRmsAbvGrd              5766.28
1stFlrSF                  4865.15
GarageCars                4818.67
MasVnrArea                4263.79
OverallCond               4193.44
TotalBsmtSF               3630.67
GarageArea                3570.11
Fireplaces                3532.19
2ndFlrSF                  3418.20
BsmtFinSF1                3227.38
AvgRmSize                 3104.24
OpenSpaceRatio            3022.08
TotalBaths                2975.14
BsmtExposure_Gd           2738.63
WoodDeckSF                2641.98
BsmtFinType1_GLQ          2369.95
ScreenPorch               2289.99
LotArea                   2157.32
Condition1_Norm           2130.09
FullBath                  2062.36
YearRemodAdd              1931.06
RoofStyle_Hip             1796.50
SaleType_New              1795.32
SaleCondition_Partial     1725.46
Neighborhood_NridgHt      1683.32
BsmtFullBath              1660.53
MSSubClass_20             1553.02
Foundation_PConc          1517.96
Neighborhood_NoRidge      1423.78
YearBuilt                 1409.75
FireplaceQu_Gd            1392.74
HalfBath                  1386.73
Functional_Typ            1366.55
LotConfig_CulDSac         1164.86
Exterior1st_BrkFace       1158.53
Neighborhood_StoneBr      1138.41
Exterior1st_CemntBd       1049.20
MasVnrType_Stone          1038.15
Exterior2nd_CmentBd        991.64
GarageType_BuiltIn         971.53
LandContour_HLS            937.27
MSZoning_RL                911.61
Neighborhood_Crawfor       903.93
OpenPorchSF                810.78
RoofMatl_WdShngl           772.41
LandSlope_Mod              712.17
3SsnPorch                  691.15
Neighborhood_BrkSide       671.86
LotShape_IR2               641.32
MSSubClass_60              587.88
HouseStyle_1Story          562.48
MasVnrType_None            530.77
Exterior2nd_BrkFace        473.08
PavedDrive_Y               425.50
```

```
# Add predictions to the training dataset
df_train['reg9_SalePrice'] = results.predict(df_train[features])
```

```
# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg9_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg9_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg9_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg9_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg9_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.xlim(0, 700000)
plt.ylim(0, 700000)
plt.show()
```

```
Correlation: 0.9186607251389116
RMSE: 31,777
MAE: 17,701
R² Score: 0.8399
```




```
# Add predictions to test dataset
df_test['reg9_SalePrice'] = results.predict(df_test[features])


# Exporting prediction into a csv
df_test[['Id', 'reg9_SalePrice']].to_csv('reg9_prediction.csv', index=False)
```

Elastic Net Hyperparameter Tuning

Best alpha: 0.0152

Best l1_ratio: 0.5

```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)
```

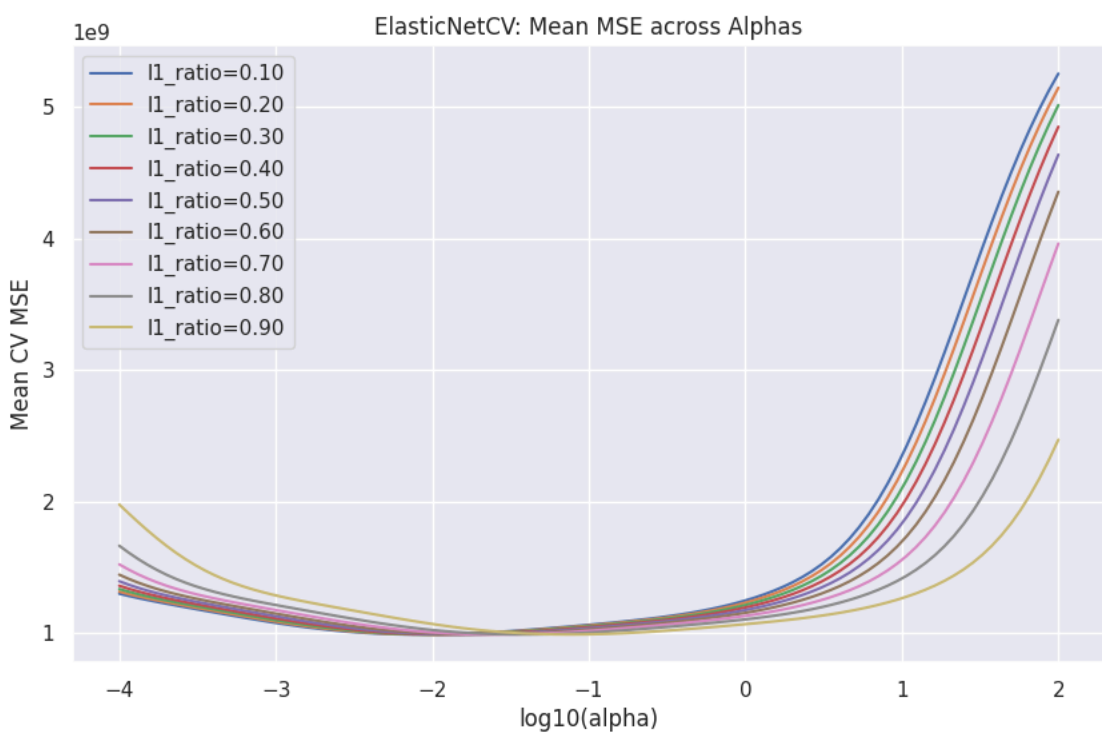
```python
# Run ElasticNetCV
reg_cv = ElasticNetCV(
    l1_ratio=np.linspace(0.1, 0.9, 9),  # Try different l1_ratios
    alphas=np.logspace(-4, 2, 100),     # Try alphas from 0.0001 to 100
    cv=5,
    max_iter=10000
    )
reg_cv.fit(X, y)
```

Show hidden output

```python
# Best parameters
print(f"Best alpha: {reg_cv.alpha_}")
print(f"Best l1_ratio: {reg_cv.l1_ratio_}")
```

```
Best alpha: 0.015199110829529332
Best l1_ratio: 0.5
```

```python
# Visualizing hyperparameter tuning
mse_path = reg_cv.mse_path_  # shape (n_l1_ratios, n_alphas, n_folds)

# Average MSE across folds
mean_mse = mse_path.mean(axis=2)  # shape (n_l1_ratios, n_alphas)

# Plot for each l1_ratio
alphas = reg_cv.alphas_
l1_ratios = reg_cv.l1_ratio if isinstance(reg_cv.l1_ratio, np.ndarray) else [reg_cv.l1_ratio]

plt.figure(figsize=(10, 6))
for i, l1_ratio in enumerate(l1_ratios):
    plt.plot(np.log10(alphas), mean_mse[i], label=f"l1_ratio={l1_ratio:.2f}")

plt.xlabel('log10(alpha)')
plt.ylabel('Mean CV MSE')
plt.title('ElasticNetCV: Mean MSE across Alphas')
plt.legend()
plt.show()
```



Elastic Net (reg10)

All numeric and categorical features, standardized, includes engineered features

Kaggle Score: 0.15413 (alpha = 0.0152, l1_ratio = 0.5)

TAKEAWAY: Model has too many noisy variables, hyper parameter tuning doesn't improve the model.

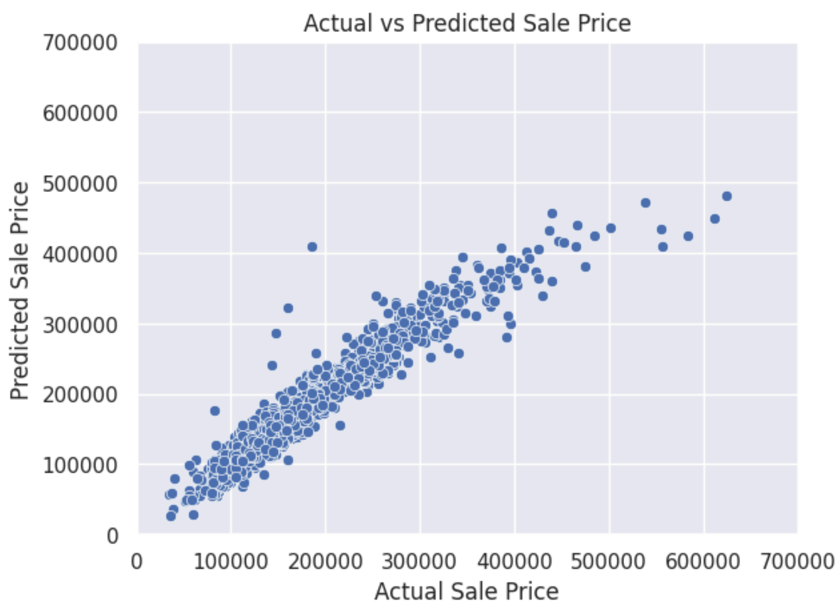
```
# Select the features and target variable
features = numeric_features + boolean_features
target = 'SalePrice'

# Create feature and target DataFrames for training
X = df_train[features].copy()
y = df_train[target]

# Convert boolean features to int (0 or 1)
for feature in boolean_features:
    X[feature] = X[feature].astype(int)

# Create and fit Elastic Net model
reg10 = ElasticNet(alpha=0.0152, l1_ratio=0.5)
results = reg10.fit(X, y)

# Display coefficients with feature names
coef_df = pd.Series(results.coef_, index=X.columns)
sorted_coef = coef_df.round(2).sort_values(ascending=False)
print(sorted_coef)

# Display intercept
print(f"\nIntercept: {results.intercept_:.4f}")
```

```
RoofMatl_WdShngl         26869.92
GrLivArea                22638.16
TotalLivArea             19223.40
2ndFlrSF                 17488.47
BsmtExposure_Gd          15132.28
Neighborhood_NoRidge     13550.24
AvgSalePrice             13186.74
Neighborhood_StoneBr     12925.35
Exterior1st_BrkFace      12337.51
OverallQual              11517.90
Functional_Typ           11483.34
1stFlrSF                 11029.33
Exterior2nd_ImStucc      10488.60
LandContour_HLS          10452.12
SaleType_New             10023.68
Condition1_Norm           9440.78
Street_Pave               9393.09
TotalBsmtSF               8990.25
Neighborhood_BrkSide      8672.88
Condition2_Norm           8361.73
LotConfig_CulDSac         7611.07
Neighborhood_Crawfor      7587.07
LandContour_Lvl           7173.35
MSSubClass_30             7051.50
OverallCond               6862.10
GarageCars                6531.41
BsmtFinSF1                6426.56
MasVnrType_None           5899.39
MSSubClass_20             5643.41
HouseStyle_1Story         5640.50
HouseStyle_1.5Unf         5623.89
OpenSpaceRatio            5488.17
MSSubClass_180            5335.80
LandSlope_Mod             5282.93
SaleCondition_Normal      5025.92
SaleCondition_Partial     4985.39
Neighborhood_Sawyer       4983.54
MasVnrType_Stone          4772.55
Neighborhood_BrDale       4756.22
Foundation_PConc          4750.70
Fireplaces                4559.93
HouseStyle_SFoyer         4489.37
MSZoning_RM               4410.38
BsmtFinType1_GLQ          4353.78
LotShape_IR2              4273.75
MasVnrArea                4149.73
GarageType_Detchd         4118.01
MSZoning_FV               3843.05
Exterior2nd_BrkFace       3811.89
YearBuilt                 3749.40
SaleType_CWD              3685.45
Condition1_RRAn           3636.91
Condition2_PosA           3602.47
MSSubClass_45             3566.26
BsmtCond_TA               3529.65
```

```
MSSubClass_40            3434.42
Neighborhood_MeadowV     3279.32
```

```
# Add predictions to the training dataset
df_train['reg10_SalePrice'] = results.predict(df_train[features])

# Calculate correlation between actual and predicted SalePrice
correlation = df_train['SalePrice'].corr(df_train['reg10_SalePrice'])
rmse = np.sqrt(mean_squared_error(df_train['SalePrice'], df_train['reg10_SalePrice']))
mae = mean_absolute_error(df_train['SalePrice'], df_train['reg10_SalePrice'])
r2 = r2_score(df_train['SalePrice'], df_train['reg10_SalePrice'])

# Print the correlation
print(f"Correlation: {correlation}")
print(f"RMSE: {rmse:,.0f}")
print(f"MAE: {mae:,.0f}")
print(f"R² Score: {r2:.4f}")

# Create a scatter plot to visualize the relationship
sns.scatterplot(x='SalePrice', y='reg10_SalePrice', data=df_train)
plt.title('Actual vs Predicted Sale Price')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.xlim(0, 700000)
plt.ylim(0, 700000)
plt.show()
```

```
Correlation: 0.9532676708487194
RMSE: 24,014
MAE: 15,025
R² Score: 0.9086
```




```
# Add predictions to test dataset
df_test['reg10_SalePrice'] = results.predict(df_test[features])
```

Introduction

Link to access this code - https://colab.research.google.com/drive/1T15VtcJFBm4mtg9WGiOuHo01eKdAly3Q?usp=sharing

Earlier versions of this code include additional EDA, reasoning for cleanup and handling missing data.

- v0 - https://colab.research.google.com/drive/1gKRrXN0jYrheIwl3eefoSEj9gh3FwewI
- v1 - https://colab.research.google.com/drive/1Cg5IznYQKiKiPJjcd3S_TltXvsYuW5Cw#scrollTo=WeDqFsFMadku
- v2 (part1) - https://colab.research.google.com/drive/1mryWq2_iZNdTU_TyRtH-5n162GsY_jpn#scrollTo=9HWrWlpE3gFQ

Data taken from - https://www.kaggle.com/c/house-prices-advanced-regression-techniques/

Import modules and data files

[ ] ↳ 5 cells hidden

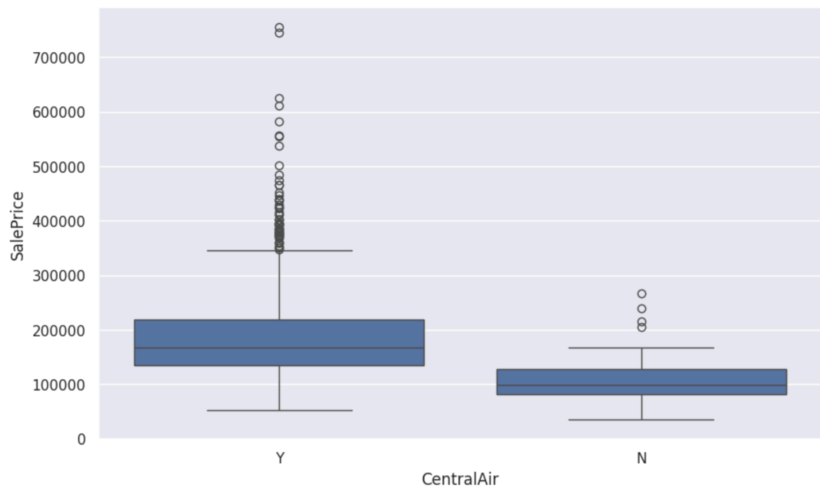
Merge and clean data

[ ] ↳ 20 cells hidden

EDA

```
# Plot SalePrice vs Central Air
plt.figure(figsize=(10, 6))
sns.boxplot(x='CentralAir', y='SalePrice', data=df_train)
```

```
<Axes: xlabel='CentralAir', ylabel='SalePrice'>
```



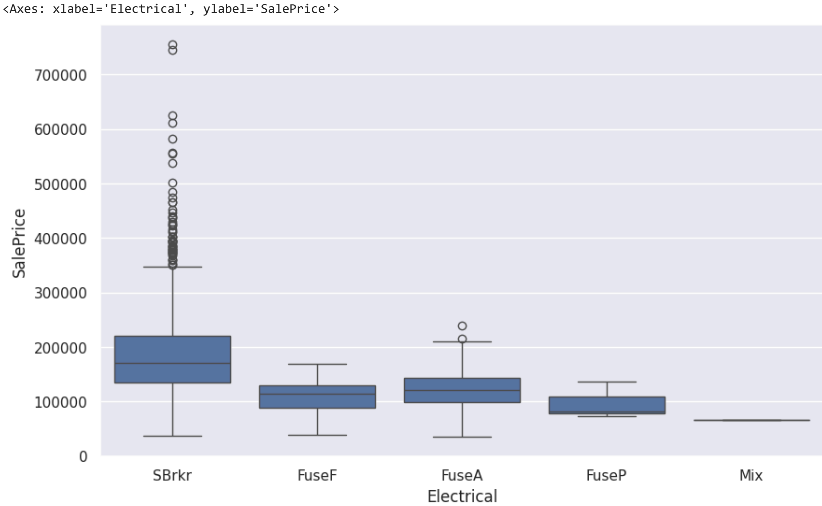

```
# Count CentralAir
df_train['CentralAir'].value_counts()
```

	count
CentralAir	
Y	1365
N	95

dtype: int64

```
# Plot SalePrice vs Electrical
plt.figure(figsize=(10, 6))
sns.boxplot(x='Electrical', y='SalePrice', data=df_train)
```

```
<Axes: xlabel='Electrical', ylabel='SalePrice'>
```



Maybe simplify this to SBrkr or not

```
df_train['Electrical'].value_counts()
```

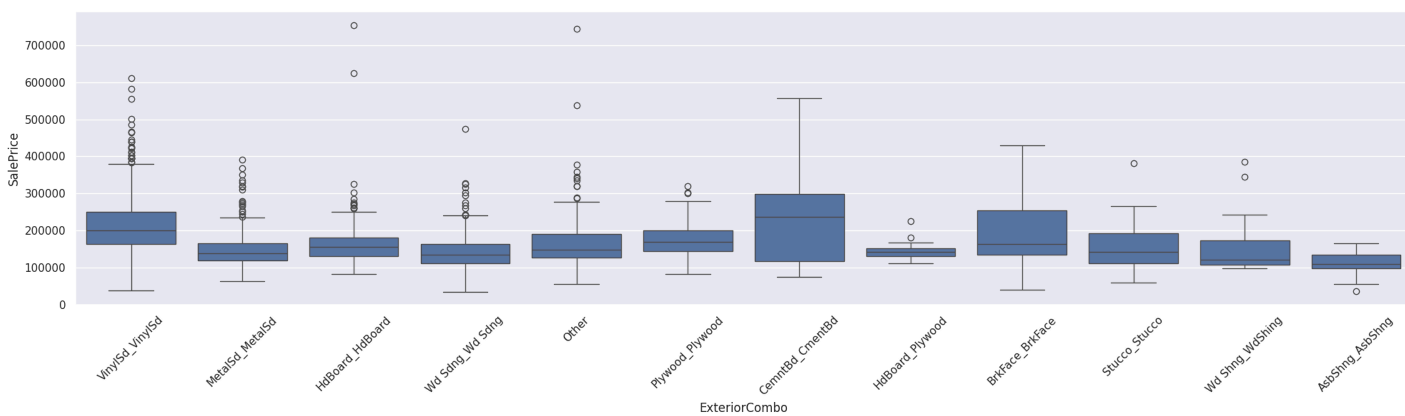
	count
Electrical	
SBrkr	1335
FuseA	94
FuseF	27
FuseP	3
Mix	1

dtype: int64

```
# Combining Exterior1st and Exterior2nd into a single variable
df_train['ExteriorCombo'] = df_train[['Exterior1st', 'Exterior2nd']].apply(lambda x: '_'.join(sorted([str(v) for v in x if pd.notnull(v)])), axis=1)

# Identify rare categories (less than 15 occurrences)
combo_counts = df_train['ExteriorCombo'].value_counts()
rare_combos = combo_counts[combo_counts < 15].index

# Replace rare categories with 'Other'
df_train['ExteriorCombo'] = df_train['ExteriorCombo'].apply(lambda x: 'Other' if x in rare_combos else x)

print(df_train['ExteriorCombo'].value_counts())
```

```
ExteriorCombo
VinylSd_VinylSd    502
MetalSd_MetalSd    212
HdBoard_HdBoard    193
Wd Sdng_Wd Sdng    177
Other              118
Plywood_Plywood     96
CemntBd_CmentBd     59
HdBoard_Plywood     25
BrkFace_BrkFace     24
Stucco_Stucco       20
Wd Shng_WdShing     17
AsbShng_AsbShng     17
Name: count, dtype: int64
```

```
# Calculate the order of ExteriorCombo based on value counts
order = df_train['ExteriorCombo'].value_counts().index

# Plot SalePrice vs ExteriorCombo with sorted x-axis
plt.figure(figsize=(20, 6))
sns.boxplot(x='ExteriorCombo', y='SalePrice', data=df_train, order=order)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Optional: makes sure labels don't get cut off
plt.show()
```



TAKEAWAY: Interestingly enought the most common combinations of exterior have the same for Exterior1st and Exterior2nd.