In [ ]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import optuna
```

I removed the `Material` and `Id` columns since the former is not numerical and the latter is not informative.

In [ ]:
```python
# Load data
X_raw = pd.read_csv('train_X.csv')
y_raw = pd.read_csv('train_y.csv')

# Drop all non-numeric features
X = X_raw.drop(columns=['Material', 'Id'])
y = y_raw['Egap']

# Normalization
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
X_normalized_df = pd.DataFrame(X_normalized, columns=X.columns)

# Spliting
X_train, X_val, y_train, y_val = train_test_split(X_normalized_df, y, test_size=0.2, random_state=5)
```

First, I trained a baseline model.

In [ ]:
```python
# Baseline Linear Regression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on the training set (optional, for in-sample evaluation)
lr_train_pred = lr_model.predict(X_train)

# Predict on the validation set
lr_val_pred = lr_model.predict(X_val)

# Evaluate the model
lr_train_mse = mean_squared_error(y_train, lr_train_pred)
lr_val_mse = mean_squared_error(y_val, lr_val_pred)

# Print evaluation metrics
print("Training MSE:", lr_train_mse)
print("Validation MSE:", lr_val_mse)
```

Then, after trying different models, I selected XGBoost, which gives the best performance in general. I also found the importance of each feature which will be used later.

In [ ]:
```python
# XGBoost
xgb_model = xgb.XGBRegressor(
    n_estimators=7417,        # Number of boosting rounds
    max_depth=8,             # Maximum depth of trees
    learning_rate=0.02,       # Step size shrinkage
    subsample=0.90,           # Fraction of samples to use per boosting round
    colsample_bytree=0.36,    # Fraction of features to use per tree
    reg_alpha=0.76,
    reg_lambda=0.58,
    min_child_weight=15,
    random_state=5
)

# Fit the model
X_train_inner, X_val_inner, y_train_inner, y_val_inner = train_test_split(
    X_train, y_train, test_size=0.2, random_state=5
)
xgb_model.fit(X_train, y_train, eval_set=[(X_val_inner, y_val_inner)], verbose=False)


# Predict on training and validation sets
xgb_train_pred = xgb_model.predict(X_train)
xgb_val_pred = xgb_model.predict(X_val)

# Evaluate the model
xgb_train_mse = mean_squared_error(y_train, xgb_train_pred)
xgb_val_mse = mean_squared_error(y_val, xgb_val_pred)

# Print evaluation metrics
print("Training MSE:", xgb_train_mse)
print("Validation MSE:", xgb_val_mse)

# Find inportant feature
important_features = xgb_model.feature_importances_
```

I used grid search to find the best parameters.

In [ ]:
```python
# Define the parameter grid
param_grid = {
    'n_estimators': [5000],       # Number of boosting rounds
    'max_depth': [6, 7, 8, 9],              # Maximum depth of trees
    'learning_rate': [0.02],   # Step size shrinkage
    'subsample': [0.8, 0.9, 0.95],             # Fraction of samples per boosting round
    'colsample_bytree': [0.4, 0.5, 0.6, 0.7],       # Fraction of features per tree
    #'reg_alpha': [0.1, 0.5, 1, 5, 10],
    #'reg_lambda': [0.1, 0.5, 1, 5, 10],
    'min_child_weight': [5, 10, 15, 20, 25]
}

# Initialize the GridSearchCV
grid_search = GridSearchCV(
    estimator=xgb.XGBRegressor(random_state=42),
    param_grid=param_grid,
    scoring='neg_mean_squared_error',  # Use MSE as the scoring metric
    cv=3,                             # 3-fold cross-validation
    verbose=2,                        # Display progress
    n_jobs=-1                         # Use all available cores
)

# Fit GridSearchCV
grid_search.fit(X_train_reduced, y_train)

# Print the best parameters and the best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score (MSE):", -grid_search.best_score_)
```

Then I reduced the least important features, and trained another XGBoost model.

In [ ]:
```python
# XGBoost Reduced

xgb_model = xgb.XGBRegressor(
    n_estimators=5000,        # Number of boosting rounds
    max_depth=6,             # Maximum depth of trees
    learning_rate=0.02,       # Step size shrinkage
    subsample=0.8,           # Fraction of samples to use per boosting round
    colsample_bytree=0.4,    # Fraction of features to use per tree
    reg_alpha=0.76,
    reg_lambda=0.58,
    min_child_weight=10,
    random_state=5
)

threshold = 0.0001
X_train_reduced = X_train.iloc[:, important_features > threshold]
X_val_reduced = X_val.iloc[:, important_features > threshold]

# Fit the model with reduced dataset
xgb_model.fit(X_train_reduced, y_train)

# Predict on training and validation sets
xgb_train_pred = xgb_model.predict(X_train_reduced)
xgb_val_pred = xgb_model.predict(X_val_reduced)

# Evaluate the model
xgb_train_mse = mean_squared_error(y_train, xgb_train_pred)
xgb_val_mse = mean_squared_error(y_val, xgb_val_pred)

# Print evaluation metrics
print("Training MSE:", xgb_train_mse)
print("Validation MSE:", xgb_val_mse)
```

I used Optuna for more hyperparameter tuning.

In [ ]:
```python
def objective(trial):
    # Define the hyperparameter search space
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 900, 20000),
        "max_depth": 8,
        "learning_rate": 0.02,
        "subsample": trial.suggest_float("subsample", 0.9, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.4, 0.5),
        "reg_alpha": trial.suggest_float("reg_alpha", 1e-8, 1.0),
        "reg_lambda": trial.suggest_float("reg_lambda", 1e-8, 1.0),
        "min_child_weight": trial.suggest_float("min_child_weight", 10, 15),
        "random_state": 5
    }

    # Train-test split within the training data for validation
    X_train_inner, X_val_inner, y_train_inner, y_val_inner = train_test_split(
        X_train, y_train, test_size=0.2, random_state=42
    )

    # Initialize the XGBoost Regressor
    model = xgb.XGBRegressor(**params, early_stopping_rounds=10, eval_metric="rmse")

    # Train the model
    model.fit(
        X_train_inner, y_train_inner,
        eval_set=[(X_val_inner, y_val_inner)],

        verbose=False
    )

    # Predict and calculate validation MSE
    preds = model.predict(X_val_inner)
    mse = mean_squared_error(y_val_inner, preds)

    return mse  # Optuna will minimize this metric
```

In [ ]:
```python
# Create the study and optimize
study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=50)

# Print the best parameters
print("Best parameters:", study.best_params)
print("Best MSE:", study.best_value)
```

Then I trained the final model with all training data.

In [ ]:
```python
# XGBoost Reduced FINAL

# Load data
test_raw = pd.read_csv('test_X.csv')
test = test_raw.drop(columns=['Material', 'Id'])
scaler = StandardScaler()
test_normalized = scaler.fit_transform(test)
test_normalized_df = pd.DataFrame(test_normalized, columns=test.columns)

# Train model
xgb_model = xgb.XGBRegressor(
    n_estimators=9000,        # Number of boosting rounds
    max_depth=6,             # Maximum depth of trees
    learning_rate=0.02,       # Step size shrinkage
    subsample=0.8,           # Fraction of samples to use per boosting round
    colsample_bytree=0.4,    # Fraction of features to use per tree
    reg_alpha=0.76,
    reg_lambda=0.58,
    min_child_weight=10,
    random_state=5
)

threshold = 0
X_reduced = X_normalized_df.iloc[:, important_features > threshold]
test_reduced = test_normalized_df.iloc[:, important_features > threshold]

# Fit the model with reduced dataset
xgb_model.fit(X_reduced, y)

# Predict on training and validation sets
test_pred = xgb_model.predict(test_reduced)
```

Finally, I wrote the csv file to submit.

In [ ]:
```python
# Make File to Submit

out = pd.read_csv('y_sample_submission.csv')
out['Egap'] = test_pred
out.to_csv('predictions.csv', index=False)
```