Machine Learning

Objective: This file

1. Preprocesses realistic data (multiple variable types) in a pipeline that handles each variable type
2. Estimates a model using CV
3. Hypertunes a model on a CV folds within training sample
4. Finally, evaluate its performance in the test sample

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')

# load data and split off X and y
housing = pd.read_csv('input_data2/housing_train.csv')
y = np.log(housing.v_SalePrice)
housing = housing.drop('v_SalePrice',axis=1)

housing.head()
```

Out[1]:

	parcel	v_MS_SubClass	v_MS_Zoning	v_Lot_Frontage	v_Lot_Area
0	1056_528110080	20	RL	107.0	13891
1	1055_528108150	20	RL	98.0	12704
2	1053_528104050	20	RL	114.0	14803
3	2213_909275160	20	RL	126.0	13108
4	1051_528102030	20	RL	96.0	12444

5 rows × 80 columns

To ensure you can be graded accurately, we need to make the "randomness" predictable. (I.e. you should get the exact same answers every single time we run this.)

Per the recommendations in the sk-learn documentation, what that means is we need to put `random_state=rng` inside every function in this file that accepts "random_state" as an argument.

In [2]:
```python
# create test set for use later - notice the (random_state=rng)
rng = np.random.RandomState(0)
X_train, X_test, y_train, y_test = train_test_split(housing, y, random
```

Part 1: Preprocessing the data

1. Set up a single pipeline called `preproc_pipe` to preprocess the data.
   A. For **all** numerical variables, impute missing values with SimpleImputer and scale them with StandardScaler
   B. `v_Lot_Config`: Use OneHotEncoder on it
   C. Drop any other variables (handle this **inside** the pipeline)
2. Use this pipeline to preprocess X_train.
   A. Describe the resulting data **with two digits.**
   B. How many columns are in this object?

HINTS:

- *You do NOT need to type the names of all variables. There is a lil trick to catch all the variables.*
- *The first few rows of my print out look like this:*

	count	mean	std	min	25%	50%	75%	max
v_MS_SubClass	1455	0	1	-0.89	-0.89	-0.2	0.26	3.03
v_Lot_Frontage	1455	0	1	-2.2	-0.43	0	0.39	11.07
v_Lot_Area	1455	0	1	-1.17	-0.39	-0.11	0.19	20.68
v_Overall_Qual	1455	0	1	-3.7	-0.81	-0.09	0.64	2.8

In [3]:
```python
#housing.info()
```

In [4]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from df_after_transform import df_after_transform
from sklearn import set_config
from sklearn.calibration import CalibrationDisplay
from sklearn.compose import (
    ColumnTransformer,
    make_column_selector,
    make_column_transformer,
)
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    ConfusionMatrixDisplay,
```

```
    DetCurveDisplay,
    PrecisionRecallDisplay,
    RocCurveDisplay,
    classification_report,
    r2_score
)
from sklearn.model_selection import (
    GridSearchCV,
    KFold,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.compose import make_column_transformer, make_column_selec
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, Polyn
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import HistGradientBoostingRegressor
import numpy as np
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingRegressor
```

In [5]:

```
# Make Numerical Pipeline
numer_pipe = make_pipeline(SimpleImputer(strategy="mean"), StandardSca

# Make Categorical Pipeline
cat_pipe   = make_pipeline(OneHotEncoder(sparse_output=False))

# Combine those
preproc_pipe = make_column_transformer(
    (numer_pipe, make_column_selector(dtype_include=np.number)),
    (cat_pipe, ["v_Lot_Config"]),
    remainder="drop",
)

# Make preprocessing dataframe
preproc_df = df_after_transform(preproc_pipe, X_train)
```

In [6]:

```
print(f"Number of columns: {preproc_df.shape[1]}")

preproc_df.describe().T.round(2)
```

```
Number of columns: 41
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
v_MS_SubClass	1455.0	0.00	1.00	-0.89	-0.89	-0.20	0.26	3.03
v_Lot_Frontage	1455.0	0.00	1.00	-2.20	-0.43	0.00	0.39	11.07
v_Lot_Area	1455.0	0.00	1.00	-1.17	-0.39	-0.11	0.19	20.68

v_Overall_Qual	1455.0	0.00	1.00	-3.70	-0.81	-0.09	0.64	2.80
v_Overall_Cond	1455.0	0.00	1.00	-4.30	-0.53	-0.53	0.41	3.24
v_Year_Built	1455.0	-0.00	1.00	-3.08	-0.62	0.05	0.98	1.22
v_Year_Remod/Add	1455.0	0.00	1.00	-1.63	-0.91	0.43	0.96	1.20
v_Mas_Vnr_Area	1455.0	0.00	1.00	-0.57	-0.57	-0.57	0.33	7.87
v_BsmtFin_SF_1	1455.0	0.00	1.00	-0.96	-0.96	-0.16	0.65	11.20
v_BsmtFin_SF_2	1455.0	0.00	1.00	-0.29	-0.29	-0.29	-0.29	8.29
v_Bsmt_Unf_SF	1455.0	0.00	1.00	-1.28	-0.77	-0.23	0.55	3.58
v_Total_Bsmt_SF	1455.0	0.00	1.00	-2.39	-0.59	-0.14	0.55	11.35
v_1st_Flr_SF	1455.0	-0.00	1.00	-2.07	-0.68	-0.19	0.55	9.76
v_2nd_Flr_SF	1455.0	-0.00	1.00	-0.78	-0.78	-0.78	0.85	3.98
v_Low_Qual_Fin_SF	1455.0	0.00	1.00	-0.09	-0.09	-0.09	-0.09	14.09
v_Gr_Liv_Area	1455.0	-0.00	1.00	-2.23	-0.72	-0.14	0.43	7.82
v_Bsmt_Full_Bath	1455.0	-0.00	1.00	-0.82	-0.82	-0.82	1.11	3.04
v_Bsmt_Half_Bath	1455.0	-0.00	1.00	-0.24	-0.24	-0.24	-0.24	7.94
v_Full_Bath	1455.0	0.00	1.00	-2.84	-1.03	0.78	0.78	2.59
v_Half_Bath	1455.0	0.00	1.00	-0.76	-0.76	-0.76	1.25	3.26
v_Bedroom_AbvGr	1455.0	0.00	1.00	-3.51	-1.07	0.15	0.15	6.24
v_Kitchen_AbvGr	1455.0	-0.00	1.00	-5.17	-0.19	-0.19	-0.19	4.78
v_TotRms_AbvGrd	1455.0	-0.00	1.00	-2.83	-0.93	-0.30	0.33	5.39
v_Fireplaces	1455.0	-0.00	1.00	-0.94	-0.94	0.63	0.63	5.32
v_Garage_Yr_Blt	1455.0	-0.00	1.00	-3.41	-0.67	0.00	0.97	1.22
v_Garage_Cars	1455.0	-0.00	1.00	-2.34	-1.03	0.28	0.28	2.91
v_Garage_Area	1455.0	-0.00	1.00	-2.20	-0.69	0.01	0.46	4.65
v_Wood_Deck_SF	1455.0	0.00	1.00	-0.74	-0.74	-0.74	0.59	10.54
v_Open_Porch_SF	1455.0	-0.00	1.00	-0.71	-0.71	-0.31	0.32	7.67
v_Enclosed_Porch	1455.0	-0.00	1.00	-0.36	-0.36	-0.36	-0.36	9.33
v_3Ssn_Porch	1455.0	0.00	1.00	-0.09	-0.09	-0.09	-0.09	19.74
v_Screen_Porch	1455.0	-0.00	1.00	-0.29	-0.29	-0.29	-0.29	9.69
v_Pool_Area	1455.0	-0.00	1.00	-0.08	-0.08	-0.08	-0.08	17.05
v_Misc_Val	1455.0	-0.00	1.00	-0.09	-0.09	-0.09	-0.09	24.19

v_Mo_Sold	1455.0	-0.00	1.00	-2.03	-0.55	-0.18	0.56	2.04
v_Yr_Sold	1455.0	-0.00	1.00	-1.24	-1.24	0.00	1.25	1.25
v_Lot_Config_Corner	1455.0	0.18	0.38	0.00	0.00	0.00	0.00	1.00
v_Lot_Config_CulDSac	1455.0	0.06	0.24	0.00	0.00	0.00	0.00	1.00
v_Lot_Config_FR2	1455.0	0.02	0.15	0.00	0.00	0.00	0.00	1.00
v_Lot_Config_FR3	1455.0	0.01	0.08	0.00	0.00	0.00	0.00	1.00
v_Lot_Config_Inside	1455.0	0.73	0.44	0.00	0.00	1.00	1.00	1.00

Model 1

In [7]:

```
# 1:
# Define Model
lasso_model = make_pipeline(preproc_pipe, Lasso(alpha = 0.3, tol = 1e-

# Perform cross-validation
scores = cross_val_score(lasso_model, X_train, y_train, cv = 10, scori

# Print the mean test score with 5 digits
print(f"Mean R^2 score (CV=10, alpha=0.3): {scores.mean():.5f}")
```

```
Mean R^2 score (CV=10, alpha=0.3): 0.08666
```

In [8]:

```
# 2:
from tqdm import tqdm

alphas = np.arange(0.005, 0.01, 0.00001)  # 5-digit precision
mean_scores = []

for alpha in tqdm(alphas):
    model = make_pipeline(preproc_pipe, Lasso(alpha=alpha, max_iter=10
    scores = cross_val_score(model, X_train, y_train, cv=10, scoring='
    mean_scores.append(scores.mean())

# Find the best alpha
best_idx = np.argmax(mean_scores)
best_alpha = alphas[best_idx]
best_score = mean_scores[best_idx]

print(f"A --> Best alpha: {best_alpha:.5f}")
print(f"B --> Best mean CV R²: {best_score:.5f}")
```

```
100%|██████████████████████████████████████████████████| 500/500 [01:17<00:00,
6.48it/s]
A --> Best alpha: 0.00771
B --> Best mean CV R²: 0.83108
```

In [9]:
```python
# Fit the best model on ALL of X_train
final_model = make_pipeline(preproc_pipe, Lasso(alpha=best_alpha, max_
final_model.fit(X_train, y_train)

# Get feature names
feature_names = final_model.named_steps['columntransformer'].get_featu

# Get coefficients
lasso_coef = final_model.named_steps['lasso'].coef_

# Non-zero coefficients
non_zero_mask = lasso_coef != 0
non_zero_features = feature_names[non_zero_mask]
non_zero_values = lasso_coef[non_zero_mask]

print(f'C --> Number of non-zero coefficients: {len(non_zero_values)}'
print(' ')

# Get top 5 highest coefficients
top_5_idx = np.argsort(non_zero_values)[-5:][::-1]
print("D --> Top 5 coefficients:")
for i in top_5_idx:
    print(f"{non_zero_features[i]}: {non_zero_values[i]:.5f}")

# Get 5 lowest coefficients
bottom_5_idx = np.argsort(non_zero_values)[:5]
print("\nE --> Bottom 5 coefficients:")
for i in bottom_5_idx:
    print(f"{non_zero_features[i]}: {non_zero_values[i]:.5f}")
```

```
C --> Number of non-zero coefficients: 21

D --> Top 5 coefficients:
pipeline-1__v_Overall_Qual: 0.13446
pipeline-1__v_Gr_Liv_Area: 0.09828
pipeline-1__v_Year_Built: 0.06627
pipeline-1__v_Garage_Cars: 0.04761
pipeline-1__v_Overall_Cond: 0.03573

E --> Bottom 5 coefficients:
pipeline-1__v_MS_SubClass: -0.02029
pipeline-1__v_Misc_Val: -0.01789
pipeline-1__v_Kitchen_AbvGr: -0.00402
pipeline-1__v_Pool_Area: -0.00243
pipeline-1__v_Bedroom_AbvGr: 0.00392
```

In [10]:
```python
# Predict on test set
y_pred = final_model.predict(X_test)

# Report R²
r2 = r2_score(y_test, y_pred)
print(f"Test set R^2: {r2:.5f}")
```

```
Test set R^2: 0.86545
```

Model 2

In [11]:

```python
# OUTPUT 1 #

from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer, make_column_selec
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, Polyn
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import HistGradientBoostingRegressor
import numpy as np


# Make Numerical Pipeline
num_pipe = make_pipeline(SimpleImputer(strategy="mean"), StandardScale

# Make Categorical Pipeline
c_pipe   = make_pipeline(OneHotEncoder(drop='first',sparse_output=Fals

# Combine those
pp_pipe = make_column_transformer(
    (num_pipe, make_column_selector(dtype_include=np.number)),
    (c_pipe, make_column_selector(dtype_include=object)),
    remainder="drop",
)

feature = SelectKBest(score_func = f_regression, k = 15)

model = HistGradientBoostingRegressor()

full_pipe = make_pipeline(
    pp_pipe,
    feature,
    model
)

print(full_pipe)
```

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('pipeline-1',
                                                  Pipeline(steps=[('sim
pleimputer',
                                                                   Simp
leImputer()),
                                                                  ('sta
ndardscaler',
                                                                   Stan
dardScaler())]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150ca70b0>),
                                                 ('pipeline-2',
                                                  Pipeline(steps=[('one
hotencoder',
                                                                   OneH
otEncoder(drop='first',

handle_unknown='ignore',

sparse_output=False))]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150ca5580>)])),
                ('selectkbest',
                 SelectKBest(k=15,
                             score_func=<function f_regression at 0x150
753ec0>)),
                ('histgradientboostingregressor',
                 HistGradientBoostingRegressor())])
```

In [12]:

```python
hyperparameters = {
    'selectkbest__k': [5, 10, 15, 20, 25],
    'histgradientboostingregressor__learning_rate': [0.05, 0.075, 0.1,
}

grid_search = GridSearchCV(
    full_pipe,
    hyperparameters,
    cv = 5,
    scoring = 'r2'
)

# Run grid search
grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best R^2 Score: {grid_search.best_score_:.5f}")
```

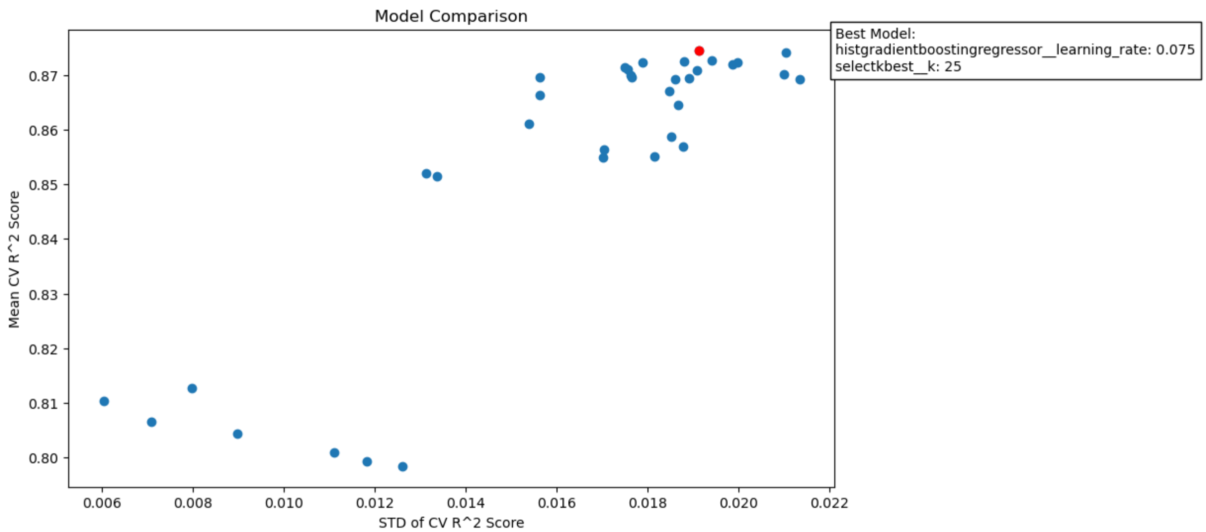
```
Best Parameters: {'histgradientboostingregressor__learning_rate': 0.07
5, 'selectkbest__k': 25}
Best R^2 Score: 0.87451
```

In [13]:

```
# OUTPUT 2 #

# SelectKBest__K:
# My first hyperparameter specifies the number of features (or variabl
# the model will use the top 10 variables with the strongest relations
# because the number of variables in a model has a large impact on the
# use in the model.

# HGBR Learning Rate:
# The machine learning model uses trees to regress the data. Trees are
# The more trees there are, the model is less likely to overfit the da
# controls how much each tree contributes to the overall prediction. I
# the risk of overfitting.
```

In [14]:

```
# OUTPUT 3 #

results = grid_search.cv_results_

# Get means and stds
means = results['mean_test_score']
stds = results['std_test_score']

# Best model
best_ind = np.argmax(means)
best_params = grid_search.best_params_

# Plot
plt.figure(figsize=(10,6))
plt.scatter(stds, means, label = 'Other Models')
plt.scatter(stds[best_ind], means[best_ind], c= 'red', label = 'Best M

# Label best model
param_text = '\n'.join([f'{k}: {v}' for k, v in best_params.items()])
plt.text(stds[best_ind] + 0.003, means[best_ind], f'Best Model:\n{para
         fontsize=10, verticalalignment='center', bbox=dict(facecolor=

# Clean Up
plt.xlabel('STD of CV R^2 Score')
plt.ylabel('Mean CV R^2 Score')
plt.title('Model Comparison')
plt.show()
```



In [15]:
```
# OUTPUT 4 #

# Hyperparameters used in last figure:
    # 'selectkbest__k': [5, 10, 15, 20, 25],
    # 'histgradientboostingregressor__learning_rate': [0.05, 0.075, 0.

# Best set found: k = 25, learning rate = 0.075

new_hyperparameters = {
    'selectkbest__k': range(23,27),
    'histgradientboostingregressor__learning_rate': [0.05, 0.07, 0.075
}

new_grid_search = GridSearchCV(
    full_pipe,
    new_hyperparameters,
    cv = 5,
    scoring = 'r2'
)

# Run grid search
new_grid_search.fit(X_train, y_train)

print(f"Best Parameters: {new_grid_search.best_params_}")
print(f"Best R^2 Score: {new_grid_search.best_score_:.5f}")
```

```
Best Parameters: {'histgradientboostingregressor__learning_rate': 0.07,
'selectkbest__k': 26}
Best R^2 Score: 0.87625
```

In [16]:
```
# OUTPUT 5 #

best_model = new_grid_search.best_estimator_

# Predict on test set
y_bestpred = best_model.predict(X_test)
```

```python
# Calculate R^2
best_r2 = r2_score(y_test, y_bestpred)

print(f'Holdout R^2 score with optimized parameters:{best_r2: .5f}')
print('')
print(f"Best Parameters: {new_grid_search.best_params_}")
```

```
Holdout R^2 score with optimized parameters: 0.85621

Best Parameters: {'histgradientboostingregressor__learning_rate': 0.07,
'selectkbest__k': 26}
```

Model 3

In [17]:

```python
# pipeline
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingRegressor



# Make Numerical Pipeline
num_pipe3 = make_pipeline(SimpleImputer(strategy="mean"), StandardScal

# Make Categorical Pipeline
c_pipe3   = make_pipeline(OneHotEncoder(sparse_output=False, handle_un

# Combine those
pp_pipe3 = make_column_transformer(
    (num_pipe3, make_column_selector(dtype_include=np.number)),
    (c_pipe3, make_column_selector(dtype_include=object)),
    remainder="drop",
)

feature3 = RFECV(GradientBoostingRegressor(), step=1, scoring='r2', im

model3 = HistGradientBoostingRegressor()

full_pipe3 = make_pipeline(
    pp_pipe3,
    feature3,
    model3
)

print(full_pipe3)
```

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('pipeline-1',
                                                  Pipeline(steps=[('sim
pleimputer',
                                                                   Simp
leImputer()),
                                                                  ('sta
ndardscaler',
                                                                   Stan
dardScaler())]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150cdf6e0>),
                                                 ('pipeline-2',
                                                  Pipeline(steps=[('one
hotencoder',
                                                                   OneH
otEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150cddfd0>)])),
                ('rfecv',
                 RFECV(estimator=GradientBoostingRegressor(),
                       importance_getter='feature_importances_',
                       scoring='r2')),
                ('histgradientboostingregressor',
                 HistGradientBoostingRegressor())])
```

In [19]:

```
# hyperparameters3 = {
#     'histgradientboostingregressor__learning_rate': [0.1]
# }

# grid_search3 = GridSearchCV(
#     full_pipe3,
#     hyperparameters3,
#     scoring = 'r2',
#     cv= KFold(5),
#     n_jobs=-1
# )

# # Run grid search
# grid_search3.fit(X_train, y_train)

# print(f"Best Parameters: {grid_search.best_params_}")
# print(f"Best R^2 Score: {grid_search.best_score_:.5f}")
```

Model 4 (Best Model)

In [20]:

```
from sklearn.decomposition import PCA

#exclude parcel from the dataset
```

```python
X_train_new = X_train.drop('parcel', axis=1)



# Make Numerical Pipeline
num_pipe4 = make_pipeline(SimpleImputer(strategy="mean", fill_value =

# Make Categorical Pipeline
c_pipe4   = make_pipeline(SimpleImputer(strategy='most_frequent'), One

# Combine those
pp_pipe4 = ColumnTransformer([
    ('num', num_pipe4, make_column_selector(dtype_include=['int64', 'f
    ('cat', c_pipe4, make_column_selector(dtype_include=['object', 'ca
], remainder="drop")


model4 = HistGradientBoostingRegressor()

full_pipe4 = Pipeline([
    ('preprocessor', pp_pipe4),
    ('model', model4)
])

print(full_pipe4)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('sim
pleimputer',
                                                                   Simp
leImputer(fill_value='missing')),
                                                                  ('sta
ndardscaler',
                                                                   Stan
dardScaler())]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150c89370>),
                                                 ('cat',
                                                  Pipeline(steps=[('sim
pleimputer',
                                                                   Simp
leImputer(strategy='most_frequent')),
                                                                  ('one
hotencoder',
                                                                   OneH
otEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x150c8afc0>)])),
                ('model', HistGradientBoostingRegressor())])
```

In [21]:
```
hyperparameters4 = {
    # 'pca__n_components': [0.97, 0.971, 0.972, 0.973, 0.974],  # Try
    'model__learning_rate': [0.09,0.095, 0.1, 0.105, 0.11],  # Try dif
    'preprocessor__cat__onehotencoder__sparse_output':[False]
}

grid_search4 = GridSearchCV(
    full_pipe4,
    hyperparameters4,
    scoring = 'r2',
    cv= 5,
    n_jobs=-1,
    verbose = 3 # talks to you
)

# Run grid search
grid_search4.fit(X_train_new, y_train)

print(f"Best Parameters: {grid_search4.best_params_}")
print(f"Best R^2 Score: {grid_search4.best_score_:.5f}")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best Parameters: {'model__learning_rate': 0.1, 'preprocessor__cat__oneh
otencoder__sparse_output': False}
Best R^2 Score: 0.90375
```

In [22]:
```
best_model4 = grid_search4.best_estimator_

# Predict on test set
y_bestpred4 = best_model4.predict(X_test)

# Calculate R^2
best_r2_4 = r2_score(y_test, y_bestpred4)

print(f'Holdout R^2 score with optimized parameters:{best_r2_4: .5f}')
print('')
print(f"Best Parameters: {grid_search4.best_params_}")
```

```
Holdout R^2 score with optimized parameters: 0.89556

Best Parameters: {'model__learning_rate': 0.1, 'preprocessor__cat__oneh
otencoder__sparse_output': False}
```

Model 5

In [23]:
```
from sklearn.linear_model import LinearRegression
# Make Numerical Pipeline
num_pipe5 = make_pipeline(SimpleImputer(strategy="mean"), StandardScal

# Make Categorical Pipeline
c_pipe5   = make_pipeline(OneHotEncoder(sparse_output=False, handle_un
```

```python
# Combine those
pp_pipe5 = make_column_transformer(
    (num_pipe5, make_column_selector(dtype_include=np.number)),
    (c_pipe5, make_column_selector(dtype_include=object)),
    remainder="drop",
)

feature5 = PCA(n_components = 0.95)

model5 = LinearRegression()

full_pipe5 = make_pipeline(
    pp_pipe5,
    feature5,
    model5
)

print(full_pipe5)
```

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('pipeline-1',
                                                  Pipeline(steps=[('sim
pleimputer',
                                                                   Simp
leImputer()),
                                                                  ('sta
ndardscaler',
                                                                   Stan
dardScaler())]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x15294bb60>),
                                                 ('pipeline-2',
                                                  Pipeline(steps=[('one
hotencoder',
                                                                   OneH
otEncoder(handle_unknown='ignore',

sparse_output=False))]),
                                                  <sklearn.compose._col
umn_transformer.make_column_selector object at 0x15294ba40>)])),
                ('pca', PCA(n_components=0.95)),
                ('linearregression', LinearRegression())])
```

In [ ]:

```python
hyperparameters5 = {
    'pca__n_components': [0.0975, 0.98, 0.985, 0.99, 0.995],  # Try di
    'linearregression__fit_intercept': [True, False],
    'linearregression__copy_X': [True, False],  # Try different learni
}

grid_search5 = GridSearchCV(
    full_pipe5,
    hyperparameters5,
    scoring = 'r2',
```

```python
    cv= KFold(5),
    n_jobs=-1
)

# Run grid search
grid_search5.fit(X_train, y_train)

print(f"Best Parameters: {grid_search5.best_params_}")
print(f"Best R^2 Score: {grid_search5.best_score_:.5f}")
```

In [ ]:
```python
best_model5 = grid_search5.best_estimator_

# Predict on test set
y_bestpred5 = best_model5.predict(X_test)

# Calculate R^2
best_r2_5 = r2_score(y_test, y_bestpred5)

print(f'Holdout R^2 score with optimized parameters:{best_r2_5: .5f}')
print('')
print(f"Best Parameters: {grid_search5.best_params_}")
```

Best Model is Model 4, so I will predict Sales based off that

In [ ]:
```python
# create predictions
holdout = pd.read_csv('input_data2/housing_holdout.csv') # load the ne

holdout_X_vals = holdout.drop('parcel', axis = 1) # remove the parcel

y_pred = best_model4.predict(holdout_X_vals) # make predictions!

# save for output: parcel number + y_pred to csv
df_out = pd.DataFrame({'parcel':holdout['parcel'],
                       'prediction':y_pred})

df_out.to_csv('submission/MY_PREDICTIONS.csv',index=False)

# open it... does it look like the sample version?
```