linear_regression

August 5, 2024

0.0.1 ML-A1 Implementation of Linear regression

- Instructions
  - Prepare a report to present your findings
  - Write a python code to implement stochastic gradient descent from scratch for the given house price prediction dataset.
  - Write a python code to implement stochastic gradient descent using scikit-learn for the given data and compare the output.
  - Write a python code to implement batch gradient descent from scratch and also using scikit-lean for the given house price prediction.
  - Compare the output of all the implementations and write conclusion.

  Dataset: House Price Prediction Challenge (kaggle.com)

- Submission Intruction:
  - Submission should include python notebook file for all the implementations.
  - There must be a report pdf file to illustrate your data science lifecycle implementation and present your finds. Report must not exceed 10 page or 1500 words.

Table of Content

1. Problem Statement
2. Data Understanding
3. Data Preparation
4. Modeling
   4.1. SGD using scikit-learn
   4.2 Batch Gradient Descent using scikit-learn
   4.3 SGD Scratch Implementation
   4.4 Batch Gradient Descent Scratch Implementation

Data Ingestion/Loading

- Load the necessary modules
- Load the data
- process the data

[ ]:
```
# loading the library
import numpy as np
import pandas as pd
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

plt.rcParams['figure.figsize'] = (18,6)


np.random.seed(32)
# data path
train_data  = "/home/suman/Applied-Machine-Learning/Linear Regression/train.csv"
test_data = "/home/suman/Applied-Machine-Learning/Linear Regression/test.csv"

# load the train and test data
df_train  = pd.read_csv(train_data)
df_test = pd.read_csv(test_data)

df_train.head()
```

```
[ ]:   POSTED_BY  UNDER_CONSTRUCTION  RERA  BHK_NO. BHK_OR_RK    SQUARE_FT  \
0      Owner                   0     0        2       BHK  1300.236407
1     Dealer                   0     0        2       BHK  1275.000000
2      Owner                   0     0        2       BHK   933.159722
3      Owner                   0     1        2       BHK   929.921143
4     Dealer                   1     0        2       BHK   999.009247

   READY_TO_MOVE  RESALE                      ADDRESS  LONGITUDE   LATITUDE  \
0              1       1        Ksfc Layout,Bangalore  12.969910  77.597960
1              1       1    Vishweshwara Nagar,Mysore  12.274538  76.644605
2              1       1             Jigani,Bangalore  12.778033  77.632191
3              1       1  Sector-1 Vaishali,Ghaziabad  28.642300  77.344500
4              0       1             New Town,Kolkata  22.592200  88.484911

   TARGET(PRICE_IN_LACS)
0                   55.0
1                   51.0
2                   43.0
3                   62.5
4                   60.5
```

Data Understanding and Exploration The dataset used for this assignmetn is from Kaggle Dataset: House Price Prediction Challenge (kaggle.com)

- **Training Splits:** 29451 rows x 12 columns
- **Testing Splits:** 68720 x 11 columns

– since we are using competion data testing data do not contain the labels, they are evaluated based on this splits.

- **Attributes of the Dataset**

Column	Description
POSTED_BY	Category marking who has listed the property
UNDER_CONSTRUCTION	Under Construction or Not
RERA	Rera approved or Not
BHK_NO	Number of Rooms
BHK_OR_RK	Type of property
SQUARE_FT	Total area of the house in square feet
READY_TO_MOVE	Category marking Ready to move or Not
RESALE	Category marking Resale or not
ADDRESS	Address of the property
LONGITUDE	Longitude of the property
LATITUDE	Latitude of the property

RERA stands for Real Estate (Regulation and Development) Act, which was enacted by the Indian government in 2016. It aims to protect home buyers and ensure transparency in the real estate sector. RERA establishes regulatory authorities at the state level to oversee real estate transactions and address grievances.

[ ]:
```
print(f'The train dataset contains {df_train.shape[0]} rows and {df_train.
↪shape[1]} columns.')
print(f'The test dataset contains {df_test.shape[0]} rows and {df_test.
↪shape[1]} columns.')
```

```
The train dataset contains 29451 rows and 12 columns.
The test dataset contains 68720 rows and 11 columns.
```

[ ]:
```
# info of the dateset
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29451 entries, 0 to 29450
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   POSTED_BY              29451 non-null  object
 1   UNDER_CONSTRUCTION     29451 non-null  int64
 2   RERA                   29451 non-null  int64
 3   BHK_NO.                29451 non-null  int64
 4   BHK_OR_RK              29451 non-null  object
 5   SQUARE_FT              29451 non-null  float64
 6   READY_TO_MOVE          29451 non-null  int64
 7   RESALE                 29451 non-null  int64
 8   ADDRESS                29451 non-null  object
```

```
 9   LONGITUDE              29451 non-null  float64
 10  LATITUDE               29451 non-null  float64
 11  TARGET(PRICE_IN_LACS)  29451 non-null  float64
dtypes: float64(4), int64(5), object(3)
memory usage: 2.7+ MB
```

- since we are solving linear regression problem, the target or dependent variable must be continous and here we can see that it is continous
- rest we can see that there are total of three dtypes
- there are two categorical variable which are useful: `BHK_OR_RK` and `POSTED_BY`

Exploration and Descriptive Statistics

[ ]:
```
df_train.describe()
```

[ ]:
```
       UNDER_CONSTRUCTION          RERA       BHK_NO.     SQUARE_FT  \
count        29451.000000  29451.000000  29451.000000  2.945100e+04
mean             0.179756      0.317918      2.392279  1.980217e+04
std              0.383991      0.465675      0.879091  1.901335e+06
min              0.000000      0.000000      1.000000  3.000000e+00
25%              0.000000      0.000000      2.000000  9.000211e+02
50%              0.000000      0.000000      2.000000  1.175057e+03
75%              0.000000      1.000000      3.000000  1.550688e+03
max              1.000000      1.000000     20.000000  2.545455e+08

       READY_TO_MOVE        RESALE     LONGITUDE      LATITUDE  \
count   29451.000000  29451.000000  29451.000000  29451.000000
mean        0.820244      0.929578     21.300255     76.837695
std         0.383991      0.255861      6.205306     10.557747
min         0.000000      0.000000    -37.713008   -121.761248
25%         1.000000      1.000000     18.452663     73.798100
50%         1.000000      1.000000     20.750000     77.324137
75%         1.000000      1.000000     26.900926     77.828740
max         1.000000      1.000000     59.912884    152.962676

       TARGET(PRICE_IN_LACS)
count           29451.000000
mean              142.898746
std               656.880713
min                 0.250000
25%                38.000000
50%                62.000000
75%               100.000000
max             30000.000000
```

[ ]:
```
# categorical data

df_train.describe(exclude=["float", "int"])
```

[ ]:
```
       POSTED_BY BHK_OR_RK              ADDRESS
count      29451     29451                29451
unique         3         2                 6899
top       Dealer       BHK  Zirakpur,Chandigarh
freq       18291     29427                  509
```

[ ]:
```
# check for null values
df_train.isnull().sum()
```

[ ]:
```
POSTED_BY               0
UNDER_CONSTRUCTION      0
RERA                    0
BHK_NO.                 0
BHK_OR_RK               0
SQUARE_FT               0
READY_TO_MOVE           0
RESALE                  0
ADDRESS                 0
LONGITUDE               0
LATITUDE                0
TARGET(PRICE_IN_LACS)   0
dtype: int64
```

Interpretations:

- there are numerical and categorical variables
- the dataset have no missing records (since this is competition data it is already been curated)
- target/depedent variable is continous (as float dtype)

Basic EDA

[ ]:
```
df_train['POSTED_BY'].value_counts()
```

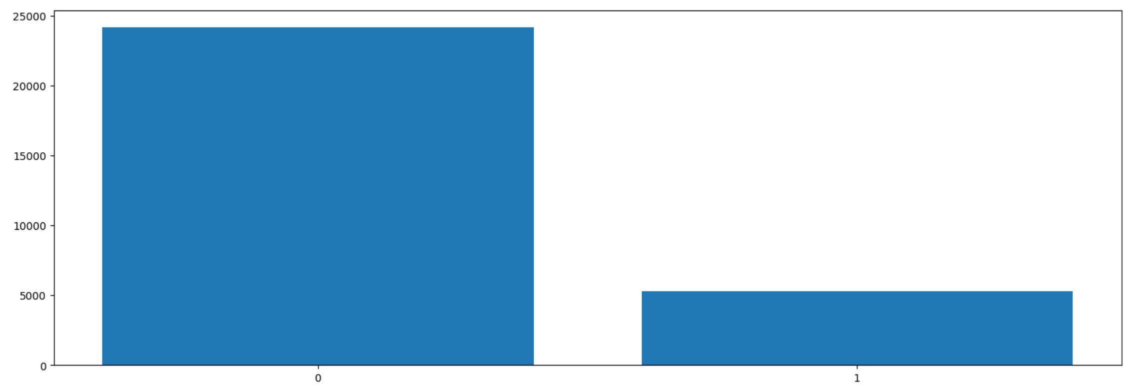
[ ]:
```
POSTED_BY
Dealer     18291
Owner      10538
Builder      622
Name: count, dtype: int64
```

[ ]:
```
df_train['UNDER_CONSTRUCTION'].value_counts()
```

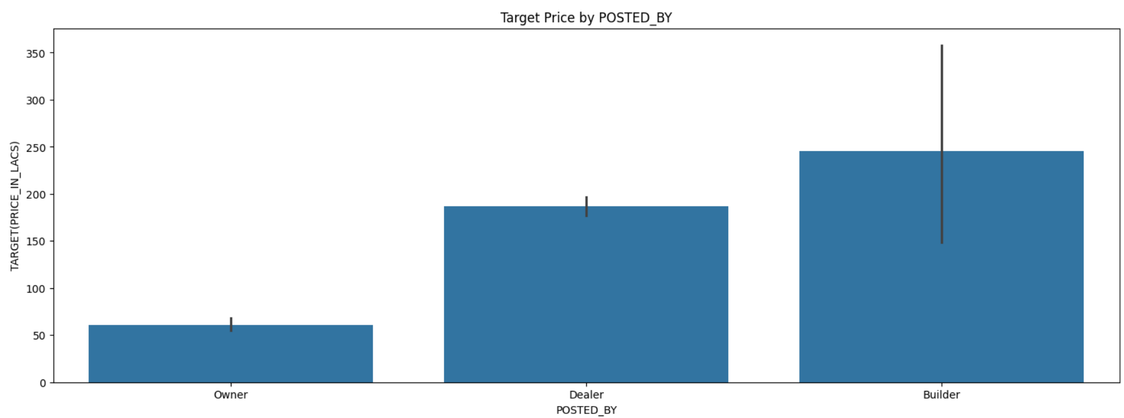
[ ]:
```
UNDER_CONSTRUCTION
0    24157
1     5294
Name: count, dtype: int64
```

[ ]:
```
plt.bar(["0","1"],df_train["UNDER_CONSTRUCTION"].value_counts())
```

[ ]: <BarContainer object of 2 artists>

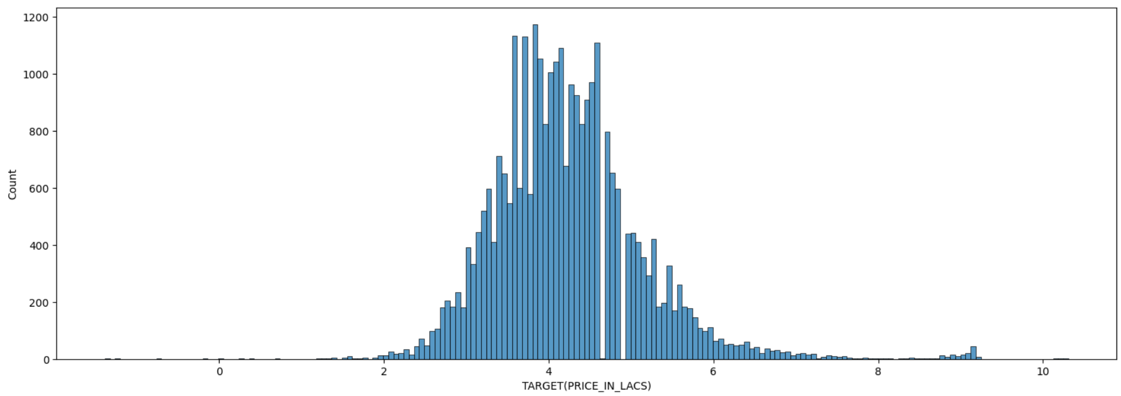


[ ]:
```
sns.barplot(x='POSTED_BY', y='TARGET(PRICE_IN_LACS)', data=df_train)
plt.title('Target Price by POSTED_BY')
plt.show()

# looks like majority of the property listinga re made dealers
```



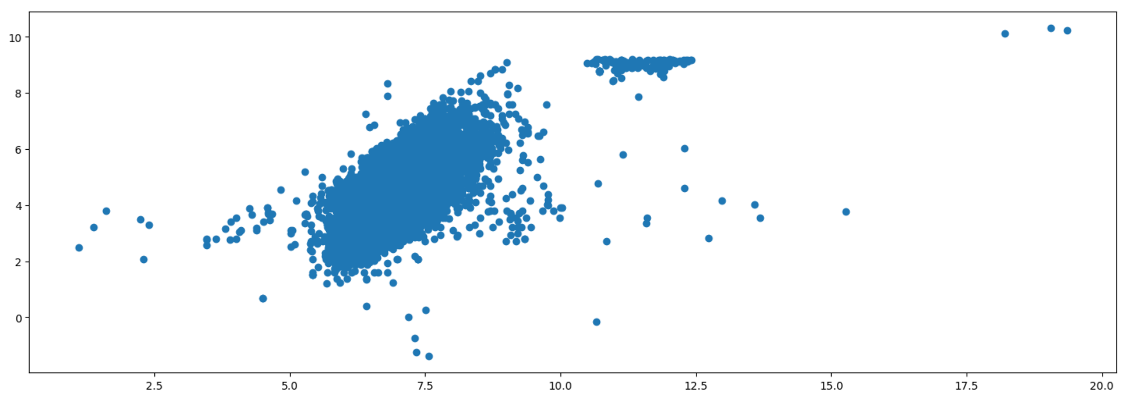
[ ]:
```
sns.histplot(np.log(df_train["TARGET(PRICE_IN_LACS)"]))
```

[ ]: <Axes: xlabel='TARGET(PRICE_IN_LACS)', ylabel='Count'>



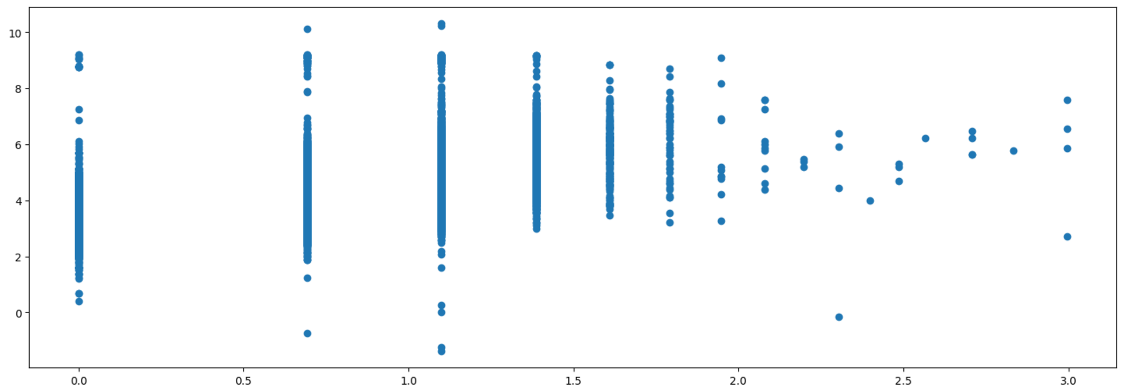
[ ]:
```
# check the relationship between square ft and the price
# does sqaure ft influences price?
plt.scatter(x=np.log(df_train["SQUARE_FT"]), y=np.
 ↪log(df_train["TARGET(PRICE_IN_LACS)"]))

# looks like it does
```

[ ]: <matplotlib.collections.PathCollection at 0x7fcfdf4382c0>



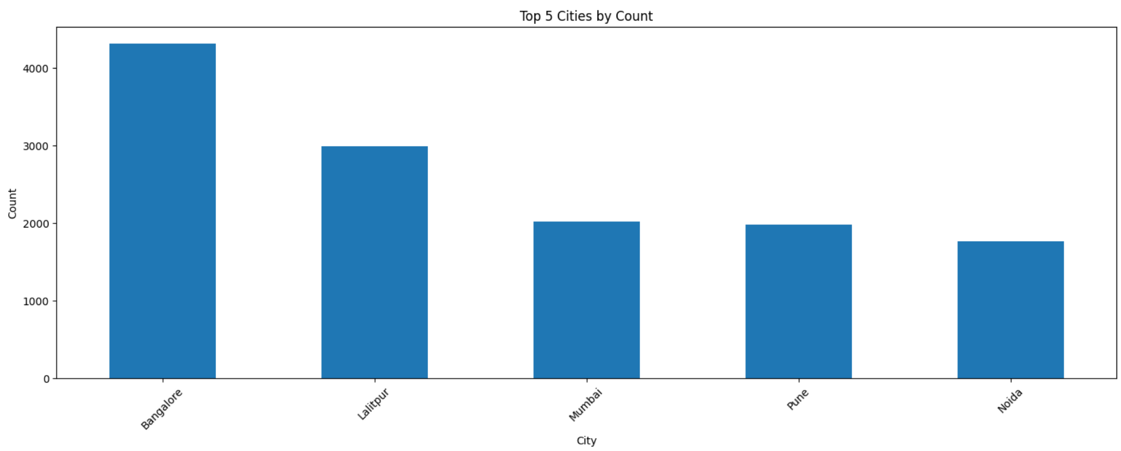
[ ]:
```
plt.scatter(x=np.log(df_train["BHK_NO."]), y=np.
 ↪log(df_train["TARGET(PRICE_IN_LACS)"]))
```

[ ]: <matplotlib.collections.PathCollection at 0x7fcfdf524680>




```
[ ]: # which city is most popular?
new_df = df_train['ADDRESS'].str.split(',').str.get(1)
city_counts = new_df.value_counts().head(5)
```

```
[ ]: city_counts.plot(kind='bar')
plt.title('Top 5 Cities by Count')
plt.xlabel('City')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```




```
[ ]: # make a copy so that we need not have to restart the notebook should we mess␣
↪up the data.
df_train_copy = df_train.copy()
```

Data Preprocessing

- Check Missing Data

  - since there is no missing data, we skip this part.
- Check any redundant data, if present drop them
- Standardization/Normalization
- Encoding the Categorical Variables

[ ]:
```
df_train.duplicated().sum()
print(f"There are {df_train[df_train.duplicated()].shape[0]} duplicates in␣
↪training dataset.")
```

```
There are 401 duplicates in training dataset.
```

[ ]:
```
# let's drop the duplicates
df_train.drop_duplicates(inplace=True)
```

[ ]:
```
df_train.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
# now let's remove the uwanted columns, as they do
df_train = df_train.drop(['ADDRESS'], axis=1)
```

Encoding Categorical Variable

[ ]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer

# Label encoding function
def label_encoder(X):
    X_transformed = X.copy()
    for column in X.columns:
        le = LabelEncoder()
        X_transformed[column] = le.fit_transform(X[column])
    return X_transformed
```

[ ]:
```
# Define the preprocessing steps for categorical features
categorical_features = ['POSTED_BY', 'BHK_OR_RK']
categorical_transformer = Pipeline(steps=[
    ('label_encoder', FunctionTransformer(label_encoder, validate=False))
])
```

Standardizing

[ ]:
```
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
# Define the preprocessing steps for numerical features
numerical_features_standard = ['BHK_NO.', 'SQUARE_FT', 'LONGITUDE', 'LATITUDE']
```

```
numerical_transformer_standard = Pipeline(steps=[
    ('standard_scaler', StandardScaler())
])
```

[ ]:
```
# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('categorical', categorical_transformer, categorical_features),
        ('numerical', numerical_transformer_standard,␣
 ↪numerical_features_standard)
    ],
    remainder='passthrough'
)
```

Pipelining

[ ]:
```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
```

[ ]:
```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SGDRegressor(max_iter=1, tol=None, warm_start=True))
])
```

[ ]:
```
LABELS = 'TARGET(PRICE_IN_LACS)'
train_features = [col for col in df_train.columns if col not in [LABELS]]
train_data = df_train[train_features]
train_labels = df_train[LABELS]
```

Splitting the dataset

[ ]:
```
train_data, valid_data, train_labels, valid_labels =␣
 ↪train_test_split(train_data, train_labels, random_state=42, test_size = 0.2)

print(f"The shape of training dataset is: {train_data.shape}")
print(train_data.shape)


print(f"The shape of training dataset is: {train_data.shape}")
print(valid_data.shape)
```

```
The shape of training dataset is: (23240, 10)
(23240, 10)
The shape of training dataset is: (23240, 10)
(5810, 10)
```

[ ]:
```python
test_features = [col for col in df_test.columns if col not in [LABELS]]
test_data = df_test[test_features]
```

Finding the best hyperparameters

[ ]:
```python
# initial search
param_grid = {
    'regressor__loss': ['squared_error'],
    'regressor__penalty': ['l2', 'l1'],
    'regressor__alpha': [0.0001, 0.001, 0.01],
    'regressor__l1_ratio': [0.0, 0.1, 0.5],
    'regressor__eta0': [0.001, 0.01, 0.1],
}
```

[ ]:
```python
from sklearn.model_selection import GridSearchCV
```

[ ]:
```python
grid_search = GridSearchCV(pipeline, param_grid, cv=5,
                           scoring='neg_mean_squared_error', n_jobs=-1)

grid_search.fit(train_data, train_labels)
```

```
/home/suman/.conda/envs/documentai/lib/python3.12/site-
packages/sklearn/compose/_column_transformer.py:1623: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(remainder='passthrough',
transformers=[('categorical',
Pipeline(steps=[('label_encoder',
          FunctionTransformer(func=<function label_encoder at
0x7fcfdf3d79c0>))]),
['POSTED_BY',
'BHK_OR_RK']),
('numerical',
Pipeline(steps=[('standard_scaler',
          StandardScaler())]),
['BHK_NO.',
```

```
'SQUARE_FT',
'LONGITUDE',
'LATITUDE'])])),
                                        ('regressor',
                                         SGDRegressor(max_iter=1, tol=None,
                                                      warm_start=True))]),
             n_jobs=-1,
             param_grid={'regressor__alpha': [0.0001, 0.001, 0.01],
                         'regressor__eta0': [0.001, 0.01, 0.1],
                         'regressor__l1_ratio': [0.0, 0.1, 0.5],
                         'regressor__loss': ['squared_error'],
                         'regressor__penalty': ['l2', 'l1']},
             scoring='neg_mean_squared_error')
```

[ ]:
```python
best_parameters = grid_search.best_params_
best_model = grid_search.best_estimator_

print(f'The best model is : {best_model} with parameters {best_parameters}')
```

```
The best model is : Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('categorical',
Pipeline(steps=[('label_encoder',
FunctionTransformer(func=<function label_encoder at 0x7fcfdf3d79c0>))]),
                                                 ['POSTED_BY', 'BHK_OR_RK']),
                                                ('numerical',
Pipeline(steps=[('standard_scaler',
StandardScaler())]),
                                                 ['BHK_NO.', 'SQUARE_FT',
                                                  'LONGITUDE',
                                                  'LATITUDE'])])),
                ('regressor',
                 SGDRegressor(alpha=0.001, eta0=0.001, l1_ratio=0.5, max_iter=1,
                              tol=None, warm_start=True))]) with parameters
{'regressor__alpha': 0.001, 'regressor__eta0': 0.001, 'regressor__l1_ratio':
0.5, 'regressor__loss': 'squared_error', 'regressor__penalty': 'l2'}
```

[ ]:
```python
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SGDRegressor(alpha= 0.0001, eta0= 0.001, l1_ratio = 0.0,
                               loss= 'squared_error', penalty= 'l1',
                               tol= 0.01, max_iter=2000, warm_start=True))
])
```

Fit the model with best parameters and estimate

[ ]:
```python
pipeline.fit(train_data, train_labels)
```

```
/home/suman/.conda/envs/documentai/lib/python3.12/site-
packages/sklearn/compose/_column_transformer.py:1623: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('categorical',
Pipeline(steps=[('label_encoder',
FunctionTransformer(func=<function label_encoder at 0x7fcfdf3d79c0>))]),
                                                  ['POSTED_BY', 'BHK_OR_RK']),
                                                 ('numerical',
Pipeline(steps=[('standard_scaler',
StandardScaler())]),
                                                  ['BHK_NO.', 'SQUARE_FT',
                                                   'LONGITUDE',
                                                   'LATITUDE'])])),
                ('regressor',
                 SGDRegressor(eta0=0.001, l1_ratio=0.0, max_iter=2000,
                              penalty='l1', tol=0.01, warm_start=True))])
```

[ ]:
```
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SGDRegressor(alpha= 0.0001, eta0= 0.001, l1_ratio = 0.0,
                               loss= 'squared_error', penalty= 'l1',
                               tol= 0.01, max_iter=2000, warm_start=True))
])
```

[ ]:
```
pipeline.fit(train_data, train_labels)
```

```
/home/suman/.conda/envs/documentai/lib/python3.12/site-
packages/sklearn/compose/_column_transformer.py:1623: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).
```

```
  warnings.warn(
```

```
[ ]: Pipeline(steps=[('preprocessor',
                  ColumnTransformer(remainder='passthrough',
                                    transformers=[('categorical',
Pipeline(steps=[('label_encoder',
FunctionTransformer(func=<function label_encoder at 0x7fcfdf3d79c0>))]),
                                                   ['POSTED_BY', 'BHK_OR_RK']),
                                                  ('numerical',
Pipeline(steps=[('standard_scaler',
StandardScaler())]),
                                                   ['BHK_NO.', 'SQUARE_FT',
                                                    'LONGITUDE',
                                                    'LATITUDE'])])),
                 ('regressor',
                  SGDRegressor(eta0=0.001, l1_ratio=0.0, max_iter=2000,
                               penalty='l1', tol=0.01, warm_start=True))])
```

Evaluation of the model

```
[ ]: y_hat = pipeline.predict(valid_data)
     print(f'Mean Squared Error: {mean_squared_error(valid_labels, y_hat)}')
```

```
Mean Squared Error: 378435.8539077567
```

```
[ ]: print(f'Mean Absolute Error: {mean_absolute_error(valid_labels, y_hat)}')
```

```
Mean Absolute Error: 135.96579072419206
```

```
[ ]: print(f'Root Mean Square Error: {np.sqrt(mean_squared_error(valid_labels, y_hat))}')
```

```
Root Mean Square Error: 615.1714020561723
```

```
[ ]: print(f'Variance captured by the model is : {r2_score(valid_labels, y_hat)}')
```

```
Variance captured by the model is : 0.3088774233519437
```

Prediction

```
[ ]: test_pred = pipeline.predict(test_data)
```

```
[ ]: neg_index = np.where(test_pred < 0)
     test_pred[neg_index] = np.abs(test_pred[neg_index])
```

```
[ ]: test_pred
```

```
[ ]: array([ 39.31517246, 478.74386283,  44.85543851, …, 394.81795119,
             84.4035233 , 170.04528112])
```