```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import VotingRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from category_encoders import CatBoostEncoder
import matplotlib.pyplot as plt

from xgboost import XGBRegressor
from catboost import CatBoostRegressor
import optuna
```

```
train_dataset = pd.read_csv('/kaggle/input/home-data-for-ml-course/train.csv')
train_dataset.head()
test_dataset = pd.read_csv('/kaggle/input/home-data-for-ml-course/test.csv')
test_dataset.head()
```

Feature Selection

In [4]:

```
X = train_dataset.drop(columns=['SalePrice', 'Id'])
y = train_dataset['SalePrice']
X.head()

test_df = test_dataset.drop(columns=['Id'])
test_Id = test_dataset['Id']
test_Id.head()
```

Missing Values

In [6]:

```
numerical_columns = X.select_dtypes(include=[np.number]).columns
categorical_columns = X.select_dtypes(include=['object']).columns

X[numerical_columns] = X[numerical_columns].fillna(-1)
X[categorical_columns] = X[categorical_columns].fillna('No Attribute')

test_df[numerical_columns] = test_df[numerical_columns].fillna(-1)
test_df[categorical_columns] = test_df[categorical_columns].fillna('No Attribute')
```

Splting The Training Set

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Defining Column Transformers¶

In [8]:

```
numeric_transformers = Pipeline(steps=[('scaling', StandardScaler())])
category_transformers = Pipeline(steps=[('catboosting',
CatBoostEncoder(cols=categorical_columns, random_state=0))])
preprocessor = ColumnTransformer(transformers=[('num', numeric_transformers,
numerical_columns),
                                               ('cat', category_transformers,
categorical_columns)
                                             ])
```

Defining Hyperparameters

In [9]:

```
def objective(trial):
    xgb_params = {
        "learning_rate": trial.suggest_float("xgb_learning_rate",0.0001,0.1, log=True),
        "max_depth": trial.suggest_int("xgb_max_depth",3,12),
        "subsample": trial.suggest_float("xgb_subsample",0.5,1.0),
        "colsample_bytree": trial.suggest_float("xgb_colsample_bytree",0.5,1.0),
        "n_estimators": trial.suggest_int("xgb_n_estimators",50,300),
    }

    cat_params = {
        "learning_rate": trial.suggest_float("cat_learning_rate",0.0001,0.1, log=True),
        "depth": trial.suggest_int("cat_depth",3,10),
        "iterations": trial.suggest_int("cat_iterations",100,500),
        "l2_leaf_reg": trial.suggest_float("cat_l2_leaf_reg",0.0001,0.1, log=True),
        "subsample": trial.suggest_float("cat_subsample",0.5,1.0),
        "random_strength": trial.suggest_float("cat_random_strength",0.0001,0.1),

    }

    xgb = XGBRegressor(**xgb_params, objective='reg:squarederror')
    cat = CatBoostRegressor(**cat_params, loss_function='RMSE', verbose=0)



    pipeline = Pipeline([('preprocessor', preprocessor),
                          ('voting_regressor', VotingRegressor([('xgb', xgb), ('cat', cat)]))
                         ])

    score = cross_val_score(pipeline, X_train, y_train, cv=5,
scoring='neg_mean_squared_error').mean()

    return score
```

Running Optuna

In [10]:

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=30)

best_params = study.best_params
print(best_params)
```

Training The Model

In [11]:

```python
best_xgb_params = {
        "learning_rate": study.best_params['xgb_learning_rate'],
        "max_depth": study.best_params['xgb_max_depth'],
        "subsample": study.best_params['xgb_subsample'],
        "colsample_bytree": study.best_params['xgb_colsample_bytree'],
        "n_estimators": study.best_params['xgb_n_estimators'],
}
best_cat_params = {
        "learning_rate": study.best_params['cat_learning_rate'],
        "depth": study.best_params['cat_depth'],
        "iterations": study.best_params['cat_iterations'],
        "l2_leaf_reg": study.best_params['cat_l2_leaf_reg'],
        "subsample": study.best_params['cat_subsample'],
        "random_strength": study.best_params['cat_random_strength'],
}

xgb_2 = XGBRegressor(**best_xgb_params, objective='reg:squarederror')
cat_2 = CatBoostRegressor(**best_cat_params, loss_function='RMSE', verbose=0)

pipeline_2 = Pipeline([('preprocessor', preprocessor),
                        ('voting_regressor', VotingRegressor([('xgb', xgb_2), ('cat', cat_2)]))
                           ])
pipeline_2.fit(X_train,y_train)
test_score = pipeline_2.score(X_test,y_test)
print(f"The Model Accuracy is {test_score}")
```

Making Predictions

In [12]:

```python
y_pred = pipeline_2.predict(test_df)

# mse = mean_squared_error(y_test, y_pred)
# mae = mean_absolute_error(y_test, y_pred)
# r2 = r2_score(y_test, y_pred)
# print(f"Mean Squared Error is: {mse}")
# print(f"Mean Average Error is: {mae}")
# print(f"r2 score: {r2}")
result = pd.DataFrame()
result['Id'] = test_Id
result['SalePrice'] = y_pred
result.to_csv('submission.csv', index=False)
```