kaggle

PORTFOLIO

HOUSE PRICE PREDICTION

with Advanced Feature Engineering and Stacking Model

by

Aliffa Agnur

STEP 1

DATASET OVERVIEW

KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING

Submit Prediction

House Prices - Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

DATASET OVERVIEW

Columns

```
train_data.columns
```

✓ 0.0s

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

THIS DATASET CONSISTS OF A TOTAL OF 1460 ROWS AND 81 FEATURES/COLUMNS.

Library used

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

from sklearn.preprocessing import StandardScaler , PowerTransformer , RobustScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split , cross_val_score , KFold

from lazypredict.Supervised import LazyRegressor
import optuna
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.ensemble import  ExtraTreesRegressor , StackingRegressor , GradientBoostingRegressor

import lightgbm
import xgboost as xgb
import catboost
```

STEP 2

EXPLORATORY DATA ANALYSIS (EDA)

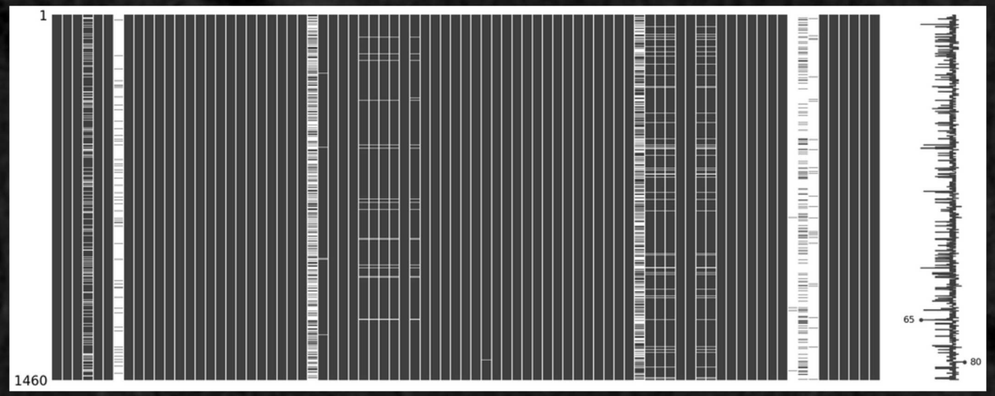
CHECK NULL VALUES

	missing	Percentage
PoolQC	1453	99.52
MiscFeature	1406	96.30
Alley	1369	93.77
Fence	1179	80.75
MasVnrType	872	59.73
FireplaceQu	690	47.26
LotFrontage	259	17.74
GarageYrBlt	81	5.55
GarageCond	81	5.55
GarageType	81	5.55
GarageFinish	81	5.55
GarageQual	81	5.55
BsmtFinType2	38	2.60
BsmtExposure	38	2.60
BsmtQual	37	2.53
BsmtCond	37	2.53
BsmtFinType1	37	2.53
MasVnrArea	8	0.55
Electrical	1	0.07
Id	0	0.00

we can see here, features like PoolQC, Alley, Fence, FireplaceQU, almost all rows are null. we remove features that have a null percentage of more than 80%

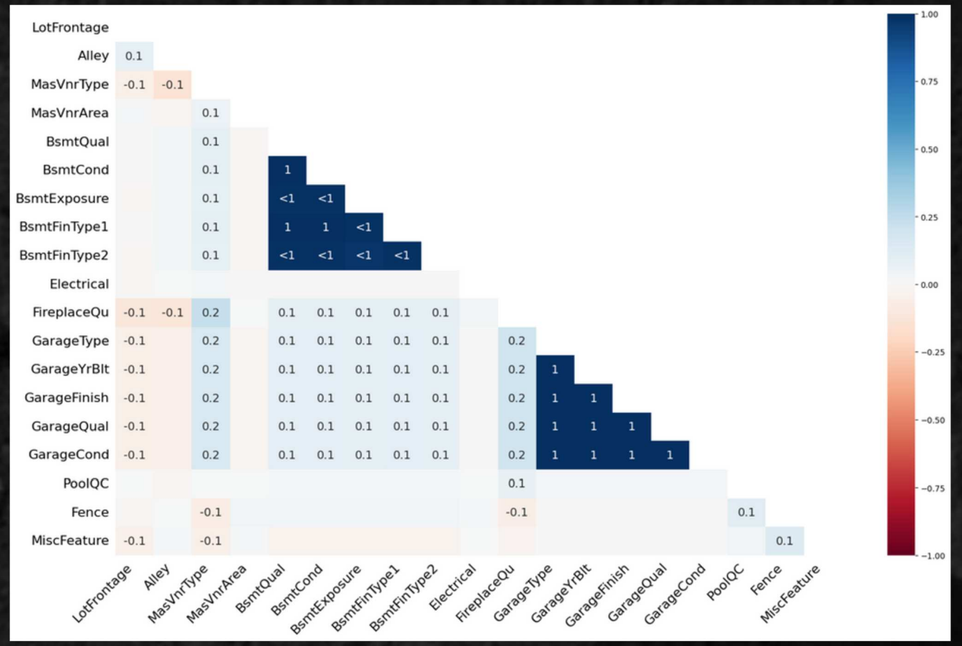
VISUALIZE MISSING VALUES

MATRIX PLOT



The white area is the null value area.

HEATMAP PLOT



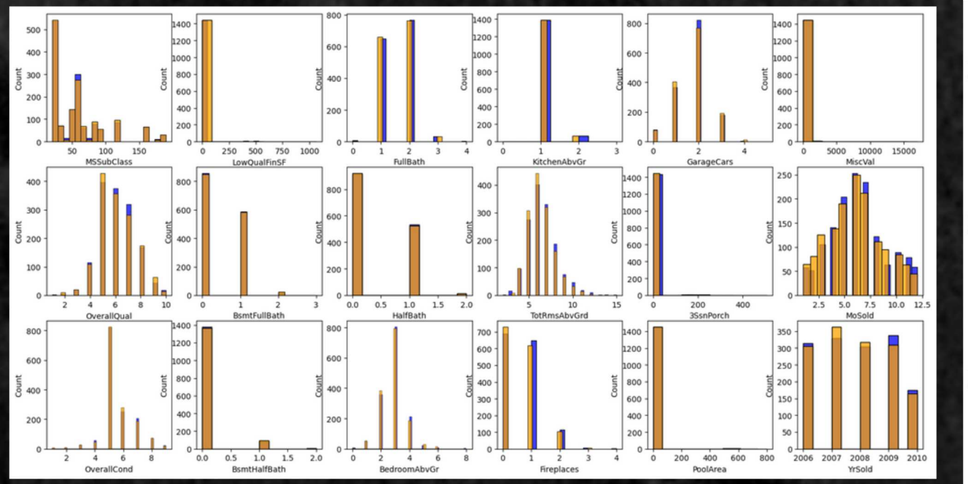
seen in the heatmap, a value close to 1 indicates that if one feature has a missing value, the other features also tend to have a missing value.

CHECK DISTRIBUTION DATA

Categorical Columns

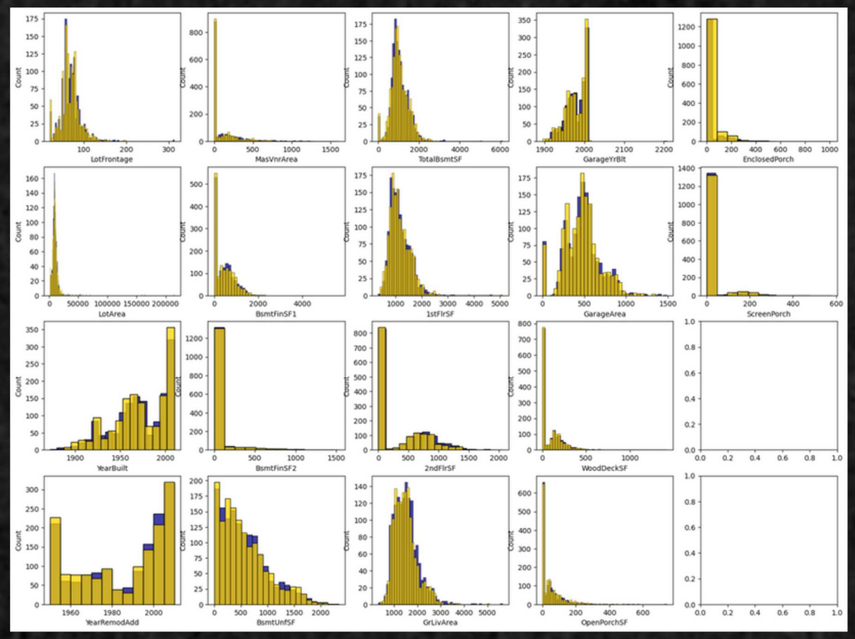
Some Feature have Dominants Label ,So we drop features that are only dominated by 1 label, such as RoofMatl, Street, GarageCond, Condition2, Utilities, Heating.

Discrete Columns



some features are dominated by 0 values, such as LowQualFinSF, MiscVal, 3SsnPorch, PoolArea. I will drop those columns

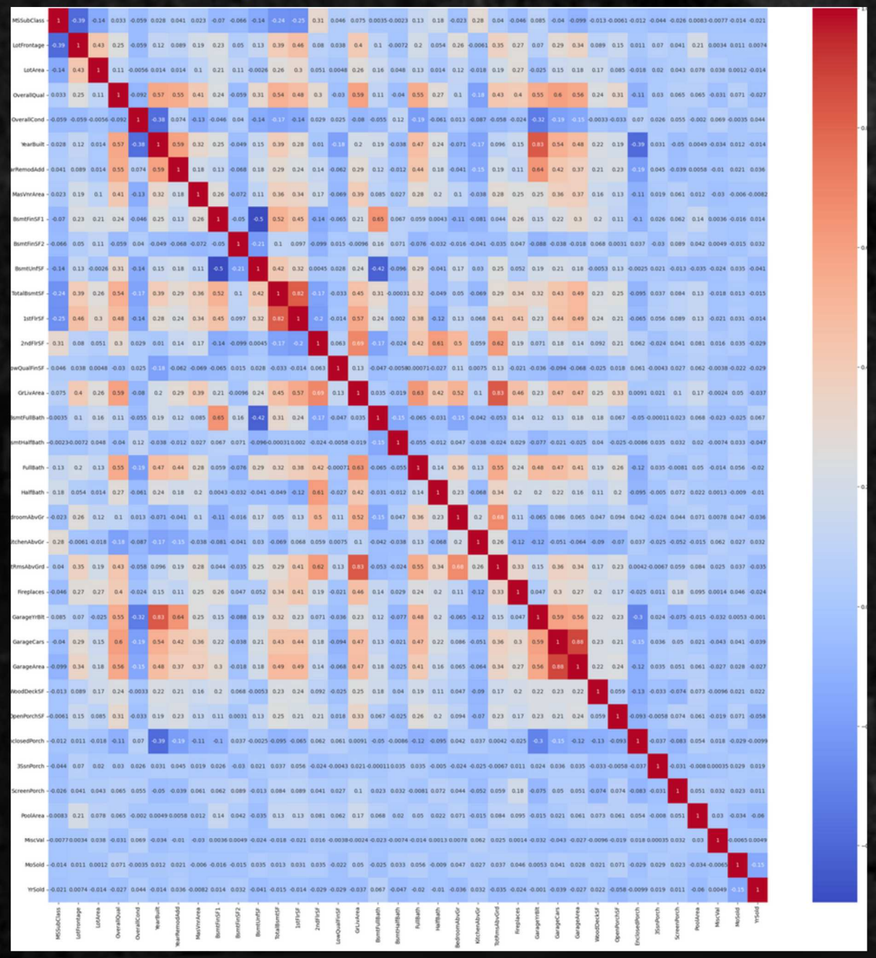
Continuous Columns



some features have a negative skewed distribution. Feature that have Skewed distribution i will do Feature Transformation

CHECK CORRELATION

Check Correlation for every Independent Variable



CONCLUSION OF CORRELATION:

1. THERE IS A STRONG CORRELATION BETWEEN:

- GarageArea and GarageCars
- 1stFlrSF and TotalBsmtSF.
- GrLivArea and TotRmsAbvGrd.

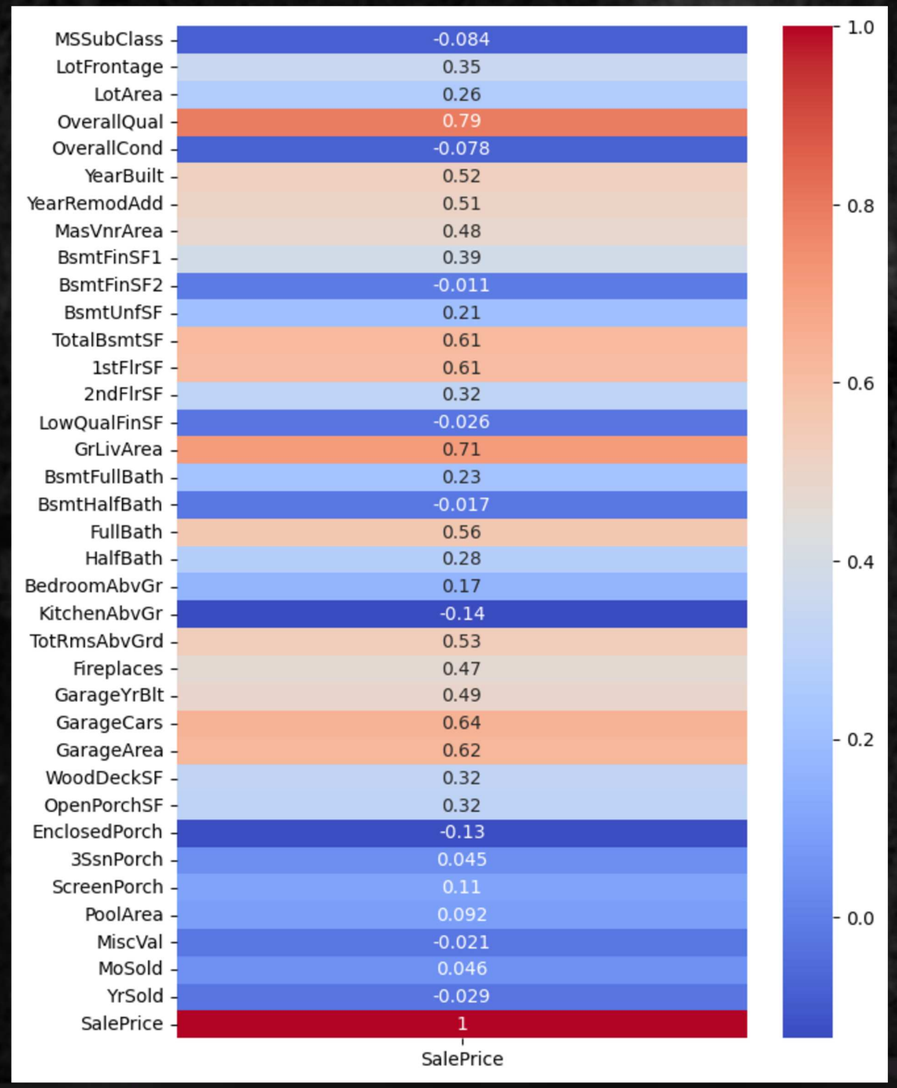
WE CAN CONSIDER REMOVING ONE OF THEM OR USING PCA TO AVOID MULTICOLINEARITY

2. CREATE NEW FEATURES SUCH AS:

- df['GarageEfficiency'] = df['GarageArea'] / df['GarageCars']
- df['LivabilityScore'] = df['GrLivArea'] + (df['FullBath'] * 2) + (df['HalfBath'])

CHECK CORRELATION

Check Correlation with Target Feature (SalePrice)



1. HIGHLY CORRELATED VARIABLES WITH SALEPRICE(POSITIVE, >0.5):

- OVERALLQUAL (0.79)
- GRLIVAREA (0.71)
- GARAGECARS (0.64)
- 1STFLRSF (0.61)
- TOTALBSMTSF (0.61)

THESE VARIABLES ARE STRONG CANDIDATES FOR INCLUSION IN THE MODEL BECAUSE THEY PROVIDE SIGNIFICANT INFORMATION TO PREDICT HOME PRICE.

2. MODERATE CORRELATIONS WITH SALEPRICE(0.3-0.5):

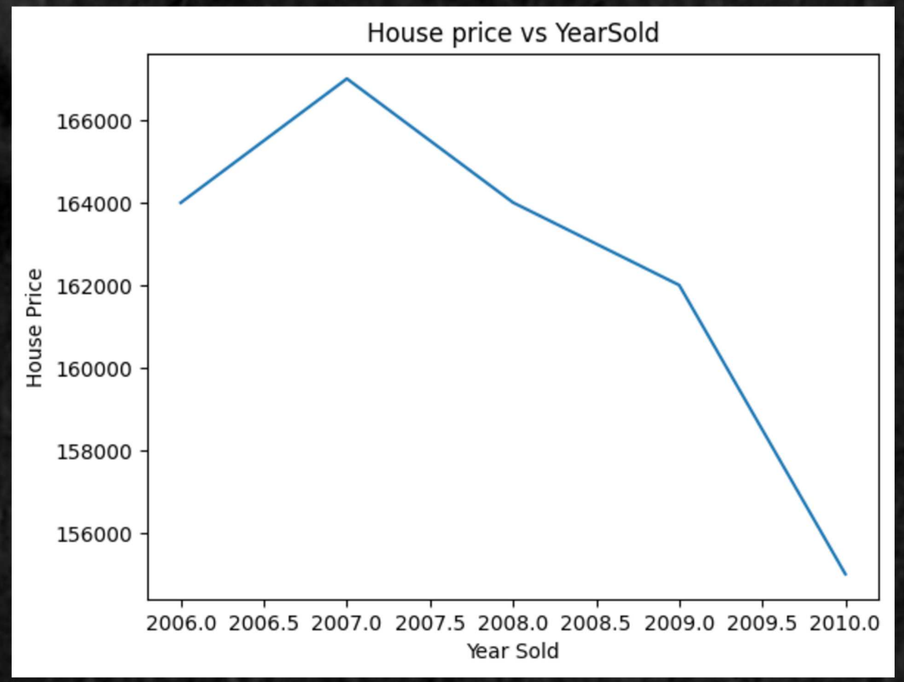
- YEARBUILT (0.52)
- YEARREMODADD (0.51)
- FULLBATH (0.56)
- FIREPLACES (0.47)
- TOTRMSABVGRD (0.53)
- MASVNRAREA (0.48)

THESE VARIABLES PROVIDE ADDITIONAL INFORMATION THAT CAN HELP THE MODEL IMPROVE PREDICTION ACCURACY, BUT ARE NOT AS STRONG AS THE HIGHLY CORRELATED VARIABLES.

3. create new feature TotalArea = GrLivArea + TotalBsmtSF . Adds the area of living space and basement.

CHECK CORRELATION

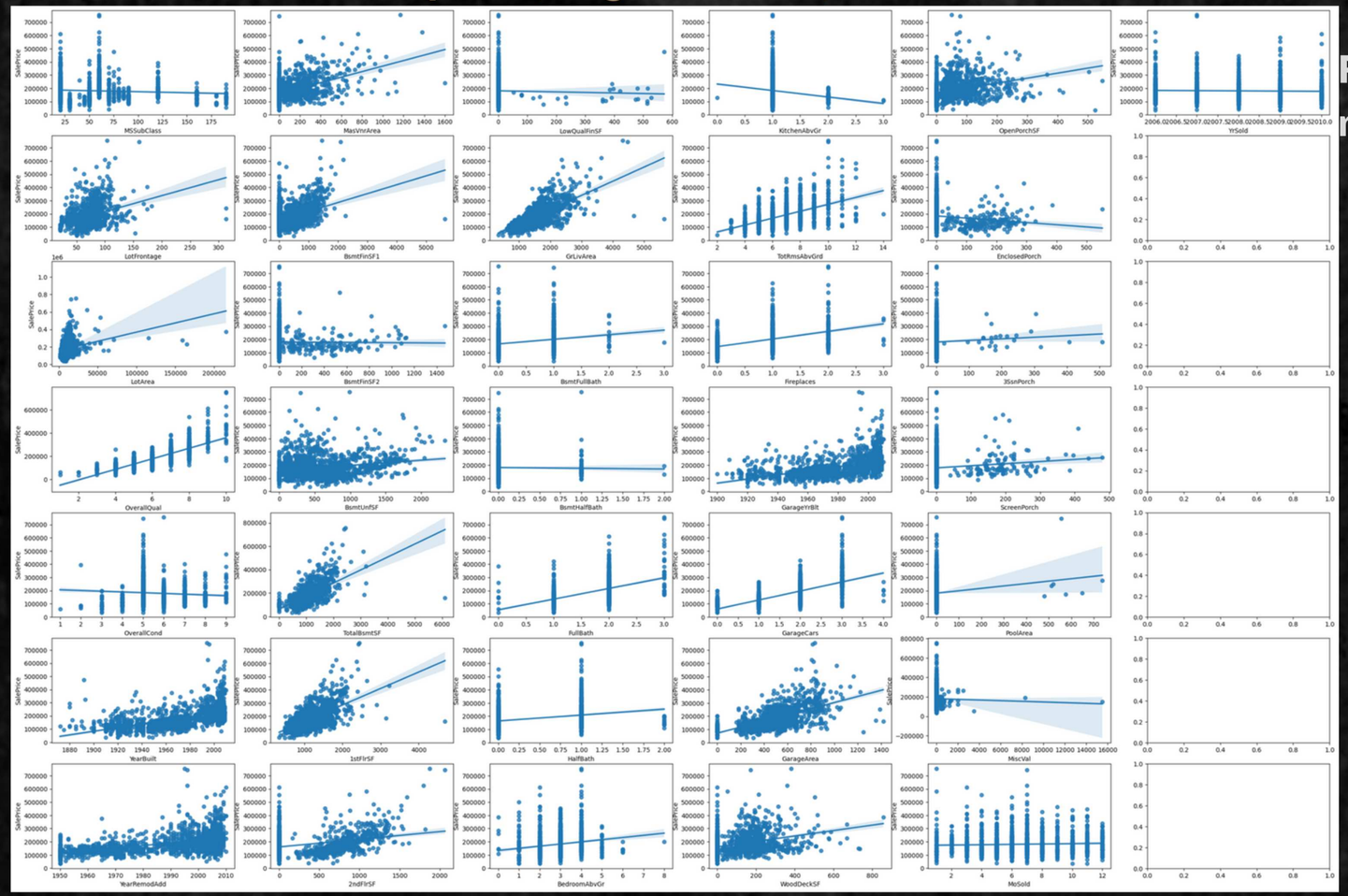
Check Relation between House Price vs YearSold



There is a negative correlation between YrSold and SalePrice. This means that YrSold has an effect on SalePrice.

CHECK LINEARITY

check linearity of continuous variables against sale price (target feature)

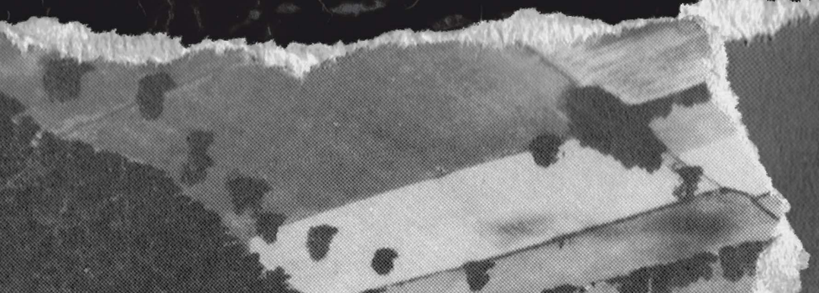


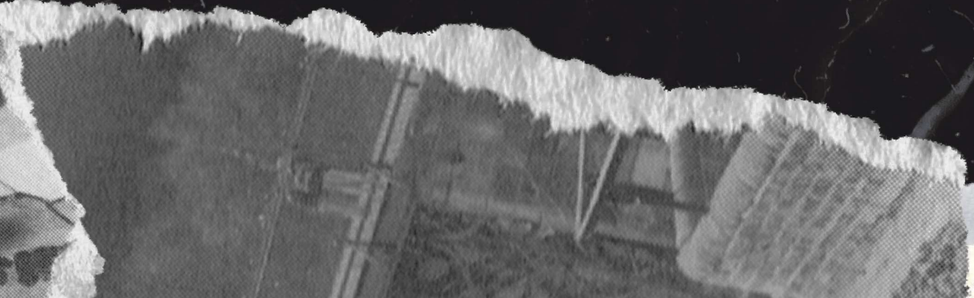
From the data above, there are non-linear relationships such as:

- GrLivArea and SalePrice (n log(n) pattern),
- BsmtFinSF1 and SalePrice (exponential pattern),
- TotalBsmtSF and SalePrice (exponential pattern),
- LotFrontage and SalePrice (logarithm pattern),
- LotArea and SalePrice (logarithm pattern),
- GarageArea and SalePrice (logarithm pattern)

STEP 3

FEATURE ENGINEERING





y and fell in love with it.

DROP IRRELEVANT COLUMNS

```
# FIRST DROP COLUMNS WITH MANY NULL VALUES
cols_with_many_null = ['PoolQC', 'Alley', 'Fence', 'FireplaceQu', 'LotFrontage']
combined_data.drop(labels=cols_with_many_null, axis=1, inplace=True)

# NEXT DROP COLUMNS WITH MANY ZERO VALUES
cols_with_many_zero = ['LowQualFinSF', 'MiscVal', '3SsnPorch', 'PoolArea']
combined_data.drop(labels=cols_with_many_zero, axis=1, inplace=True)

# LAST DROP COLUMNS WITH DOMINANT 1 LABEL
cols_with_dominant_label = ['Id','RoofMatl', 'Street', 'Condition2', 'Utilities', 'Heating']
combined_data.drop(labels=cols_with_dominant_label, axis=1, inplace=True)

combined_data.columns  , len(combined_data.columns)   # CHECK FINAL COLUMNS AFTER DROPPED
```

```
(Index(['MSSubClass', 'MSZoning', 'LotArea', 'LotShape', 'LandContour',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC', 'CentralAir',
       'Electrical', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', 'ScreenPorch', 'MiscFeature', 'MoSold', 'YrSold',
       'SaleType', 'SaleCondition', 'SalePrice'],
      dtype='object'),
 66)
```

THE COLUMNS/FEATURE WE DELETED :

1. Columns with many null values
2. Columns with many zero values
3. Columns with Dominant 1 Label

FILL MISSING VALUES

```
# FILL NUMERICAL MISSING VALUES WITH ZERO VALUES
numerical_feature = combined_data.select_dtypes(include=['int64', 'float64']).columns.tolist()
for feature in numerical_feature:
    combined_data[feature] = combined_data[feature].fillna(0)

# FILL CONTINUOUS MISSING VALUES
dropped_cols = cols_with_dominant_label + cols_with_many_null
mode_feature = ['Electrical', 'MsZoning', 'Functional', 'KitchenQual', 'Exterior2nd', 'Exterior1st', 'Utilities', 'SaleType']


for feature in categorical_feature:
    if feature not in dropped_cols:
        if feature not in mode_feature:
            combined_data[feature] = combined_data[feature].fillna('Unknown')
        else:
            combined_data[feature] = combined_data[feature].fillna(combined_data[feature].mode()[0])
```

I FILLED IN THE MISSING VALUES IN SEVERAL WAYS

for continuous columns, I fill missing values with the value 0

for categorical columns, if there are many missing values, then I will fill it with the label 'Unknown'.

And if there are few missing values, I will fill it with the mode label.

CREATE NEW FEATURE

```
# ADD NEW FEATURE 'GarageEfficiency'
combined_data['GarageEfficiency'] = combined_data['GarageArea'] / (combined_data['GarageCars'] + 1)   # +1 TO AVOID DIVISION BY ZERO

# ADD NEW FEATURE
#combined_data['LivabilityScore'] = combined_data['GrLivArea'] + (combined_data['FullBath'] * 2) + (combined_data['HalfBath'])

# ADD NEW FEATURE TotalArea
combined_data['TotalArea'] = combined_data['GrLivArea'] + combined_data['TotalBsmtSF']
```

we created 2 new features, the **GarageEfficiency** feature and the **TotalArea** feature.

OVERCOME MULTICOLINEARITY USING PCA

```python
# USING PCA TO ADDRESSING MULTICOLINEARITY


zscore = StandardScaler()

# NORMALIZATION
multi_cols_1 = combined_data[['GarageArea','GarageCars']]   # COLUMN THAT HAS MULTICOLINEARITY
multi_cols_1 = zscore.fit_transform(multi_cols_1)

# FIND PRINCIPAL COMPONENT OPTIMAL
pca_1  = PCA(n_components=None)
pca_1.fit_transform(multi_cols_1)


# NORMALIZATION
multi_cols_2 = combined_data[['1stFlrSF', 'TotalBsmtSF']]
multi_cols_2 = zscore.fit_transform(multi_cols_2)

# FIND PRINCIPAL COMPONENT OPTIMAL
pca_2  = PCA(n_components=None)
pca_2.fit_transform(multi_cols_2)


# NORMALIZATION
multi_cols_3 = combined_data[['GrLivArea', 'TotRmsAbvGrd']]
multi_cols_3 = zscore.fit_transform(multi_cols_3)

# FIND PRINCIPAL COMPONENT OPTIMAL
pca_3  = PCA(n_components=None)
pca_3.fit_transform(multi_cols_3)

for pca in [pca_1, pca_2, pca_3]:
    print(f'Number of Components : {pca.n_components_}')
    print(f'Ratio every Component / PC : {pca.explained_variance_ratio_}\n')
    print(f'PCA Components : \n{pca.components_}\n')
    print('---------------------------------')
```

OUTPUT :

```
Number of Components : 2
Ratio every Component / PC : [0.94494511 0.05505489]

PCA Components :
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]

---------------------------------
Number of Components : 2
Ratio every Component / PC : [0.90068794 0.09931206]

PCA Components :
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]

---------------------------------
Number of Components : 2
Ratio every Component / PC : [0.90417721 0.09582279]

PCA Components :
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]

---------------------------------
```

PC1 explains up to 90%, so we only take PC1

Take only pc1

```python
# CHOOSE PC1

pca_1  = PCA(n_components=1)
multi_cols_1 = pca_1.fit_transform(multi_cols_1)

pca_2  = PCA(n_components=1)
multi_cols_2 = pca_2.fit_transform(multi_cols_2)

pca_3  = PCA(n_components=1)
multi_cols_3 = pca_3.fit_transform(multi_cols_3)

# ADD PC1 TO DATAFRAME
combined_data['multi_cols_1'] = multi_cols_1.ravel()
combined_data['multi_cols_2'] = multi_cols_2.ravel()
combined_data['multi_cols_3'] = multi_cols_3.ravel()
```

FEATURE TRANSFORMATION

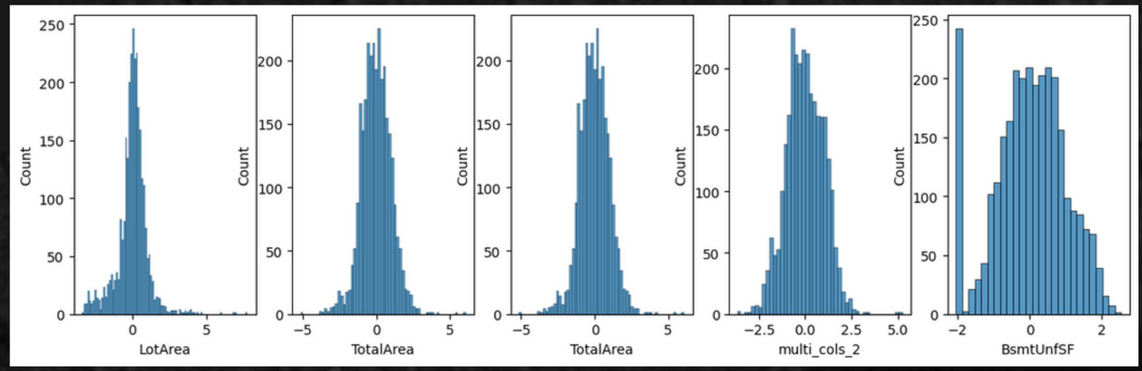
use Yeo-Johnson to change the data distribution to normal distribution

```
# APPLY POWER TRANSFORMATION TO FEATURE THAT SKEWNESS VALUE > 1 OR < -1

# SELECT ALL NUMERICAL DATA
transform_data      = combined_data.select_dtypes(include=['int64', 'float64'])
continuous_feature  = [col for col in transform_data if len(transform_data[col].unique()) > 25]    # SELECT ONLY CONTINUOUS FEATURE
transform_data = combined_data[continuous_feature]
transform_data.drop(labels=['SalePrice'], axis=1 , inplace=True)

# CHECK SKEWNESS
skewness = transform_data.skew().sort_values(ascending=False)

# APPLY TRANSFORM FOR FEATURE HAVE SKEWNESS > 1 OR < -1
# cols_to_transform = skewness[(skewness > 1) | (skewness < -1)].index
cols_to_transform = ['LotArea', 'TotalArea', 'TotalArea', 'multi_cols_2', 'BsmtUnfSF']  # CHOSEN FEATURE TO TRANSFORM

for col in cols_to_transform:
    yeo_johnson = PowerTransformer(method='yeo-johnson', standardize=True, copy=True)
    combined_data[[col]] = yeo_johnson.fit_transform(combined_data[[col]])
```

Feature that we transform :

['LotArea', 'TotalArea', 'TotalArea', 'multi_cols_2', 'BsmtUnfSF']

After Transformation



One Hot Encoder

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

combined_data
```

convert all category features to numeric format using one hot encoder

FEATURE SCALING

```
# NORMALIZATION

# CHOOSE COLUMNS TO NORMALIZE
cols_to_robust = ['MSSubClass','YearRemodAdd', '2ndFlrSF','BedroomAbvGr','OpenPorchSF','MasVnrArea','EnclosedPorch','BsmtFinSF1','ScreenPorch','BsmtFinSF2','GarageYrBlt','YearBuilt','WoodDeckSF']
cols_to_zscore = ['GarageEfficiency','LotArea','TotalArea','OverallQual','multi_cols_1','OverallCond','multi_cols_2','BsmtUnfSF','multi_cols_3']

# ROBUST SCALING NORM
robust = RobustScaler()
robust.fit(x_train[cols_to_robust])

x_train[cols_to_robust] = robust.transform(x_train[cols_to_robust])
x_test[cols_to_robust]  = robust.transform(x_test[cols_to_robust])

# ZSCORE NORM
zscore = StandardScaler()
zscore.fit(x_train[cols_to_zscore])

x_train[cols_to_zscore] = zscore.transform(x_train[cols_to_zscore])
x_test[cols_to_zscore]  = zscore.transform(x_test[cols_to_zscore])
```

We will perform **robust scaling normalization** for features that have many outliers and are not normally distributed. and use **Z-score normalization** for features that are almost normally distributed.

STEP 4

MODEL DEVELOPMENT

LAZY PREDICT

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
LassoLarsIC	0.11	0.88	0.14	0.07
TransformedTargetRegressor	0.10	0.88	0.14	0.03
LinearRegression	0.10	0.88	0.14	0.03
Ridge	0.10	0.88	0.14	0.03
HistGradientBoostingRegressor	0.10	0.88	0.14	1.29
BayesianRidge	0.10	0.88	0.14	0.04
RidgeCV	0.10	0.88	0.14	0.05
LassoLarsCV	0.10	0.88	0.14	0.12
LassoCV	0.10	0.88	0.14	0.21
ElasticNetCV	0.10	0.88	0.14	0.23
LGBMRegressor	0.09	0.88	0.14	0.36
NuSVR	0.09	0.88	0.14	0.50
LinearSVR	0.07	0.88	0.14	0.19
GradientBoostingRegressor	0.07	0.88	0.14	1.33
SGDRegressor	0.06	0.88	0.14	0.03
PoissonRegressor	0.06	0.88	0.14	0.03
HuberRegressor	0.06	0.88	0.14	0.10
ExtraTreesRegressor	0.05	0.88	0.14	2.28

we can see the best algorithm. Ridge and Linear Regression are better compare to each other. we can use them for Stacking-Models.

so, i want use Linear Model + Non-Linear Model Combination for Stacking Model .

Here my Combination :

- Ridge Regression (Linear Model)
- Linear Regression (Linear Model)
- Lasso Regression (Linear Model)
- Gradient Boosting Machine (Non-Linear Model)
- XGBoost (Non-Linear Model)
- LightGBM (Non-Linear Model)
- CatBoost (Non-Linear Model)

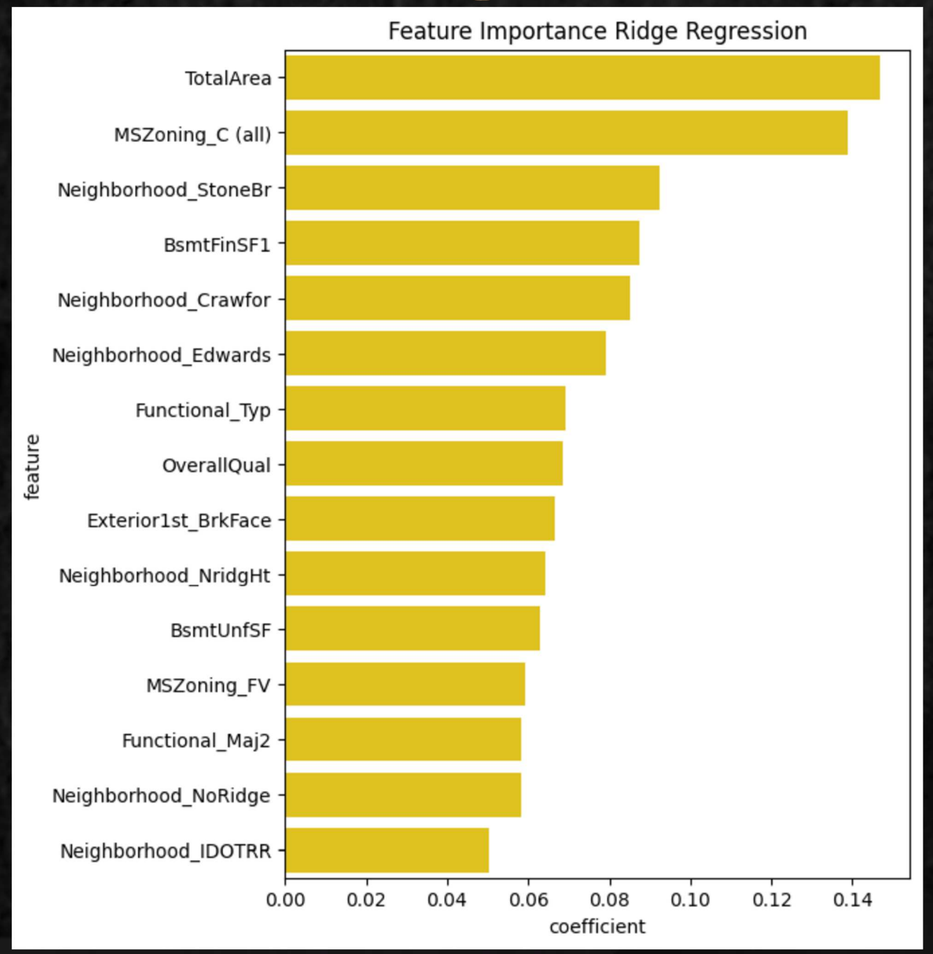
RIDGE REGRESSION

Train model

```
# BUILD A MODEL

ridge_best_params = {'alpha': 10.907628470160567, 'max_iter': 685}

ridge = Ridge(fit_intercept=True, solver='auto', random_state=12, **ridge_best_params)
ridge.fit(x_train, y_train)
```

```
Ridge
Ridge(alpha=10.907628470160567, max_iter=685, random_state=12)
```

Feature Importances



LASSO REGRESSION

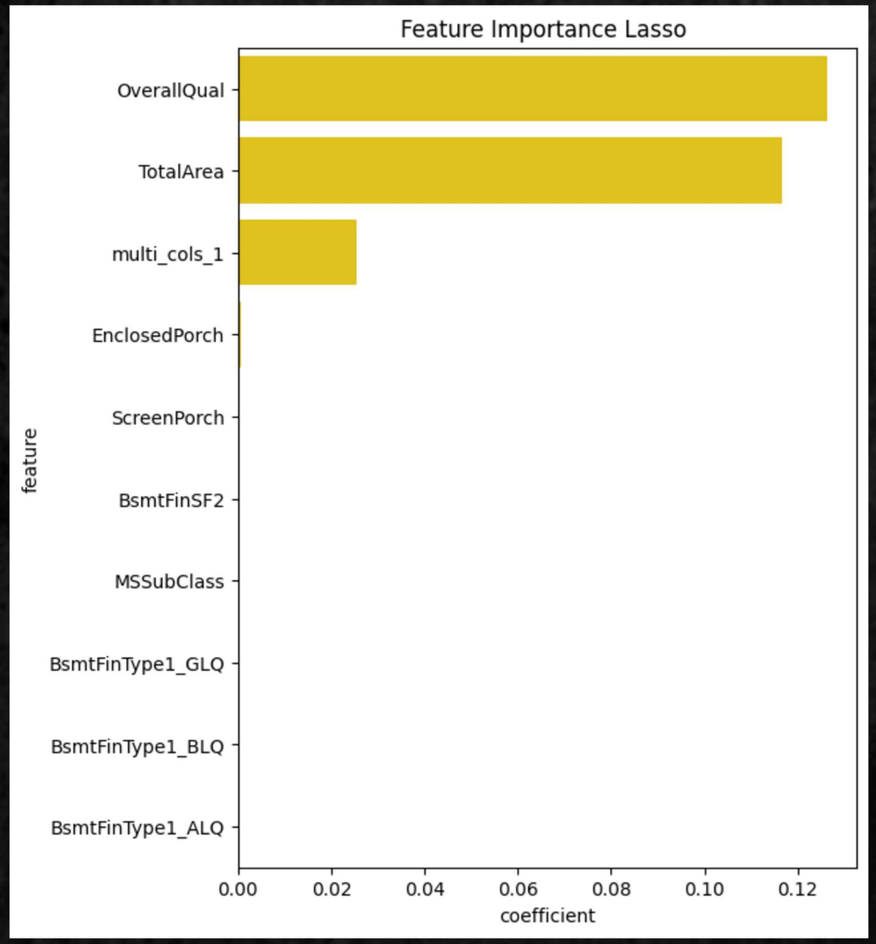
Train model

```
lasso_best_params = {'alpha': 0.10064802295354133, 'max_iter': 1839}

# BUILD A MODEL
lasso = Lasso(fit_intercept=True, random_state=12, **lasso_best_params)
lasso.fit(x_train, y_train)
```

✓ 0.0s

```
Lasso
Lasso(alpha=0.10064802295354133, max_iter=1839, random_state=12)
```

Feature Importances



almost all of its features have zero coefficient

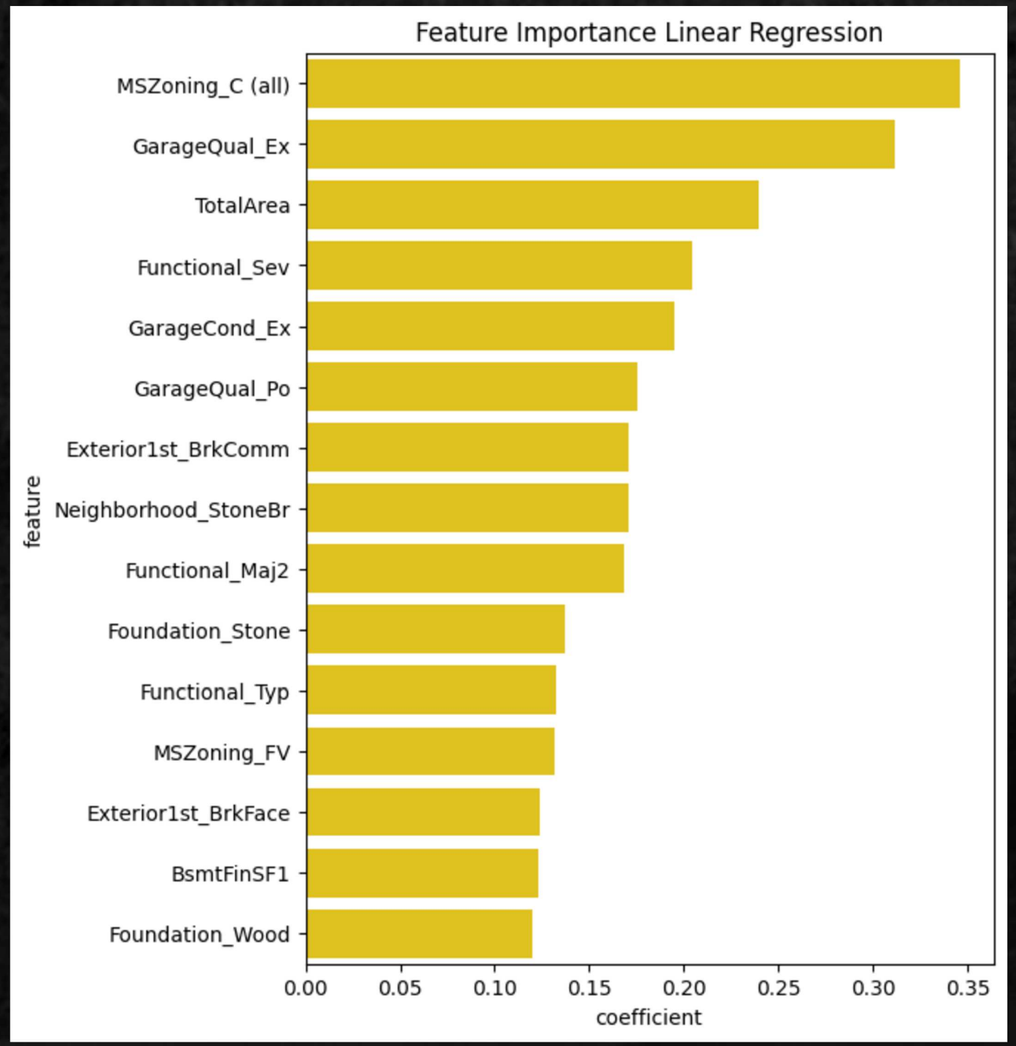
LINEAR REGRESSION

Train model

```
linear = LinearRegression(fit_intercept=True, positive=False)
linear.fit(x_train, y_train)
```

```
LinearRegression()
```

Feature Importances



GRADIENT BOOSTING MACHINE

Train model

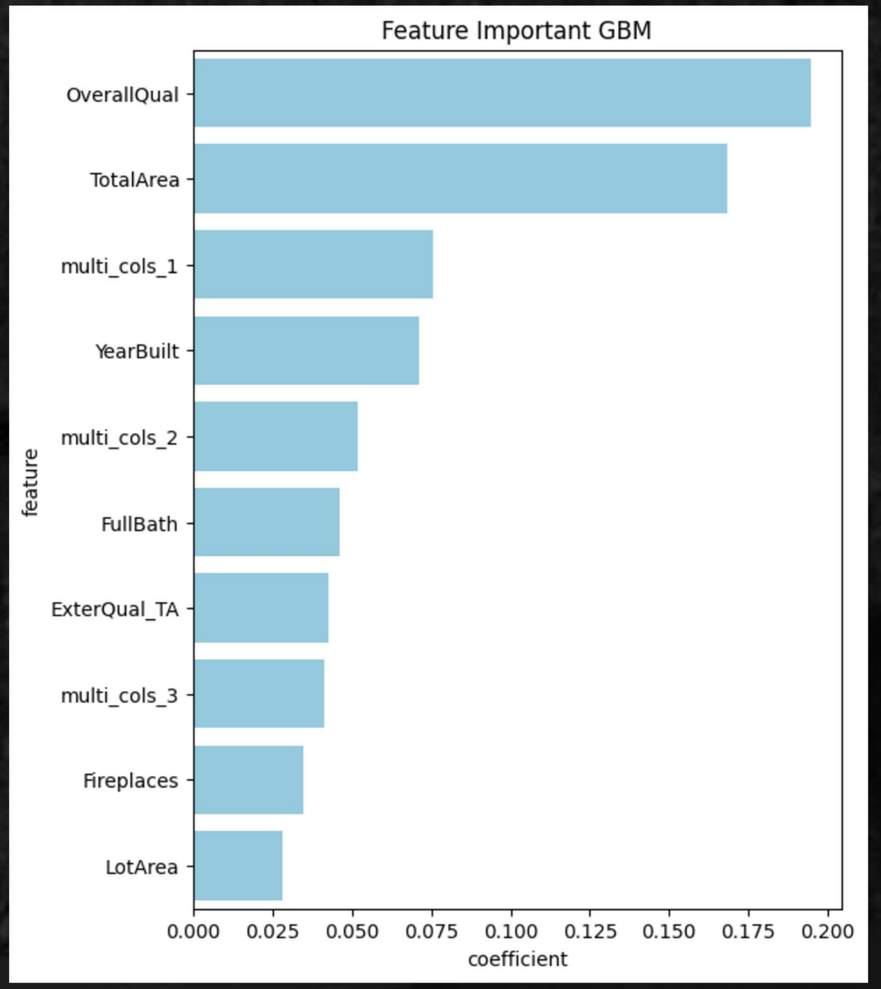
```
# FIT MODEL GBM

gbr_best_params = {'n_estimators': 1552, 'learning_rate': 0.03236403825303992, 'max_depth': 3,
                   'min_samples_split': 14, 'max_features': 46}
gbm = GradientBoostingRegressor(random_state=12, **gbr_best_params)
gbm.fit(x_train,y_train)
```

✓ 4.9s

GradientBoostingRegressor

```
GradientBoostingRegressor(learning_rate=0.03236403825303992, max_features=46,
                          min_samples_split=14, n_estimators=1552,
                          random_state=12)
```

Feature Importances

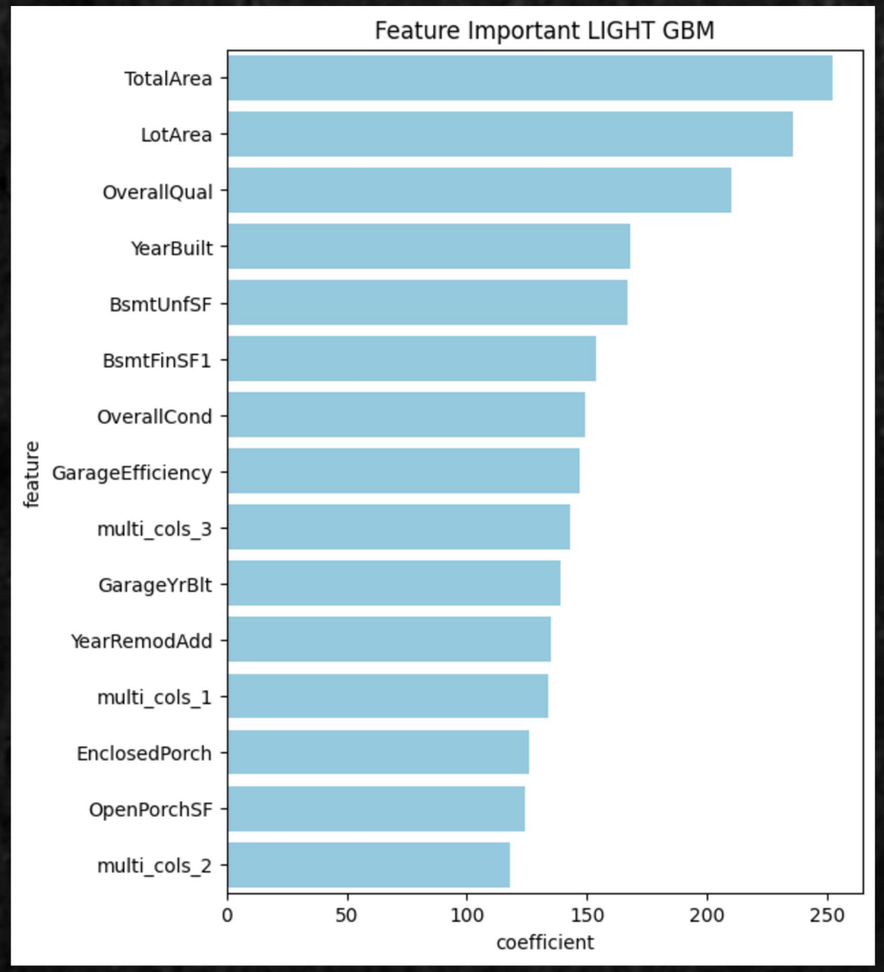


LIGHT GBM

Train model

```
# BUILD AND FIT MODEL WITH BEST HYPERPARAMETERS

lgbm_best_params = {'num_leaves': 92,
                    'max_depth': 2,
                    'learning_rate': 0.059514026241983445,
                    'n_estimators': 1524,
                    'min_child_weight': 3,
                    'min_child_samples': 17,
                    'subsample': 0.48984666486323647,
                    'reg_alpha': 0.033788563133902494,
                    'reg_lambda': 4.040907598613931}

lgbm = lightgbm.LGBMRegressor(random_state=12, **lgbm_best_params)
lgbm.fit(x_train, y_train)
```

Feature Importances



XGBOOST

Train model

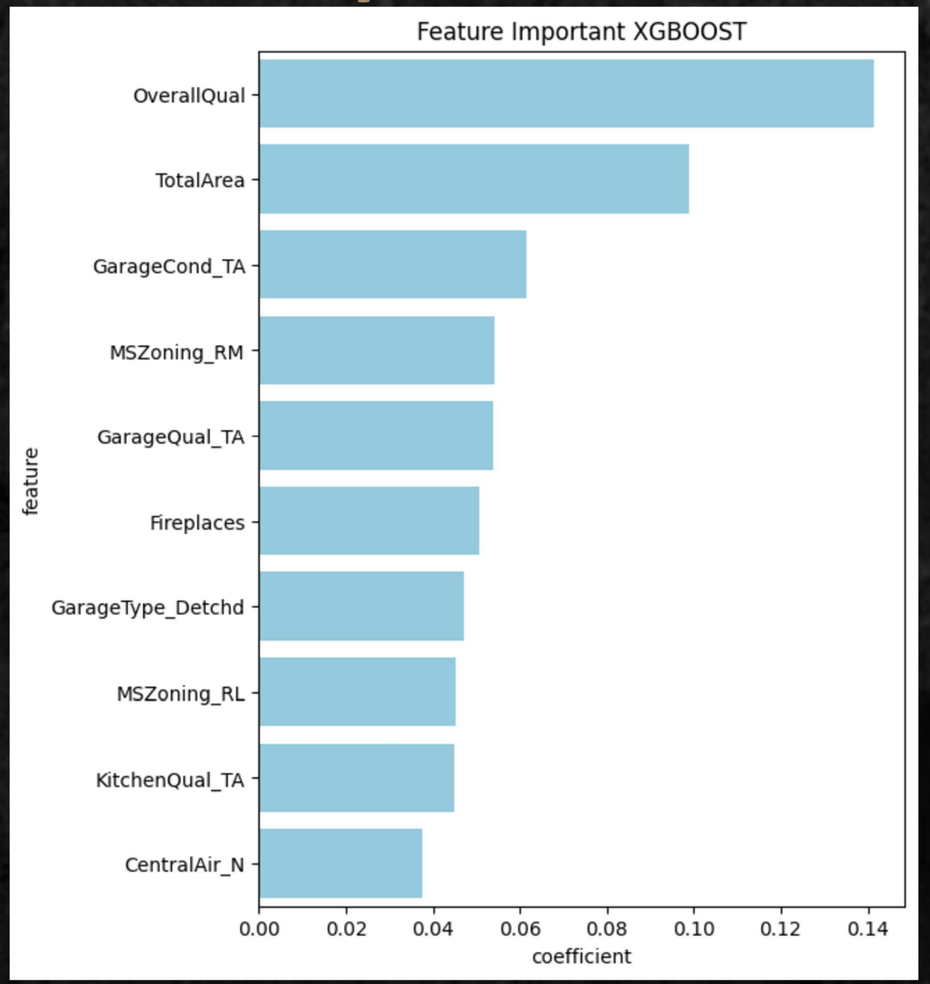
```
xgb_best_params = {'n_estimators': 1552,
                   'learning_rate': 0.03236403825303992,
                   'max_depth': 3,
                   'min_samples_split': 14,
                   'max_features': 46}

xgboost = xgb.XGBRegressor(random_state=12, **xgb_best_params)
xgboost.fit(x_train, y_train)
```

✓ 2.2s

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.03236403825303992,
             max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_features=46, max_leaves=None,
             min_child_weight=None, min_samples_split=14, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=1552,
             n_jobs=None, ...)
```

Feature Importances

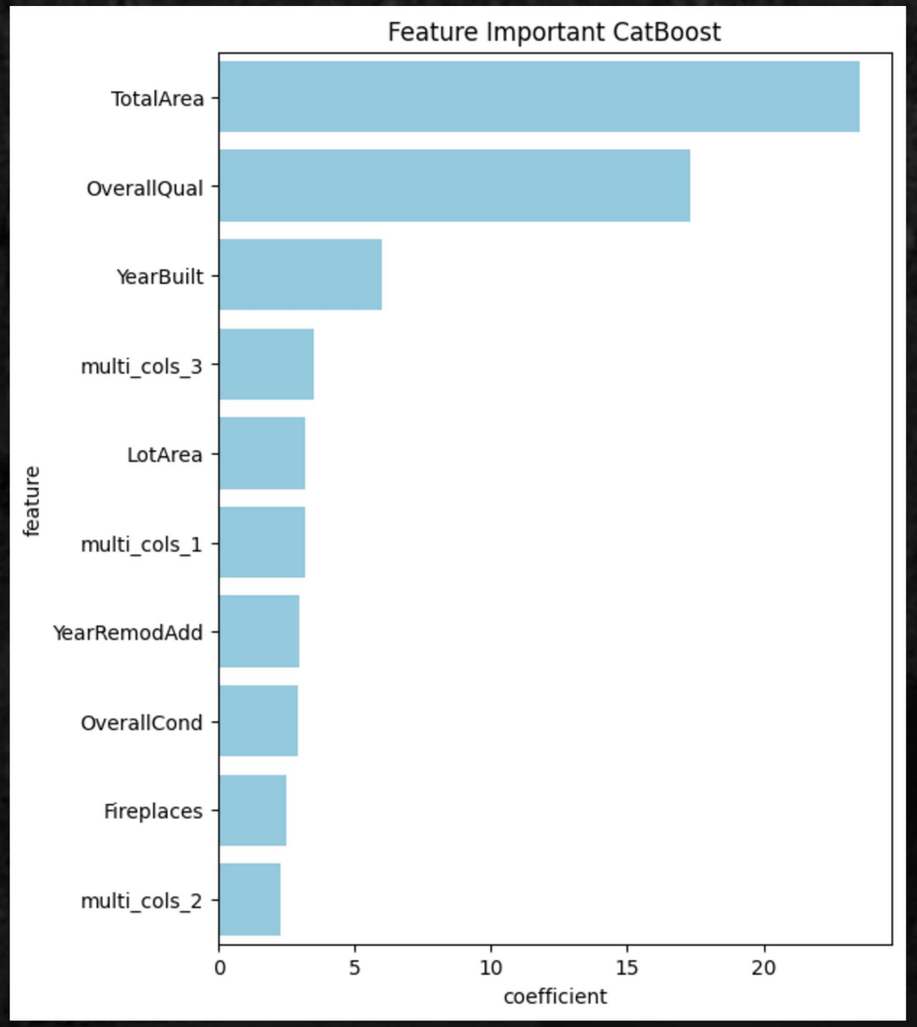


CATBOOST

Train model

```
catboost_best_params = {'iterations': 940,
                        'learning_rate': 0.05397757917685036,
                        'depth': 4,
                        'l2_leaf_reg': 9.88267503317357e-05,
                        'subsample': 0.9616596331860532,
                        'colsample_bylevel': 0.6942581356179441,
                        'min_data_in_leaf': 61,
                        'max_bin': 197}

cat = catboost.CatBoostRegressor(random_state=12, devices=0,**catboost_best_params)
cat.fit(x_train, y_train)
```

Feature Importances



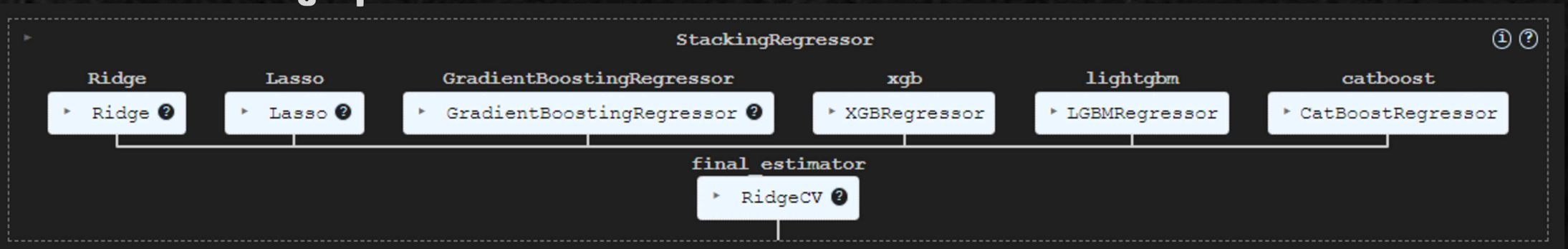
STACKING REGRESSOR

merge all created models into stacking model.

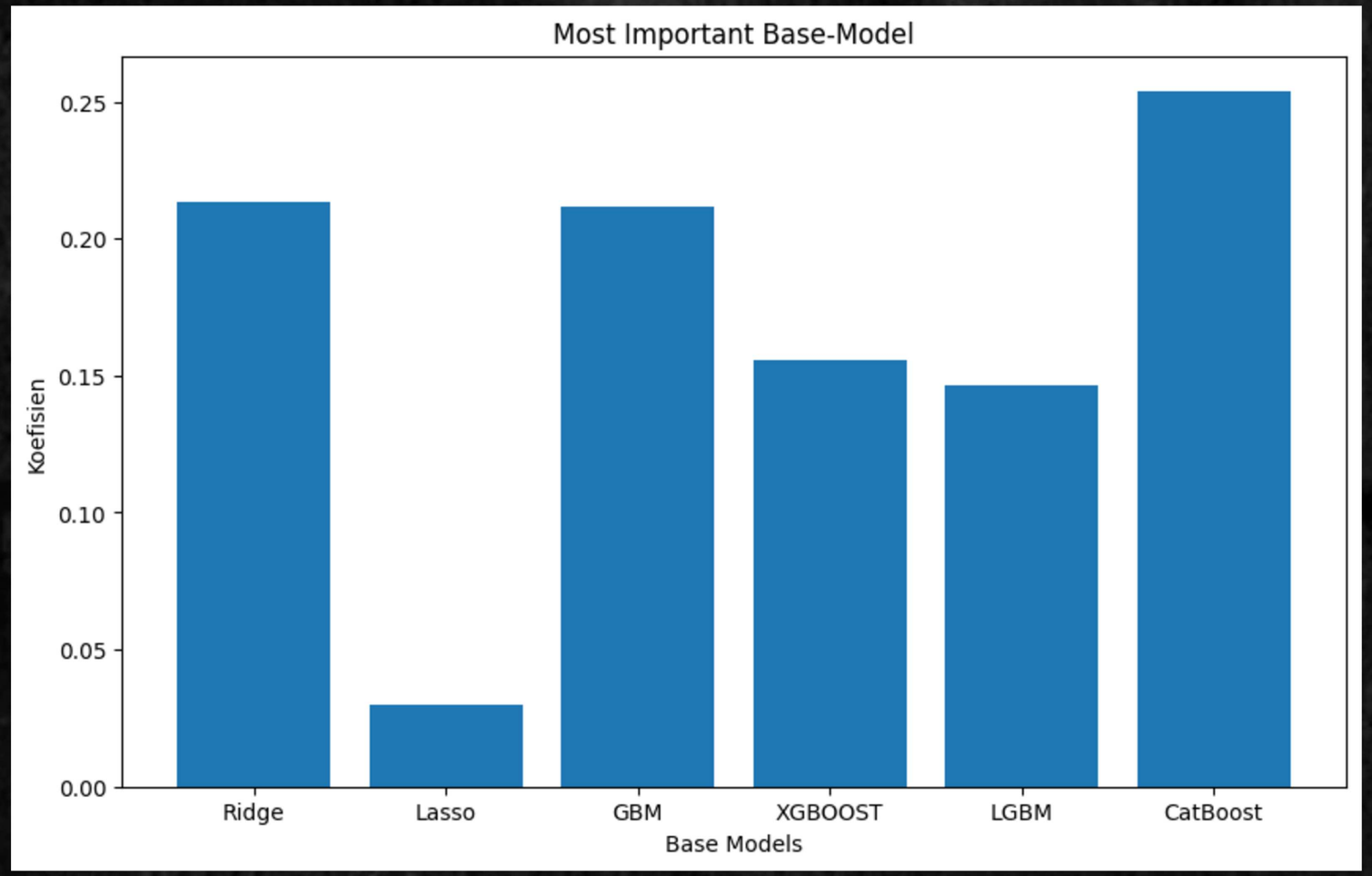
```
cv_fold = KFold(n_splits= 10, shuffle=True, random_state=12)

# CREATE STACKING REGRESSOR
model = StackingRegressor(
    estimators=[
        ('Ridge', ridge),
        ('Lasso', lasso),
        #('LinearRegression', linear),       # USING LINEAR REGRESSION CAUSE A MODEL GET WORSE
        ('GradientBoostingRegressor', gbm),
        ('xgb', xgboost),
        ('lightgbm', lgbm),
        ('catboost', cat)
    ],

    final_estimator= RidgeCV(),
    cv=cv_fold
)

model.fit(x_train, y_train)
```

Stacking Pipeline

Fuse them all!!



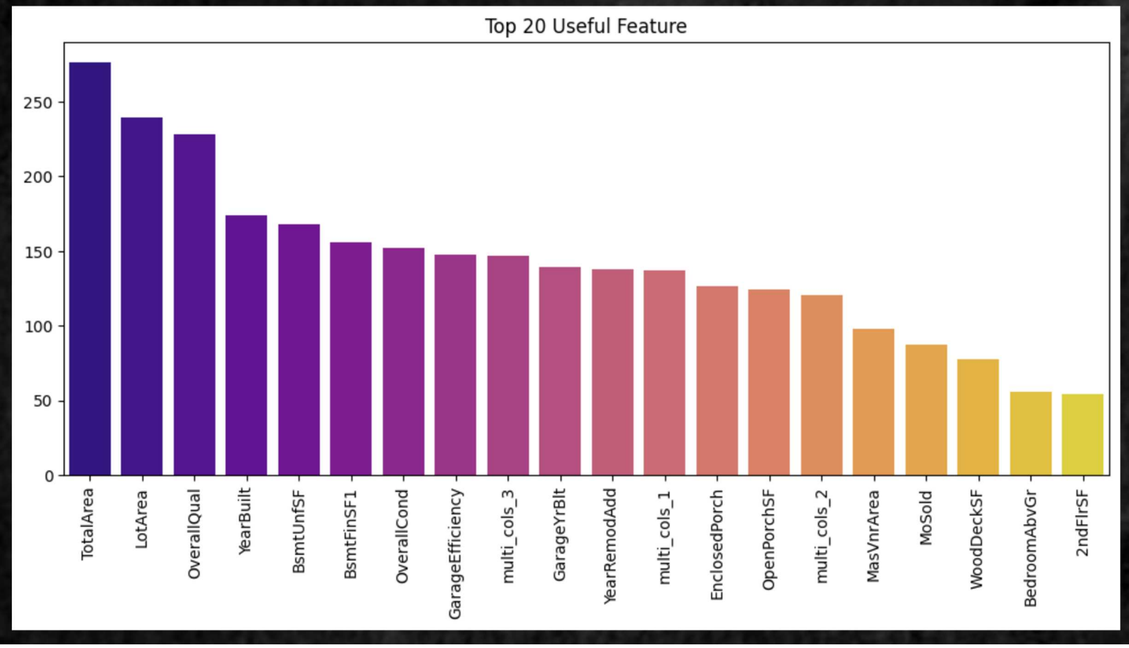
MODEL EVALUATION



Evaluation Results :

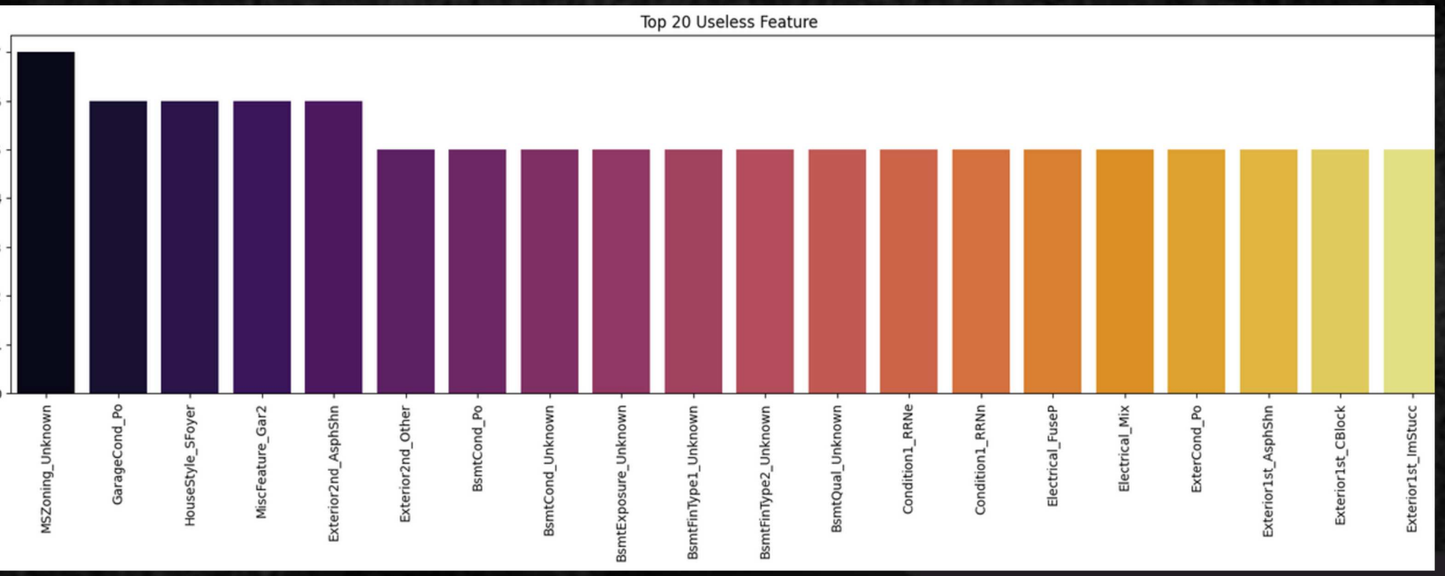
1. **Root Mean Squared Log Error : 0.11993**
2. **Root Mean Squared Error : 13351.31352**

FEATURE IMPORTANCE



There are many columns/features with the label 'Unknown' which are useless, such as:

- MSZoning_Unknown ,
- BsmtCond_Unknown,
- BsmtExposure_Unknown,
- BsmtFinType1_Unknown,
- BsmtFinType2_Unknown,
- BsmtQual_Unknown.



this shows some columns do not match their null values with the label 'Unknown'.

IN THE FUTURE I WANT TO IMPROVE IT AND DO RETRAINING WITH THE BEST SELECTED FEATURES

KAGGLE LEADERBOARD

top 2 out of 5000+ participants

Search

Housing Prices Competition for Kaggle Learn Users

Submit Prediction ...

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

#	Team	Members	Score	Entries	Last	Join
1	Abd-alrhman		757.53217	1	2mo	
2	**Aliffa Agnur**		5585.28580	10	11s	
3	Dana Aubakirovaa		7469.51111	2	1mo	
4	Ömer Tanır		11579.70233	42	1d	
5	Roll20		11602.83533	25	1mo	

Your Best Entry!
Your most recent submission scored 5585.28580, which is an improvement of your previous score of 13327.41496. Great job!

Tweet this

Search

House Prices - Advanced Regression Techniques

Submit Predict

Overview Data Code Models Discussion Leaderboard Rules Team Submissions

210	ChangRun Fu		0.11989	1	1mo
211	**Aliffa Agnur**		0.11993	37	10s
212	tsusshii		0.11997	3	2mo
213	hinemos		0.11999	16	13d
214	[Deleted] 60533d21-61cc-4196-8898-c29f519b210e		0.11999	1	14d

Your Best Entry!
Your most recent submission scored 0.11993, which is an improvement of your previous score of 0.12062. Great job!

Tweet this