House Price Prediction

Isaac Gregory and Bibek Lamsal

Software Development for AI

Imports and Dataset

In [ ]:
```
from google.colab import files
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
import copy as cp
from scipy.stats import f_oneway, chi2_contingency
from sklearn.preprocessing import StandardScaler
import joblib

# Hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# Implementing linear regression
from sklearn.linear_model import LinearRegression

import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score
```

In [ ]:
```
# Replace the below URL with the 'raw' link of your GitHub CSV file
url = "https://raw.githubusercontent.com/Isaac-Gregory/House-Pricing-SWD/refs/l

# Read the CSV file into a pandas DataFrame
data = pd.read_csv(url)

data.drop(['Id'], axis=1, inplace=True)
data.head()
```

Out[ ]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilitie
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPu
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPu
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPu
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPu
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPu

5 rows × 80 columns

In [ ]:
```
# Split numerical from categorical features
X_train = data.drop(['SalePrice'], axis=1)
cat_cols = X_train.select_dtypes(include=['object']).columns.tolist()
```

```
num_cols = X_train.select_dtypes(include=['number']).columns.tolist()
print(len(cat_cols), len(num_cols))
```

```
43 36
```

Feature Analysis

Individual Feature Analysis

In [ ]:

```
nan_cols = {}

# Printing all features with NaN values
for col, i in data.isnull().sum().items():
  if i > 0:
    print(col, i)
    nan_cols[col] = i

# Investigating NaN values in numerical columns
print("---------------------------")
for col in nan_cols:
  if col in num_cols:
    print(col, nan_cols[col])
```

```
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
---------------------------
LotFrontage 259
MasVnrArea 8
GarageYrBlt 81
```

The following is the analysis of the outputted features:

LotFrontage 259 - May not have street connected to property (should just be set to 0 for NaN values)

Alley 1369 - May not have alley access

MasVnrType 872 - No veneer

MasVnrArea 8 - ???? (should probably set NaN values to 0)

NO BASEMENT: BsmtQual 37 BsmtCond 37 BsmtExposure 38 BsmtFinType1 37 BsmtFinType2 38

Electrical 1 - ????

FireplaceQu 690 - No fireplace

NO GARAGE: GarageType 81 GarageYrBlt 81 GarageFinish 81 GarageQual 81 GarageCond 81

PoolQC 1453 - No Pool

Fence 1179 - No fence

MiscFeature 1406 - No miscellaneous (high amount, but potentially highly predictive)

In [ ]:
```
# Replacing numerical NaNs with zeros
data['LotFrontage'].fillna(0, inplace=True)
data['MasVnrArea'].fillna(0, inplace=True)

# Since NaNs have a meaning in this dataset, we will make them a string part o
# Convert all NaN in cat_cols to strings
for col in cat_cols:
  data[col].fillna('N/A', inplace=True)
```

Categorical Class Distribution Analysis

In [ ]:
```
# for col in cat_cols:
#   plt.bar(data[col].value_counts().index, data[col].value_counts())
#   plt.title(col)
#   plt.show()
```

Analysis of the above distributions:

Neighborhoods, Condition1, Condition2 are reliant on local (Iowa) information. Should be removed prior to final model.

Utilities could have been useful, but has too uneven of a distribution.

In [ ]:
```
# # Dropping features dependent on this specific dataset (i.e. not generalizab
# data.drop(['Neighborhood', 'Condition1', 'Condition2', 'Utilities', 'YearBui

# # Resplitting numerical from categorical features
# X_train = data.drop(['SalePrice'], axis=1)
# cat_cols = X_train.select_dtypes(include=['object']).columns.tolist()
# num_cols = X_train.select_dtypes(include=['number']).columns.tolist()
```

Numerical Noisy Feature Analysis

In [ ]:
```
# prompt: remove SalesPrice from X_train adn save it in y_train,split categori
numerical_features = num_cols

# Separate labels from samples
```

```
y_train = data['SalePrice']

# Find and graph correlation between all features and the labels
targ_corr_num = {}
for feature in numerical_features:
  # Setting zeros in feature to NaN
  updated_data = cp.deepcopy(data[feature].replace(0, np.nan))

  # Calculate correlation without zeros
  pearson_woz = updated_data.corr(y_train)
  spearman_woz = updated_data.corr(y_train, method='spearman')

  # Calculate correlation with zeros
  pearson_wz = data[feature].corr(y_train)
  spearman_wz = data[feature].corr(y_train, method='spearman')

  # Save correlation into array
  targ_corr_num[feature] = (pearson_wz,spearman_wz, pearson_woz, spearman_woz)

# Observe features and remove using threshold on array
noisy_numerical_strong = []
noisy_numerical_weak = []
for feature in targ_corr_num:
  with_zero = False
  without_zero = False

  # Determining impact of inclusion/exclusion of zeros
  if(abs(targ_corr_num[feature][0]) < 0.3 and abs(targ_corr_num[feature][1]) <
    with_zero = True
  if(abs(targ_corr_num[feature][2]) < 0.3 and abs(targ_corr_num[feature][3]) <
    without_zero = True

  # Separating features by strong or weak impact of zeros
  if with_zero and without_zero:
    noisy_numerical_strong.append(feature)
  elif (not with_zero and without_zero) or (with_zero and not without_zero):
    noisy_numerical_weak.append(feature)

# Printing strong and weak noisy features
for feature in noisy_numerical_strong:
  print(feature, targ_corr_num[feature])
print("------------------------------------------------------------------")
for feature in noisy_numerical_weak:
  print(feature, targ_corr_num[feature])

# Setting only to strong noise for now
noisy_numerical = noisy_numerical_strong

print(noisy_numerical)
```

```
MSSubClass (-0.08428413512659531, 0.007192252911733476, -0.08428413512659531, 
0.007192252911733476)
OverallCond (-0.07785589404867803, -0.12932494660061317, -0.07785589404867803, 
-0.12932494660061317)
BsmtFinSF2 (-0.011378121450215125, -0.03880613204589418, 0.19895609430836594, 
0.11843388567766258)
BsmtUnfSF (0.21447910554696892, 0.185196629420762, 0.1692610004951418, 0.11282
241442039748)
BsmtFullBath (0.22712223313149382, 0.22512486719612368, 0.01143916334040866, 
0.024792357595010216)
BsmtHalfBath (-0.016844154297359016, -0.012188876310787316, -0.028834567185481
722, -0.016703554806725977)
BedroomAbvGr (0.16821315430073988, 0.23490671789027862, 0.18093669310848812, 
0.24029795563186981)
KitchenAbvGr (-0.13590737084214122, -0.1648257549850205, -0.1392006921778579, 
-0.16924325951926172)
EnclosedPorch (-0.12857795792595653, -0.2183936205521982, 0.24127883630117508, 
0.24740585729184555)
3SsnPorch (0.04458366533574846, 0.06544021620062833, 0.06393243256889079, 0.22
91441132422282)
MiscVal (-0.02118957964030325, -0.0627270024962966, 0.08896338917298922, 0.154
58686602901253)
MoSold (0.046432245223819384, 0.06943224370457042, 0.046432245223819384, 0.069
43224370457042)
YrSold (-0.028922585168730378, -0.029899134912615286, -0.028922585168730378, -
0.029899134912615286)
-------------------------------------------------------------------
LotFrontage (0.2096239447994838, 0.23849611297908194, 0.35179909657067804, 0.4
090755179546496)
LowQualFinSF (-0.02560613000067959, -0.06771915407896568, 0.30007501655501334, 
0.10778447057412376)
HalfBath (0.2841076755947831, 0.34300754918568294, -0.0843917112717989, -0.09
943111283268087)
Fireplaces (0.46692883675152724, 0.5192474498367013, 0.12166058421363923, 0.07
482399463375336)
WoodDeckSF (0.3244134445681294, 0.35380160795878884, 0.1937060123752066, 0.206
79337134940587)
OpenPorchSF (0.3158562271160555, 0.47756066228252647, 0.08645298857147718, 0.1
592807444169871)
ScreenPorch (0.11144657114291105, 0.1000697202012266, 0.2554300795487841, 0.31
540170391892247)
PoolArea (0.09240354949187321, 0.058452996689891755, -0.014091521506356936, 0.
3571428571428572)
['MSSubClass', 'OverallCond', 'BsmtFinSF2', 'BsmtUnfSF', 'BsmtFullBath', 'Bsmt
HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'EnclosedPorch', '3SsnPorch', 'Misc
Val', 'MoSold', 'YrSold']
```

In [ ]:

```
# for col in numerical_features:
#   if (col not in noisy_numerical_strong) and (col not in noisy_numerical_weal
#     plt.scatter(data[col], y_train)
#     plt.title(col)
#     plt.show()
# print("------------------------------------------------------------------")
# for col in noisy_numerical_strong:
#   plt.scatter(data[col], y_train)
#   plt.title(col)
#   plt.show()
# print("------------------------------------------------------------------")
# for col in noisy_numerical_weak:
#   plt.scatter(data[col], y_train)
```

```
#   plt.title(col)
#   plt.show()
```

Analysis of the above

Halfbath may not be correctly indicative of real-world since higher half-bath count would typically lead to higher prices. However, this could be due to absense of full bathrooms?

Garage Cars may also be fairly noisy, but it may be that more rural homes have higher amount of cars instead.

YearBuilt and YearRemodAdd may both be too far in the past to utilize (right?)

MsSubClass is more categorical and shouldn't be unincluded from the dataset due to a numerical correlation assumption.

Overall Condition looks more linear.

Year-sold won't be very applicable to the model that we are planning on training.

Categorical Noisy Feature Analysis

In [ ]:
```
# For categorical features
categorical_features = cat_cols
targ_corr_cat = {}

for feature in categorical_features:
  # Group the target variable by the categorical feature
  grouped_target = y_train.groupby(data[feature])

  # Perform ANOVA
  f_statistic, p_value = f_oneway(*[grouped_target.get_group(group) for group :

  # Save results
  targ_corr_cat[feature] = (f_statistic, p_value)

# Observe features and remove using a threshold on p-value
noisy_categorical = []
for feature in targ_corr_cat:
  if targ_corr_cat[feature][1] > 0.01:  # Adjust threshold as needed
    print(feature, targ_corr_cat[feature])
    noisy_categorical.append(feature)
print(noisy_categorical)
```

```
Street (2.4592895583691994, 0.11704860406782483)
Utilities (0.29880407484898486, 0.5847167739689381)
LandSlope (1.9588170374149438, 0.1413963584114019)
Condition2 (2.0738986215227877, 0.043425658360948464)
MiscFeature (2.593622339924057, 0.0350036718754261)
['Street', 'Utilities', 'LandSlope', 'Condition2', 'MiscFeature']
```

In [ ]:
```
# for col in categorical_features:
#   if col not in noisy_categorical:
#     plt.scatter(data[col].astype(str), y_train)
#     plt.title(col)
```

```
#     plt.show()
# print("-------------------------------------------------------------------")
# for col in noisy_categorical:
#   plt.scatter(data[col].astype(str), y_train)
#   plt.title(col)
#   plt.show()
```

Analysis of the above

Electrical does have one NaN point that will need to be handled.

GarageCond and GarageQual appear to not be indicative of the "real-world."

Fence does not appear to be a very good distribution.

Land Slope appears to be informative despite being considered "noise."

Feature vs Feature Analysis

In [ ]:

```
# Create copies of lists for tracking
cat_cols_copy = cp.deepcopy(cat_cols)
num_cols_copy = cp.deepcopy(num_cols)

# Dictionaries for tracking results
cat_corr = {}
num_corr = {}
```

In [ ]:

```
# Use chi-squared analysis on categorical features
for col1 in cat_cols:
  for col2 in cat_cols_copy:

    # No need to compare the same feature
    if col1 == col2:
      continue

    # Contingency table
    contingency_tbl = pd.crosstab(data[col1], data[col2])

    # Ensure table has values (i.e. is not empty)
    if contingency_tbl.size == 0:
      # Assigning default values
      cat_corr[(col1, col2)] = (0, 1)
      continue

    # Perform chi-squared analysis
    chi, p, dof, expected = chi2_contingency(contingency_tbl)

    # Save results
    cat_corr[(col1, col2)] = (chi, p)

  # Remove col from copied list
  if col1 in cat_cols_copy:
    cat_cols_copy.remove(col1)
```

In [ ]:
```
# dependent categorical features
for key1,key2 in cat_corr:
  if cat_corr[key1,key2][1] == 0.0:
    print(key1,key2, cat_corr[key1,key2])
```

```
MSZoning Neighborhood (2486.263987999627, 0.0)
Neighborhood Exterior2nd (2543.991276041277, 0.0)
Exterior1st Exterior2nd (11868.678367195604, 0.0)
Foundation BsmtQual (1664.4943148860546, 0.0)
BsmtQual BsmtCond (1631.9724267986735, 0.0)
BsmtQual BsmtExposure (1596.1099051272488, 0.0)
BsmtQual BsmtFinType1 (1960.385537564252, 0.0)
BsmtExposure BsmtFinType1 (1608.3712506742656, 0.0)
BsmtFinType1 BsmtFinType2 (1775.454707221523, 0.0)
GarageType GarageFinish (2068.5422757051683, 0.0)
GarageFinish GarageQual (1531.0202997120914, 0.0)
GarageFinish GarageCond (1517.9040342335852, 0.0)
GarageQual GarageCond (3633.062233479517, 0.0)
SaleType SaleCondition (1652.6750772643866, 0.0)
```

Creating groupings:

SaleType concat with SaleCondition

Garage data

Basement data

Exteriors

(Neighborhood gets dropped in feature selection and can be ignored)

In [ ]:
```
# Use correlation analysis on numerical features
for col1 in num_cols:
  for col2 in num_cols_copy:

    # No need to compare the same feature
    if col1 == col2:
      continue

    # Calculate correlation
    pearson_temp = data[col1].corr(data[col2])
    spearman_temp = data[col1].corr(data[col2], method='spearman')

    # Save correlation into array
    num_corr[(col1, col2)] = (pearson_temp, spearman_temp)

  # Remove col from copied list
  if col1 in num_cols_copy:
    num_cols_copy.remove(col1)
```

In [ ]:
```
# redundant numerical features
for key1,key2 in num_corr:
  if num_corr[key1,key2][1] >= 0.85:
    print(key1,key2, num_corr[key1,key2])
```

```
YearBuilt GarageYrBlt (0.825667484174342, 0.8905463872089356)
GarageCars GarageArea (0.8824754142814625, 0.8533173766076401)
```

Feature Selection

In [ ]:
```
# Removing the redudant columns (i.e. are not generalizable outside of dataset,
redundant_column = ['Neighborhood', 'Condition1', 'Condition2', 'Utilities', '

# Removing highly correlated columns (that only require one combination of fea
high_corr_column = ['Exterior1st', 'SaleType', 'GarageCars']

column_names_to_remove = redundant_column + noisy_numerical + noisy_categorica

# Copying data and removing features
data_copy = data.copy()
for column_name in column_names_to_remove:
  if column_name in data_copy.columns:
    data_copy = data_copy.drop(column_name, axis=1)
  if column_name in cat_cols:
    cat_cols.remove(column_name)
  if column_name in num_cols:
    num_cols.remove(column_name)

print(data_copy.columns)
```

```
Index(['MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'LotShape',
       'LandContour', 'LotConfig', 'BldgType', 'HouseStyle', 'OverallQual',
       'RoofStyle', 'RoofMatl', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
       'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'FullBath',
       'HalfBath', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces',
       'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageArea', 'GarageQua
l',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'ScreenPorch',
       'PoolArea', 'PoolQC', 'Fence', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

Feature Engineering

Numerical Feature Combinations

Categorical Feature Combinations

In [ ]:
```
# Preprocessing function used in website (combination of the following cells)
def preprocessing(df):
  data_copy = df.copy()
  # Combining basement quality features
  features_to_combine1 = ['BsmtQual', 'BsmtCond', 'BsmtExposure']
  mapping1 = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "N
  data_copy[features_to_combine1] = data_copy[features_to_combine1].replace(ma

  data_copy["BsmtRating"]= data_copy[features_to_combine1].apply(
      lambda x: (x['BsmtQual'] + 5 - x["BsmtCond"]) + x["BsmtExposure"],axis=1
  data_copy.drop(features_to_combine1, axis=1, inplace=True)
```

```
  # Combining basement finish type features
  features_to_combine2 = ['BsmtFinType1', 'BsmtFinType2']
  mapping2 = {"GLQ": 6, "ALQ": 5, "BLQ": 4, "Rec": 3, "LwQ": 2, "Unf": 1, "NA"
  data_copy[features_to_combine2] = data_copy[features_to_combine2].replace(ma

  data_copy["BsmtFinType"]= data_copy[features_to_combine2].apply(
      lambda x: (x['BsmtFinType1'] + x["BsmtFinType2"]),axis=1)
  data_copy.drop(features_to_combine2, axis=1, inplace=True)

  # Combining garage quality features
  features_to_combine3 = ['GarageQual', 'GarageCond']
  mapping3 = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "N
  data_copy[features_to_combine3] = data_copy[features_to_combine3].replace(ma

  data_copy["GarageRating"]= data_copy[features_to_combine3].apply(
      lambda x: (x['GarageQual'] + 5 - x["GarageCond"]),axis=1)
  data_copy.drop(features_to_combine3, axis=1, inplace=True)

  # Combining garage type features
  features_to_combine4 = ['GarageType', 'GarageFinish']

  data_copy["GarageInfo"]= data_copy[features_to_combine4].apply(
      lambda x: (x['GarageType'] + "-" + x["GarageFinish"]),axis=1)
  data_copy.drop(features_to_combine4, axis=1, inplace=True)

  cat_cols = data_copy.select_dtypes(include=['object']).columns.tolist()
  num_cols = data_copy.select_dtypes(include=['number']).columns.tolist()

  # Encode categorical features using LabelEncoder
  encoded_data = cp.deepcopy(data_copy)
  encoder_list = {}
  for col in cat_cols:

    # Encoding with rank order
    if "Ex" in encoded_data[col].unique() or "TA" in encoded_data[col].unique(
      mapping = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1
      encoded_data[col] = encoded_data[col].replace(mapping)

    else:
      encoder_list1 = joblib.load('encoder_list.pkl')
      encoder = encoder_list1[col]
      encoded_data[col] = encoder.transform(encoded_data[col])

  X_scaler = joblib.load('X_scaler.pkl')
  num_cols_norm = X_scaler.transform(encoded_data[num_cols])
  num_df = pd.DataFrame(num_cols_norm, columns=num_cols)

  # Replace the original numerical columns with normalized ones
  for col in num_cols:
    encoded_data[col] = num_df[col]

  return encoded_data
```

In [ ]:

```
# Combining basement quality features
features_to_combine = ['BsmtQual', 'BsmtCond', 'BsmtExposure']
mapping = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "No"
data_copy[features_to_combine] = data_copy[features_to_combine].replace(mappin

data_copy["BsmtRating"]= data_copy[features_to_combine].apply(
```

```
    lambda x: (x['BsmtQual'] + 5 - x["BsmtCond"]) + x["BsmtExposure"],axis=1)
data_copy.drop(features_to_combine, axis=1, inplace=True)
```

In [ ]:
```
# Combining basement finish type features
features_to_combine = ['BsmtFinType1', 'BsmtFinType2']
mapping = {"GLQ": 6, "ALQ": 5, "BLQ": 4, "Rec": 3, "LwQ": 2, "Unf": 1, "NA": 0
data_copy[features_to_combine] = data_copy[features_to_combine].replace(mappin

data_copy["BsmtFinType"]= data_copy[features_to_combine].apply(
    lambda x: (x['BsmtFinType1'] + x["BsmtFinType2"]),axis=1)
data_copy.drop(features_to_combine, axis=1, inplace=True)
```

In [ ]:
```
# Combining garage quality features
features_to_combine = ['GarageQual', 'GarageCond']
mapping = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "No"
data_copy[features_to_combine] = data_copy[features_to_combine].replace(mappin

data_copy["GarageRating"]= data_copy[features_to_combine].apply(
    lambda x: (x['GarageQual'] + 5 - x["GarageCond"]),axis=1)
data_copy.drop(features_to_combine, axis=1, inplace=True)
```

In [ ]:
```
# Combining garage type features
features_to_combine = ['GarageType', 'GarageFinish']

data_copy["GarageInfo"]= data_copy[features_to_combine].apply(
    lambda x: (x['GarageType'] + "-" + x["GarageFinish"]),axis=1)
data_copy.drop(features_to_combine, axis=1, inplace=True)
```

Training Models

In [ ]:
```
# Getting categorical features
cat_cols = data_copy.select_dtypes(include=['object']).columns.tolist()

# Encode categorical features using LabelEncoder
encoded_data = cp.deepcopy(data_copy)
encoder_list = {}
for col in cat_cols:

  # Encoding with rank order
  if "Ex" in encoded_data[col].unique() or "TA" in encoded_data[col].unique() 
    mapping = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, 
    encoded_data[col] = encoded_data[col].replace(mapping)

    # Adding col to encoder_list, using None for easy detection later on
    encoder_list[col] = None

  # Encoding normally
  else:
    encoder = LabelEncoder()
    encoder.fit(encoded_data[col])
    encoded_data[col] = encoder.transform(encoded_data[col])
    encoder_list[col] = encoder
```

In [ ]:
```
# Saving
joblib.dump(encoder_list, 'encoder_list.pkl')
```

Out[ ]: ['encoder_list.pkl']

In [ ]:
```
encoded_data
```

Out[ ]:

	MSZoning	LotFrontage	LotArea	Alley	LotShape	LandContour	LotConfig	BldgType	Ho
0	3	65.0	8450	1	3	3	4	0	
1	3	80.0	9600	1	3	3	2	0	
2	3	68.0	11250	1	0	3	4	0	
3	3	60.0	9550	1	0	3	0	0	
4	3	84.0	14260	1	0	3	2	0	
...	...	...	...	...	...	...	...	...	
1455	3	62.0	7917	1	3	3	4	0	
1456	3	85.0	13175	1	3	3	4	0	
1457	3	66.0	9042	1	3	3	4	0	
1458	3	68.0	9717	1	3	3	4	0	
1459	3	75.0	9937	1	3	3	4	0	

1460 rows × 49 columns

In [ ]:
```
# Separate features and target
X = encoded_data.drop('SalePrice', axis=1)
y = encoded_data['SalePrice']

# Normalize numerical features
X_scaler = StandardScaler()
X_scaler.fit(X)
num_cols_norm = X_scaler.transform(X)
num_df = pd.DataFrame(num_cols_norm, columns=X.columns)

# Replace the original numerical columns with normalized ones
for col in num_cols:
  X[col] = num_df[col]
```

In [ ]:
```
joblib.dump(X_scaler, 'X_scaler.pkl')
```

Out[ ]: ['X_scaler.pkl']

In [ ]:
```
# Normalize target feature
y_scaler = StandardScaler()
y_scaler.fit(y.values.reshape(-1, 1))
y_norm = y_scaler.transform(y.values.reshape(-1, 1)) # Reshape y to a 2D array
y_df = pd.DataFrame(y_norm, columns=['SalePrice']) # Use y_norm for the DataFra
y = y_df['SalePrice']
```

In [ ]:
```
joblib.dump(y_scaler, 'y_scaler.pkl')
```

Out[ ]: ['y_scaler.pkl']

In [ ]:
```
# Viewing results of feature selection and engineering
print(len(X.columns))
print(X.columns)
```

```
48
Index(['MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'LotShape',
       'LandContour', 'LotConfig', 'BldgType', 'HouseStyle', 'OverallQual',
       'RoofStyle', 'RoofMatl', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtFinSF1', 'TotalBsmtSF',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF',
       '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'FullBath', 'HalfBath',
       'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces',
       'FireplaceQu', 'GarageArea', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'SaleCondition',
       'BsmtRating', 'BsmtFinType', 'GarageRating', 'GarageInfo'],
      dtype='object')
```

In [ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randon
```

In [ ]:
```
# Creating model
lin_model = LinearRegression()

# Using GridSearch to tune the linear regression model
param_grid = {'copy_X': [True, False],
              'fit_intercept': [True, False],
              'n_jobs': [1, 2, 5, 10, 15, None],
              'positive': [True, False]}

lin_cv = GridSearchCV(lin_model, param_grid, cv=5, scoring='neg_mean_squared_e
lin_cv.fit(X_train, y_train)

# Printing the best results
print(f"Best Hyperparameters: {lin_cv.best_params_}")

# Printing the training score
print(f"Best Score: {lin_cv.best_score_}")

# Scoring test set
print(f"Test Score: {lin_cv.score(X_test, y_test)}")
```

```
Best Hyperparameters: {'copy_X': True, 'fit_intercept': False, 'n_jobs': 1, 'p
ositive': True}
Best Score: -0.28460689408770423
Test Score: -0.2044984025849035
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.p
y:540: FitFailedWarning:
12 fits failed out of a total of 240.
The score on these train-test partitions for these parameters will be set to n
an.
If these failures are not expected, you can try to debug them by setting error
_score='raise'.

Below are more details about the failures:
------------------------------------------------------------------------------
--
12 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_valid
ation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, i
n wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_base.p
y", line 647, in fit
    self.coef_ = optimize.nnls(X, y)[0]
  File "/usr/local/lib/python3.10/dist-packages/scipy/optimize/_nnls.py", line
93, in nnls
    raise RuntimeError("Maximum number of iterations reached.")
RuntimeError: Maximum number of iterations reached.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:110
3: UserWarning: One or more of the test scores are non-finite: [         nan -
12.93131827          nan -12.93131827          nan
 -12.93131827          nan -12.93131827          nan -12.93131827
          nan -12.93131827  -0.28460689 -12.04921986  -0.28460689
 -12.04921986  -0.28460689 -12.04921986  -0.28460689 -12.04921986
  -0.28460689 -12.04921986  -0.28460689 -12.04921986          nan
 -12.93131827          nan -12.93131827          nan -12.93131827
          nan -12.93131827          nan -12.93131827          nan
 -12.93131827  -0.28460689 -12.04921986  -0.28460689 -12.04921986
  -0.28460689 -12.04921986  -0.28460689 -12.04921986  -0.28460689
 -12.04921986  -0.28460689 -12.04921986]
  warnings.warn(
```

In [ ]:

```
# Getting test predictions
y_pred = lin_cv.predict(X_test)

# Evaluating the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)

# Obtaining RMSE in more interpretable terms
unscaled_pred = y_scaler.inverse_transform(y_pred.reshape(-1, 1))
unscaled_actual = y_scaler.inverse_transform(y_test.values.reshape(-1, 1))
unscaled_rmse = np.sqrt(mean_squared_error(unscaled_actual, unscaled_pred))
print("Unscaled RMSE:", unscaled_rmse)
```

```
RMSE: 0.452214995975259
Unscaled RMSE: 35912.78590063204
```

In [ ]:

```
joblib.dump(lin_cv, 'lin_cv.pkl')
```

Out[ ]:
```
['lin_cv.pkl']
```