```
import pandas as pd
import numpy as np
import optuna
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor,
ExtraTreesRegressor
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import umap

# Load the datasets

train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# Retain the 'row ID' column for the submission file

row_ids = test['row ID']
test = test.drop(columns=["row ID"])

# Data Cleaning

def clean_data(df):

    # Handle missing values for numerical features

    numeric_features = df.select_dtypes(include=[np.number]).columns
    for col in numeric_features:
        df[col] = df[col].fillna(df[col].median())

    # Categorical features: Fill missing with 'unknown'

    categorical_features = df.select_dtypes(include=[object]).columns
    for col in categorical_features:
        df[col] = df[col].fillna('unknown')

    # Remove potential outliers in `price_doc` if present

    if 'price_doc' in df.columns:
        df = df[(df['price_doc'] > 1e5) & (df['price_doc'] < 1e8)]
    return df

train = clean_data(train)
test = clean_data(test)

# Feature Engineering

def feature_engineering(df):

    # Add ratios and interaction features
    if 'full_sq' in df.columns and 'life_sq' in df.columns:
        df['full_sq_ratio_life_sq'] = df['full_sq'] / (df['life_sq'] +
1)

    if 'raion_popul' in df.columns:
        df['green_zone_ratio'] = df['green_zone_part'] /
(df['raion_popul'] + 1)
        df['indust_zone_ratio'] = df['indust_part'] /
(df['raion_popul'] + 1)

    # Add age-based features

    if 'young_all' in df.columns and 'ekder_all' in df.columns:
        df['dependency_ratio'] = (df['young_all'] + df['ekder_all']) /
df['work_all']
        df['elderly_ratio'] = df['ekder_all'] / df['raion_popul']

    # Add transportation accessibility features

    if 'metro_km_avto' in df.columns and 'railroad_station_avto_km' in
df.columns:
        df['transport_accessibility'] = (df['metro_km_avto'] +
df['railroad_station_avto_km']) / 2

    # Add healthcare accessibility features

    if 'healthcare_centers_raion' in df.columns:
        df['healthcare_density'] = df['healthcare_centers_raion'] /
(df['raion_popul'] + 1)

    # Feature interactions for education

    if 'school_education_centers_raion' in df.columns and
'preschool_education_centers_raion' in df.columns:
        df['education_access'] = df['school_education_centers_raion']
+ df['preschool_education_centers_raion']

    # Add density-based features

    if 'area_m' in df.columns:
        df['population_density'] = df['raion_popul'] / (df['area_m'] +
1)

    # Encode categorical features

    categorical_features = df.select_dtypes(include=[object]).columns
    df = pd.get_dummies(df, columns=categorical_features,
drop_first=True)

    return df
```

```
train = feature_engineering(train)
test = feature_engineering(test)

# Separate the target variable

target = train.pop("price_doc")

# Handle Missing Values

imputer = SimpleImputer(strategy="median")
train_imputed = imputer.fit_transform(train)
test_imputed = imputer.transform(test)

# Standardization

scaler = StandardScaler()
train_scaled = scaler.fit_transform(train_imputed)
test_scaled = scaler.transform(test_imputed)

# Step 5: Dimensionality Reduction using UMAP

umap_reducer = umap.UMAP(n_neighbors=15, min_dist=0.1,
n_components=20, random_state=42)
train_reduced = umap_reducer.fit_transform(train_scaled)
test_reduced = umap_reducer.transform(test_scaled)

# Split the data

X_train, X_val, y_train, y_val = train_test_split(train_reduced,
target, test_size=0.2, random_state=42)

# Hyperparameter Optimization with Optuna

def objective_rf(trial):
    n_estimators = trial.suggest_int('n_estimators', 100, 1000)
    max_depth = trial.suggest_int('max_depth', 10, 100)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 20)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 20)
    max_features = trial.suggest_categorical('max_features', ['auto',
'sqrt', 'log2'])
    bootstrap = trial.suggest_categorical('bootstrap', [True, False])

    model = RandomForestRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        bootstrap=bootstrap,
        random_state=42,
        n_jobs=-1
    )

    scores = cross_val_score(model, X_train, y_train, cv=5,
scoring='neg_root_mean_squared_error')
    return -1 * np.mean(scores)

def objective_ext(trial):
    n_estimators = trial.suggest_int('n_estimators', 100, 1000)
    max_depth = trial.suggest_int('max_depth', 10, 100)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 20)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 20)
    max_features = trial.suggest_categorical('max_features', ['auto',
'sqrt', 'log2'])
    max_leaf_nodes = trial.suggest_int('max_leaf_nodes', 10, 500)
    min_impurity_decrease =
trial.suggest_float('min_impurity_decrease', 0.0, 1.0)
    bootstrap = trial.suggest_categorical('bootstrap', [True, False])
    criterion = trial.suggest_categorical('criterion',
['squared_error', 'absolute_error'])

    model = ExtraTreesRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        max_features=max_features,
        max_leaf_nodes=max_leaf_nodes,
        min_impurity_decrease=min_impurity_decrease,
        bootstrap=bootstrap,
        criterion=criterion,
        random_state=42,
        n_jobs=-1
    )

    scores = cross_val_score(model, X_train, y_train, cv=5,
scoring='neg_root_mean_squared_error')
    return -1 * np.mean(scores)

# Optimize Random Forest

study_rf = optuna.create_study(direction='minimize')
study_rf.optimize(objective_rf, n_trials=50)
print("Best RF Params:", study_rf.best_params)

# Optimize Extra Trees

study_ext = optuna.create_study(direction='minimize')
study_ext.optimize(objective_ext, n_trials=50)
print("Best Extra Trees Params:", study_ext.best_params)
```

```python
# Train the final Random Forest model

rf_model = RandomForestRegressor(
    **study_rf.best_params,
    random_state=42,
    n_jobs=-1
)
rf_model.fit(X_train, y_train)

# Evaluate the RF model

y_pred_val_rf = rf_model.predict(X_val)
rmse_rf = np.sqrt(mean_squared_error(y_val, y_pred_val_rf))
print(f"Validation RMSE (RF): {rmse_rf}")

# Train the final Extra Trees model

ext_model = ExtraTreesRegressor(
    **study_ext.best_params,
    random_state=42,
    n_jobs=-1
)
ext_model.fit(X_train, y_train)

# Evaluate the Extra Trees model

y_pred_val_ext = ext_model.predict(X_val)
rmse_ext = np.sqrt(mean_squared_error(y_val, y_pred_val_ext))
print(f"Validation RMSE (Extra Trees): {rmse_ext}")

# Make Predictions and Save Submission

best_model = rf_model if rmse_rf < rmse_ext else ext_model

test_predictions = best_model.predict(test_reduced)
submission = pd.DataFrame({
    'row ID': row_ids,
    'price_doc': test_predictions
})
submission.to_csv('submission_optimized.csv', index=False)
print("Predictions saved to 'submission_optimized.csv'")
```