project

April 27, 2025

1 BA 476 Team 10 Jupyter Notebook

[1]:
```
%%html
<style>
.cell-output-ipywidget-background {
    background-color: transparent !important;
}
:root {
    --jp-widgets-color: var(--vscode-editor-foreground);
    --jp-widgets-font-size: var(--vscode-editor-font-size);
}
</style>
```

```
<IPython.core.display.HTML object>
```

1.1 Setup

[2]:
```
from pathlib import Path

import kagglehub
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor,␣
↪StackingRegressor
from sklearn.linear_model import Lasso, LinearRegression, Ridge
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,␣
↪root_mean_squared_error
from sklearn.model_selection import GridSearchCV, KFold, train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm.auto import tqdm
from xgboost import XGBRegressor, XGBRFRegressor
```

[3]:
```
# To collect results from all models as we go
full_results = pd.DataFrame()
```

```python
pred_vs_actual = {}

# Color palette
sns.set(
    rc={
        "axes.facecolor": (1, 1, 1, 0),
        "figure.facecolor": (1, 1, 1, 0),
        "text.color": "lightgray",
        "xtick.color": "lightgray",
        "ytick.color": "lightgray",
    }
)
spotify_palette = sns.diverging_palette(279, 141, s=92, l=68, center="light",␣
 ↪as_cmap=True)
spotify_colors = sns.diverging_palette(279, 141, s=92, l=68, center="light")
```

1.2 Data Download and Processing

Download the supplementary data from Kaggle for artist info

Source: https://www.kaggle.com/datasets/adnananam/spotify-artist-stats

[4]:
```python
if Path("data/spotify_artist_data.csv").exists():
    artist_stats = pd.read_csv("data/spotify_artist_data.csv")
else:
    path = kagglehub.dataset_download("adnananam/spotify-artist-stats")
    artist_stats = pd.read_csv(path + "/spotify_artist_data.csv", index_col=0)

    # Remove error rows b/c the creator didn't process correctly
    artist_stats = artist_stats[artist_stats["Lead Streams"] != "Lead Streams"]

    # Cast numeric columns to int
    for col in ["Lead Streams", "Feats", "Tracks", "One Billion", "100␣
 ↪Million"]:
        artist_stats[col] = artist_stats[col].str.replace(",", "").astype(int)

    # Remove the last updated column, it's not useful/relevant
    artist_stats = artist_stats.drop(columns=["Last Updated"])

    artist_stats.to_csv("data/spotify_artist_data.csv", index=False)

artist_stats.head()
```

[4]:

	Artist Name	Lead Streams	Feats	Tracks	One Billion	100 Million
0	Drake	50162292808	19246513666	262	6	130
1	Bad Bunny	44369032140	5391990975	163	5	118
2	Ed Sheeran	38153682361	2791278201	240	10	62
3	The Weeknd	34767779741	4288903657	186	8	72

```
4  Taylor Swift   32596728109    424053296     323              1            96
```

Download the dataset from HuggingFace using Pandas, and drop the extra index column. The `na`/`NaN` values were dropped from the `artists` column because that column is used to merge the supplementary data above with the main dataset.

Source: https://huggingface.co/datasets/maharshipandya/spotify-tracks-dataset

[5]:
```python
# Pulled dataset from HF, dropped unneeded index column
if Path("data/spotify_tracks.csv").exists():
    df = pd.read_csv("data/spotify_tracks.csv")
else:
    df = (
        pd.read_csv("hf://datasets/maharshipandya/spotify-tracks-dataset/
 ↪dataset.csv")
        .drop("Unnamed: 0", axis=1)
        .dropna(subset=["artists"])
    )

    df["duration_s"] = df["duration_ms"] / 1000
    df = df.drop(columns=["duration_ms"])  # Drop original duration column,␣
 ↪keep seconds

    df.to_csv("data/spotify_tracks.csv", index=False)

df_nodupe = df.drop_duplicates(subset=["track_id"]).copy()

df.head()
```

[5]:
```
                 track_id                 artists  \
0  5SuOikwiRyPMVoIQDJUgSV              Gen Hoshino
1  4qPNDBW1i3p13qLCtOKi3A             Ben Woodward
2  1iJBSr7s7jYXzM8EGcbK5b   Ingrid Michaelson;ZAYN
3  6lfxq3CG4xtTiEg7opyCyx             Kina Grannis
4  5vjLSffimiIP26QG5WcN2K         Chord Overstreet

                                         album_name  \
0                                            Comedy
1                                  Ghost (Acoustic)
2                                    To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                           Hold On

                   track_name  popularity  explicit  danceability  energy  \
0                      Comedy          73     False         0.676  0.4610
1            Ghost - Acoustic          55     False         0.420  0.1660
2              To Begin Again          57     False         0.438  0.3590
3  Can't Help Falling In Love          71     False         0.266  0.0596
```

```
4                         Hold On          82     False         0.618  0.4430

   key  loudness  mode  speechiness  acousticness  instrumentalness  liveness  \
0    1    -6.746     0       0.1430        0.0322          0.000001    0.3580
1    1   -17.235     1       0.0763        0.9240          0.000006    0.1010
2    0    -9.734     1       0.0557        0.2100          0.000000    0.1170
3    0   -18.515     1       0.0363        0.9050          0.000071    0.1320
4    2    -9.681     1       0.0526        0.4690          0.000000    0.0829

   valence    tempo  time_signature track_genre  duration_s
0    0.715   87.917               4    acoustic     230.666
1    0.267   77.489               4    acoustic     149.610
2    0.120   76.332               4    acoustic     210.826
3    0.143  181.740               3    acoustic     201.933
4    0.167  119.949               4    acoustic     198.853
```

Adding in more information to the main dataset using each artist's stats. If there are two or more artists present, the stats are averaged.

Stats merged:

- Lead streams
- Streams of features
- Number of tracks
- Number of songs with more than one billion streams
- Number of songs with more than 100 million streams

The second half of the cell creates dummy variables for the `genres` column. The genre column and duplicate song entries are then dropped from the dataframe. Each song is repeated x number of times where x is the number of genres it has.

[6]:
```
if not Path("data/spotify_tracks_processed.csv").exists():
    # Adding in information based on the artist stats (merge on names)
    art_stats_name = set(artist_stats["Artist Name"].values)
    lead_streams, feats, tracks, one_billion, hundred_million = [], [], [], [],␣
 ↪[]

    for row in tqdm(df_nodupe.iterrows(), total=df_nodupe.shape[0],␣
 ↪desc="Processing rows"):
        artists = [x.strip() for x in row[1]["artists"].split(";")]
        temp_lead_streams, temp_feats, temp_tracks, temp_one_billion,␣
 ↪temp_hundred_million = (
            [],
            [],
            [],
            [],
            [],
        )
```

```
        for artist in artists:
            if artist in art_stats_name:
                temp_lead_streams.append(
                    artist_stats[artist_stats["Artist Name"] == artist]["Lead Streams"].values[0]
                )
                temp_feats.append(artist_stats[artist_stats["Artist Name"] == artist]["Feats"].values[0])
                temp_tracks.append(
                    artist_stats[artist_stats["Artist Name"] == artist]["Tracks"].values[0]
                )
                temp_one_billion.append(
                    artist_stats[artist_stats["Artist Name"] == artist]["One Billion"].values[0]
                )
                temp_hundred_million.append(
                    artist_stats[artist_stats["Artist Name"] == artist]["100 Million"].values[0]
                )

        for col, temp in zip(
            [lead_streams, feats, tracks, one_billion, hundred_million],
            [temp_lead_streams, temp_feats, temp_tracks, temp_one_billion, temp_hundred_million],
            strict=True,
        ):
            if len(temp) > 0:
                col.append(np.mean(temp))
            else:
                col.append(0)

    df_nodupe["lead_streams"] = lead_streams
    df_nodupe["feats"] = feats
    df_nodupe["tracks"] = tracks
    df_nodupe["one_billion"] = one_billion
    df_nodupe["hundred_million"] = hundred_million

    # Creating dummy variables based on genres
    g_dummy = pd.get_dummies(df["track_genre"]).groupby(df["track_id"]).sum().astype(int).reset_index()

    dummy_val = g_dummy.copy()
    dummy_val["total"] = dummy_val.sum(axis=1, numeric_only=True)
    dummy_val = dummy_val[["track_id", "total"]].sort_values("track_id", ascending=True)
```

```
    process_check = (
        df.groupby("track_id")
        .size()
        .to_frame("total")
        .reset_index()
        .sort_values("track_id", ascending=True)
    )

    for df1, df2 in zip(process_check.iterrows(), dummy_val.iterrows(),␣
 ↪strict=True):
        assert (df1[1]["total"] == df2[1]["total"]) and (df1[1]["track_id"] ==␣
 ↪df2[1]["track_id"])

    df = df_nodupe.merge(g_dummy, on="track_id").drop(
        ["track_id", "artists", "album_name", "track_name", "track_genre"],␣
 ↪axis=1
    )
    df["explicit"] = df["explicit"].astype(int)
    df.to_csv("data/spotify_tracks_processed.csv", index=False)

else:
    df = pd.read_csv("data/spotify_tracks_processed.csv")
```

1.2.1 Baseline Linear Regression Model

A quick test of the linear regression model using only the base data and dummy variables made from genres.

[7]:
```
# Assuming 'df' is your DataFrame and 'features' is a list of feature column␣
 ↪names
X = df[
    df.columns.difference(
        ["popularity", "lead_streams", "feats", "tracks", "one_billion",␣
 ↪"hundred_million"]
    )
]
y = df["popularity"]

# Split the dataset into train and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=42)

# Initialize the LinearRegression model
model = LinearRegression(
    n_jobs=-1,
)
```

```python
# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate R² and MSE
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

pred_vs_actual["Baseline Linear Regression"] = pd.DataFrame({
    "Actual": y,
    "Predicted": model.predict(X),
})

full_results = pd.concat(
    [
        full_results,
        pd.DataFrame(
            {
                "Model": "Baseline Linear Regression",
                "Dataset": ["Test", "Train", "Full"],
                "R²": [r2, r2_score(y_train, model.predict(X_train)),␣
 ↪r2_score(y, model.predict(X))],
                "MSE": [
                    mse,
                    mean_squared_error(y_train, model.predict(X_train)),
                    mean_squared_error(y, model.predict(X)),
                ],
                "RMSE": [
                    rmse,
                    root_mean_squared_error(y_train, model.predict(X_train)),
                    root_mean_squared_error(y, model.predict(X)),
                ],
                "MAE": [
                    mae,
                    mean_absolute_error(y_train, model.predict(X_train)),
                    mean_absolute_error(y, model.predict(X)),
                ],
            }
        ),
    ]
)
```

```
# Output the results
print(f"R²: {r2}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
```

```
R²: 0.3491801608462217
MSE: 273.8976243937096
RMSE: 16.549852700060796
MAE: 11.890187841790453
```

1.3 Additional Data Processing

1.3.1 Backfill missing `lead_streams` values

Fill in values for `lead_streams` using all columns except for `lead_streams` and `popularity`.

[8]:
```
# Create mask for rows where lead_streams is 0
mask = df['lead_streams'] == 0

# Split data into features (X) and target (y)
X_train = df[~mask].drop(['lead_streams', 'popularity'], axis=1)
y_train = df[~mask]['lead_streams']

# Prepare features for prediction
X_pred = df[mask].drop(['lead_streams', 'popularity'], axis=1)

# Initialize and train the RandomForestRegressor
rf_model = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1,
    max_features='sqrt',
    verbose=1
)
rf_model.fit(X_train, y_train)

# Make predictions for empty values
predictions = rf_model.predict(X_pred)

# Fill in the empty values
df.loc[mask, 'lead_streams'] = predictions

# Verify no more zeros in lead_streams
print(f"Number of zeros in lead_streams: {(df['lead_streams'] == 0).sum()}")
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 12 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    0.1s
```

```
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    0.7s finished
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent
workers.
[Parallel(n_jobs=12)]: Done  26 tasks      | elapsed:    0.0s
```

```
Number of zeros in lead_streams: 0
```

```
[Parallel(n_jobs=12)]: Done 176 tasks      | elapsed:    0.2s
[Parallel(n_jobs=12)]: Done 200 out of 200 | elapsed:    0.2s finished
```

1.3.2 Add clusters as additional features

KMeans clusters as an additonal feature for the models to use.

[9]:
```
std_df = StandardScaler().fit_transform(df[df.columns.
  ↪difference(["popularity"])])
kmeans = KMeans(n_clusters=40, random_state=42)
kmeans.fit(std_df)
df["cluster"] = kmeans.labels_
df["cluster"] = df["cluster"].astype("category")

# Create dummy variables for clusters
cluster_dummies = pd.get_dummies(df['cluster'], prefix='cluster')
df = pd.concat([df, cluster_dummies], axis=1)
df = df.drop(['cluster'], axis=1)

print(f"Added {cluster_dummies.shape[1]} cluster dummy variables")
print(f"Total features: {df.shape[1]}")
```

```
Added 40 cluster dummy variables
Total features: 174
```

1.3.3 Manual correlation check

[10]:
```
dfc = df.corr()

# Create mask for correlations > abs(0.50)
mask = np.abs(dfc) > 0.50

# Get upper triangle of mask to avoid duplicates
mask_upper = np.triu(mask, k=1)

# Find correlation pairs exceeding threshold
high_corr = []
for i in range(len(dfc.columns)):
    for j in range(i + 1, len(dfc.columns)):
        if mask_upper[i, j]:
```

```
            high_corr.append({"var1": dfc.columns[i], "var2": dfc.columns[j],␣
 ↪"corr": dfc.iloc[i, j]})

# Convert to dataframe and sort by absolute correlation
high_corr_df = pd.DataFrame(high_corr)
high_corr_df = high_corr_df.sort_values("corr", key=abs, ascending=False)

print("Correlations > |0.50|:")
print(high_corr_df.to_string(index=False))
```

```
Correlations > |0.50|:
            var1          var2      corr
singer-songwriter    songwriter  1.000000
           samba    cluster_26  1.000000
singer-songwriter    cluster_18  1.000000
      songwriter    cluster_18  1.000000
       breakbeat    cluster_16  0.999496
        cantopop    cluster_14  0.999496
  detroit-techno    cluster_25  0.998994
             idm    cluster_20  0.998994
           study    cluster_37  0.998994
          disney     cluster_4  0.998991
        children     cluster_1  0.998490
     rock-n-roll    cluster_24  0.998482
         new-age    cluster_36  0.997016
          comedy    cluster_23  0.996994
          j-idol     cluster_2  0.996968
       metalcore     cluster_8  0.996962
         spanish    cluster_11  0.996962
     black-metal    cluster_17  0.996480
          gospel    cluster_28  0.995955
           chill     cluster_7  0.993420
      honky-tonk    cluster_21  0.990886
           anime    cluster_30  0.990877
          french    cluster_27  0.983726
       power-pop    cluster_32  0.980259
            funk     cluster_0  0.972330
         country    cluster_35  0.970250
          trance     cluster_3  0.967710
    lead_streams hundred_million  0.952045
           disco     cluster_5  0.951867
       reggaeton    cluster_19  0.947969
         dubstep    cluster_13  0.889024
             dub    cluster_13  0.878204
        alt-rock     cluster_9  0.842081
    lead_streams   one_billion  0.822557
          reggae    cluster_19  0.822527
     alternative     cluster_9  0.813701
```

minimal-techno	cluster_34	0.802951
reggae	reggaeton	0.801791
latino	cluster_19	0.781764
energy	loudness	0.758774
turkish	cluster_31	0.754182
groove	cluster_22	0.737029
latino	reggaeton	0.736928
energy	acousticness	-0.732569
dub	dubstep	0.723472
one_billion	hundred_million	0.706854
club	cluster_6	0.700849
grindcore	cluster_6	0.698637
one_billion	cluster_15	0.681707
techno	cluster_34	0.680813
deep-house	cluster_22	0.680712
lead_streams	cluster_15	0.672134
indie-pop	cluster_29	0.632810
speechiness	cluster_23	0.625313
punk	punk-rock	0.624188
speechiness	comedy	0.623655
edm	house	0.619816
latino	reggae	0.614418
hundred_million	cluster_15	0.613633
lead_streams	featured_streams	0.612210
featured_streams	hundred_million	0.593427
latin	latino	0.590402
alt-rock	alternative	0.588235
featured_tracks	classical	0.583836
loudness	acousticness	-0.582664
indie	indie-pop	0.573530
folk	cluster_29	0.569474
j-dance	cluster_10	0.566230
indie	cluster_29	0.558528
latin	cluster_19	0.553633
dance	cluster_15	0.545551
sad	cluster_31	0.538157
sleep	cluster_12	0.531570
featured_streams	one_billion	0.528161
dancehall	cluster_10	0.525143
latin	reggae	0.509465
latin	reggaeton	0.509465
loudness	cluster_12	-0.507256

[11]:
```
high_corr_df[
    high_corr_df["var1"].str.contains("cluster") | high_corr_df["var2"].str.
↪contains("cluster")
```

```
].groupby("var2")["var1"].apply(lambda x: ", ".join(x)).
↪reset_index(name="var1").sort_values("var2")
```

[11]:

	var2	var1
0	cluster_0	funk
1	cluster_1	children
2	cluster_10	j-dance, dancehall
3	cluster_11	spanish
4	cluster_12	sleep, loudness
5	cluster_13	dubstep, dub
6	cluster_14	cantopop
7	cluster_15	one_billion, lead_streams, hundred_million, dance
8	cluster_16	breakbeat
9	cluster_17	black-metal
10	cluster_18	singer-songwriter, songwriter
11	cluster_19	reggaeton, reggae, latino, latin
12	cluster_2	j-idol
13	cluster_20	idm
14	cluster_21	honky-tonk
15	cluster_22	groove, deep-house
16	cluster_23	comedy, speechiness
17	cluster_24	rock-n-roll
18	cluster_25	detroit-techno
19	cluster_26	samba
20	cluster_27	french
21	cluster_28	gospel
22	cluster_29	indie-pop, folk, indie
23	cluster_3	trance
24	cluster_30	anime
25	cluster_31	turkish, sad
26	cluster_32	power-pop
27	cluster_34	minimal-techno, techno
28	cluster_35	country
29	cluster_36	new-age
30	cluster_37	study
31	cluster_4	disney
32	cluster_5	disco
33	cluster_6	club, grindcore
34	cluster_7	chill
35	cluster_8	metalcore
36	cluster_9	alt-rock, alternative

1.3.4 Drop highly correlated columns

- singer-songwriter
  - Removed since it is an identical match to songwriter

[12]:
```
df = df.drop(columns=["singer-songwriter"])
```

1.4 Models and Evaluation

1.4.1 Baseline Linear Regression

This is another run of the Linear Regression Model but with using the data with extra features.

[13]:
```
# Assuming 'df' is your DataFrame and 'features' is a list of feature column␣
↪names
X = df[df.columns.difference(["popularity"])]
y = df["popularity"]

# Split the dataset into train and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=42)

# Initialize the RandomForestRegressor model
model = LinearRegression(
    n_jobs=-1,
)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate R² and MSE
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

pred_vs_actual["Post-processing Linear Regression"] = pd.DataFrame({
    "Actual": y,
    "Predicted": model.predict(X),
})

full_results = pd.concat(
    [
        full_results,
        pd.DataFrame(
            {
                "Model": "Post-processing Linear Regression",
                "Dataset": ["Test", "Train", "Full"],
                "R²": [r2, r2_score(y_train, model.predict(X_train)),␣
↪r2_score(y, model.predict(X))],
                "MSE": [
                    mse,
```

```
                    mean_squared_error(y_train, model.predict(X_train)),
                    mean_squared_error(y, model.predict(X)),
                ],
                "RMSE": [
                    rmse,
                    root_mean_squared_error(y_train, model.predict(X_train)),
                    root_mean_squared_error(y, model.predict(X)),
                ],
                "MAE": [
                    mae,
                    mean_absolute_error(y_train, model.predict(X_train)),
                    mean_absolute_error(y, model.predict(X)),
                ],
            }
        ),
    ]
)

# Output the results
print(f"R²: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
R²: 0.3566
MSE: 270.7777
RMSE: 16.4553
MAE: 11.7273
```

1.4.2 Lasso and Ridge Regression

[14]:
```
target = "popularity"
features = df[df.columns.difference(["popularity"])]


def preprocess_data(df, features, target):
    X = features
    y = df[target]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled, y_train, y_test


def nested_cv(model, param_grid, X, y, k_outer=5, k_inner=3):
```

```python
    outer_kf = KFold(n_splits=k_outer, shuffle=True, random_state=42)
    outer_mses = []

    # Outer loop with tqdm progress bar
    for train_index, test_index in tqdm(outer_kf.split(X), total=k_outer,␣
 ↪desc="Outer loop"):
        X_train_outer, X_test_outer = X[train_index], X[test_index]
        y_train_outer, y_test_outer = y.iloc[train_index], y.iloc[test_index]

        # Inner loop for hyperparameter tuning using GridSearchCV
        inner_kf = KFold(n_splits=k_inner, shuffle=True, random_state=42)
        grid_search = GridSearchCV(
            model, param_grid, cv=inner_kf, scoring="neg_mean_squared_error",␣
 ↪n_jobs=1
        )
        grid_search.fit(X_train_outer, y_train_outer)

        # Get the best model
        best_model = grid_search.best_estimator_

        # Predictions on the outer fold's test set
        y_pred_outer = best_model.predict(X_test_outer)
        outer_mses.append(mean_squared_error(y_test_outer, y_pred_outer))

    return np.mean(outer_mses), grid_search.best_params_


param_grid = {"alpha": np.logspace(-3, 3, 7)}
X_train_scaled, X_test_scaled, y_train, y_test = preprocess_data(df, features,␣
 ↪target)

# Perform Nested Cross-Validation for Lasso and Ridge
lasso_nested_mse, lasso_best_params = nested_cv(Lasso(), param_grid,␣
 ↪X_train_scaled, y_train)
ridge_nested_mse, ridge_best_params = nested_cv(Ridge(), param_grid,␣
 ↪X_train_scaled, y_train)

print(f"Lasso Nested CV MSE: {lasso_nested_mse}, Best Params:␣
 ↪{lasso_best_params}")
print(f"Ridge Nested CV MSE: {ridge_nested_mse}, Best Params:␣
 ↪{ridge_best_params}")

lasso_final = Lasso(**lasso_best_params)
ridge_final = Ridge(**ridge_best_params)

# Fit the models on the training data
```

```python
lasso_final.fit(X_train_scaled, y_train)
ridge_final.fit(X_train_scaled, y_train)

# Predictions on the test set
y_pred_lasso_final = lasso_final.predict(X_test_scaled)
y_pred_ridge_final = ridge_final.predict(X_test_scaled)

# Calculate the MSE on the test set
lasso_final_mse = mean_squared_error(y_test, y_pred_lasso_final)
ridge_final_mse = mean_squared_error(y_test, y_pred_ridge_final)

print(f"Lasso Final Test MSE: {lasso_final_mse}")
print(f"Ridge Final Test MSE: {ridge_final_mse}")
```

```
Outer loop:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.306e+05, tolerance: 1.416e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.178e+05, tolerance: 1.419e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 7.142e+04, tolerance: 1.428e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.376e+05, tolerance: 1.422e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.338e+04, tolerance: 1.428e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
```

```
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.141e+04, tolerance: 1.427e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.867e+05, tolerance: 2.139e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 6.673e+05, tolerance: 1.427e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 6.976e+05, tolerance: 1.419e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.034e+05, tolerance: 1.431e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.490e+06, tolerance: 2.139e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.750e+05, tolerance: 1.417e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.209e+05, tolerance: 1.420e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
```

```
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 3.478e+04, tolerance: 1.426e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.134e+05, tolerance: 1.422e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 5.561e+05, tolerance: 1.420e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 8.674e+04, tolerance: 1.421e+03
  model = cd_fast.enet_coordinate_descent(
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.684e+05, tolerance: 2.131e+03
  model = cd_fast.enet_coordinate_descent(
```

```
Outer loop:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
Lasso Nested CV MSE: 268.9925919622844, Best Params: {'alpha':
np.float64(0.001)}
Ridge Nested CV MSE: 268.94111313066367, Best Params: {'alpha':
np.float64(100.0)}
Lasso Final Test MSE: 268.7204425004814
Ridge Final Test MSE: 268.68887023195
```

```
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 3.409e+05, tolerance: 2.668e+03
  model = cd_fast.enet_coordinate_descent(
```

[15]:

```
lasso_final_mse = mean_squared_error(y_test, y_pred_lasso_final)
ridge_final_mse = mean_squared_error(y_test, y_pred_ridge_final)

lasso_final_mae = mean_absolute_error(y_test, y_pred_lasso_final)
```

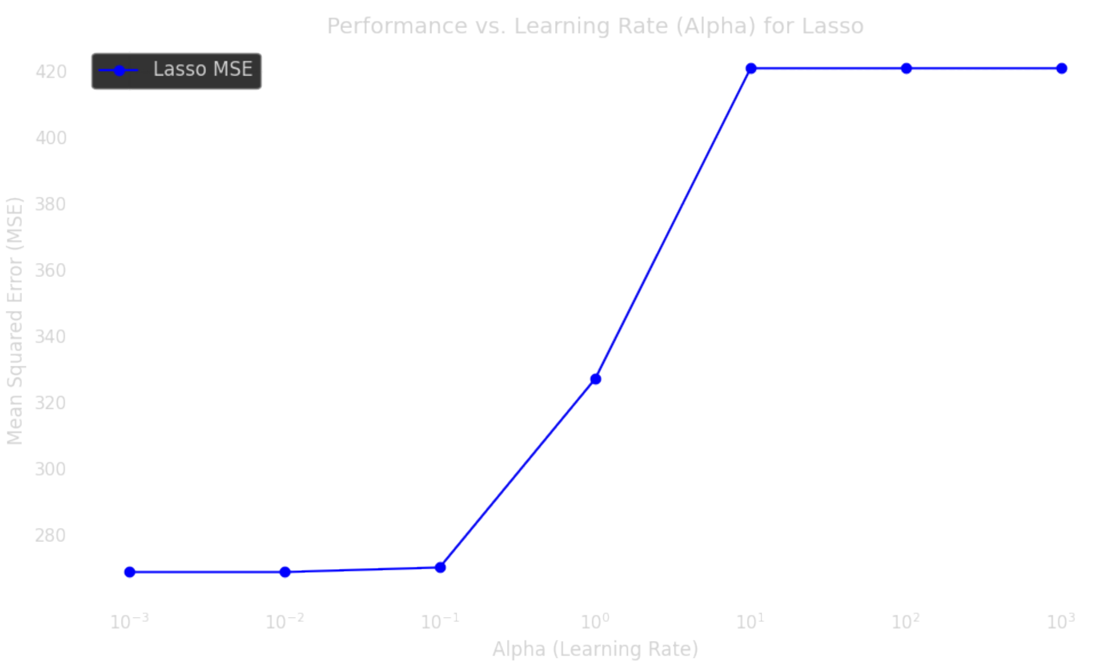
```python
ridge_final_mae = mean_absolute_error(y_test, y_pred_ridge_final)

lasso_final_rmse = root_mean_squared_error(y_test, y_pred_lasso_final)
ridge_final_rmse = root_mean_squared_error(y_test, y_pred_ridge_final)

lasso_final_r2 = r2_score(y_test, y_pred_lasso_final)
ridge_final_r2 = r2_score(y_test, y_pred_ridge_final)

print(f"Lasso Final Test MSE: {lasso_final_mse}")
print(f"Ridge Final Test MSE: {ridge_final_mse}")

print(f"Lasso Final Test MAE: {lasso_final_mae}")
print(f"Ridge Final Test MAE: {ridge_final_mae}")

print(f"Lasso Final Test RMSE: {lasso_final_rmse}")
print(f"Ridge Final Test RMSE: {ridge_final_rmse}")

print(f"Lasso Final Test r2: {lasso_final_rmse}")
print(f"Ridge Final Test r2: {ridge_final_rmse}")
```

```
Lasso Final Test MSE: 268.7204425004814
Ridge Final Test MSE: 268.68887023195
Lasso Final Test MAE: 11.684582786467717
Ridge Final Test MAE: 11.685954097294095
Lasso Final Test RMSE: 16.392694790682874
Ridge Final Test RMSE: 16.391731764275242
Lasso Final Test r2: 16.392694790682874
Ridge Final Test r2: 16.391731764275242
```

[16]:
```python
alphas = np.logspace(-3, 3, 7)  # This generates values like [0.001, 0.01, ...,␣
 ↪1000]

# Initialize lists to store MSE for Lasso at each alpha value
lasso_mses = []
ridge_mses = []
# Loop over the alphas to calculate MSE for both models
for alpha in alphas:
    # Train the models with the current alpha value
    lasso_model = Lasso(alpha=alpha)

    # Fit Lasso using the same scaled training data
    lasso_model.fit(X_train_scaled, y_train)

    # Predict and calculate MSE for the model
    y_pred_lasso = lasso_model.predict(X_test_scaled)

    lasso_mses.append(mean_squared_error(y_test, y_pred_lasso))
```

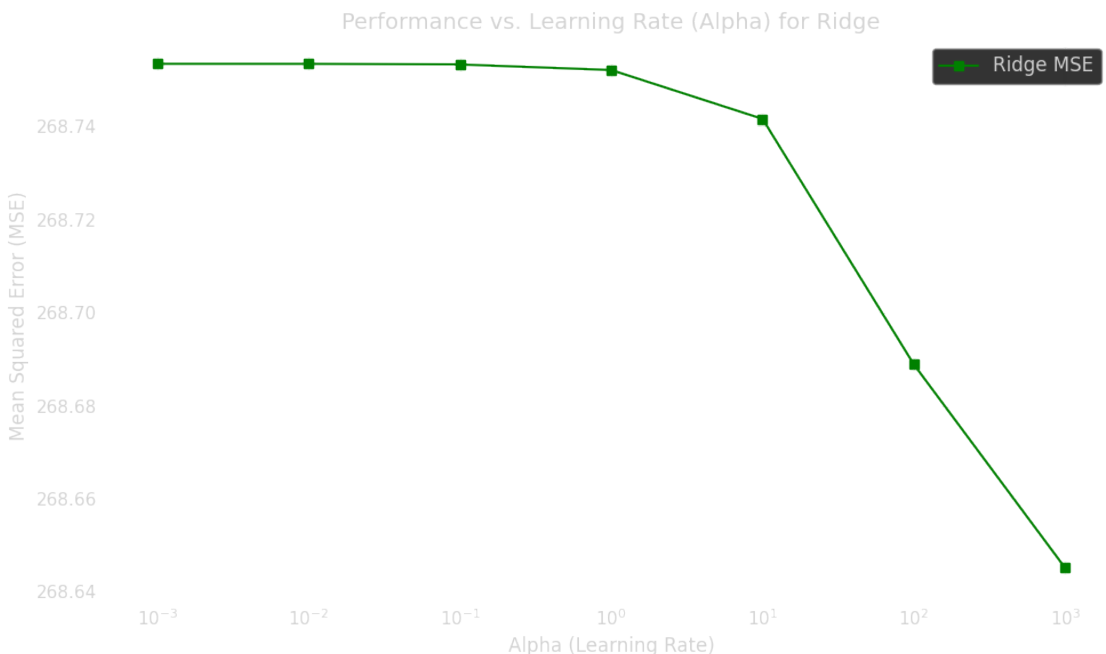
```python
# Plotting the MSE vs. alpha (learning rate)
plt.figure(figsize=(10, 6))

# Plotting for Lasso
plt.plot(alphas, lasso_mses, label="Lasso MSE", marker="o", linestyle="-",␣
 ↪color="blue")

# Set labels and title
plt.xlabel("Alpha (Learning Rate)", color="lightgray", fontsize=12)
plt.ylabel("Mean Squared Error (MSE)", color="lightgray", fontsize=12)
plt.title("Performance vs. Learning Rate (Alpha) for Lasso", fontsize=14)
plt.xscale("log")  # Use log scale for alpha, as alpha spans orders of magnitude
plt.legend(loc="upper left", fontsize=12, frameon=True, facecolor="black",␣
 ↪edgecolor="gray")
plt.grid(True, linestyle="--", alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```

```
/home/cethan/GitHub/BA476-Project/.venv/lib/python3.12/site-
packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 3.409e+05, tolerance: 2.668e+03
  model = cd_fast.enet_coordinate_descent(
```



[17]:

```
alphas = np.logspace(-3, 3, 7)
ridge_mses = []

for alpha in alphas:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train_scaled, y_train)
    y_pred_ridge = ridge_model.predict(X_test_scaled)
    ridge_mses.append(mean_squared_error(y_test, y_pred_ridge))

plt.figure(figsize=(10, 6))
plt.plot(alphas, ridge_mses, label="Ridge MSE", marker="s", linestyle="-",␣
 ↪color="green")
plt.xlabel("Alpha (Learning Rate)", color="lightgray", fontsize=12)
plt.ylabel("Mean Squared Error (MSE)", color="lightgray", fontsize=12)
plt.title("Performance vs. Learning Rate (Alpha) for Ridge", fontsize=14)
plt.xscale("log")
plt.legend(loc="upper right", fontsize=12, frameon=True, facecolor="black",␣
 ↪edgecolor="gray")
plt.tight_layout()
plt.show()
```



1.4.3 Random Forest Regressor

[18]:
```
# Assuming 'df' is your DataFrame and 'features' is a list of feature column␣
↪names
X = df[df.columns.difference(["popularity"])]
y = df["popularity"]

# Split the dataset into train and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Initialize the RandomForestRegressor model
model = RandomForestRegressor(
    n_estimators=200, random_state=42, n_jobs=-1, max_features="sqrt",␣
↪bootstrap=True
)

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate R² and MSE
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

pred_vs_actual["Random Forest Regressor"] = pd.DataFrame({
    "Actual": y,
    "Predicted": model.predict(X),
})

full_results = pd.concat(
    [
        full_results,
        pd.DataFrame(
            {
                "Model": "Random Forest Regressor",
                "Dataset": ["Test", "Train", "Full"],
                "R²": [r2, r2_score(y_train, model.predict(X_train)),␣
↪r2_score(y, model.predict(X))],
                "MSE": [
                    mse,
                    mean_squared_error(y_train, model.predict(X_train)),
                    mean_squared_error(y, model.predict(X)),
```

```
                ],
                "RMSE": [
                    rmse,
                    root_mean_squared_error(y_train, model.predict(X_train)),
                    root_mean_squared_error(y, model.predict(X)),
                ],
                "MAE": [
                    mae,
                    mean_absolute_error(y_train, model.predict(X_train)),
                    mean_absolute_error(y, model.predict(X)),
                ],
            }
        ),
    ]
)

# Output the results
print(f"R²: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
R²: 0.5619
MSE: 184.3921
RMSE: 13.5791
MAE: 9.1269
```

1.4.4 XGBoost Regressor

Given this is a boosting model, `early_stopping_rounds` has been set to 50 to avoid overfitting on the train and validation data. But unlike the other models, this one gets a custom `75:15:10` train/validation/test split.

[19]:
```
# Assuming 'df' is your DataFrame and 'features' is a list of feature column␣
 ↪names
X = df[df.columns.difference(["popularity"])]
y = df["popularity"]

# Split the dataset into train, test, and validation sets (75% train, 15% test,␣
 ↪10% validation)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=40,␣
 ↪random_state=42)

# Initialize the XGBRegressor model
model = XGBRegressor(
```

```python
    tree_method="hist",
    n_estimators=300,
    n_jobs=-1,
    random_state=42,
    enable_categorical=False,
    early_stopping_rounds=50,
)

# Fit the model to the training data
model.fit(X_train, y_train, eval_set=[(X_val, y_val)])

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
[0]     validation_0-rmse:16.86794
[1]     validation_0-rmse:17.11843
[2]     validation_0-rmse:16.94914
[3]     validation_0-rmse:16.28775
[4]     validation_0-rmse:15.77475
[5]     validation_0-rmse:15.50319
[6]     validation_0-rmse:14.68116
[7]     validation_0-rmse:14.72310
[8]     validation_0-rmse:14.74026
[9]     validation_0-rmse:14.74997
[10]    validation_0-rmse:14.62337
[11]    validation_0-rmse:14.21314
[12]    validation_0-rmse:14.04983
[13]    validation_0-rmse:14.03198
[14]    validation_0-rmse:13.94472
[15]    validation_0-rmse:14.03856
[16]    validation_0-rmse:14.07062
[17]    validation_0-rmse:14.09857
[18]    validation_0-rmse:13.99118
[19]    validation_0-rmse:13.87197
[20]    validation_0-rmse:13.75798
[21]    validation_0-rmse:13.56000
[22]    validation_0-rmse:13.52287
[23]    validation_0-rmse:13.45103
[24]    validation_0-rmse:13.47599
[25]    validation_0-rmse:13.45573
[26]    validation_0-rmse:13.28203
[27]    validation_0-rmse:13.30003
[28]    validation_0-rmse:13.19011
[29]    validation_0-rmse:13.13425
[30]    validation_0-rmse:13.12120
[31]    validation_0-rmse:13.10097
[32]    validation_0-rmse:13.07349
[33]    validation_0-rmse:13.08953
```

```
[34]	validation_0-rmse:13.04330
[35]	validation_0-rmse:13.00719
[36]	validation_0-rmse:12.97485
[37]	validation_0-rmse:12.97322
[38]	validation_0-rmse:12.93318
[39]	validation_0-rmse:12.91695
[40]	validation_0-rmse:12.92997
[41]	validation_0-rmse:12.92679
[42]	validation_0-rmse:12.90183
[43]	validation_0-rmse:12.86906
[44]	validation_0-rmse:12.82164
[45]	validation_0-rmse:12.80089
[46]	validation_0-rmse:12.77686
[47]	validation_0-rmse:12.61819
[48]	validation_0-rmse:12.60432
[49]	validation_0-rmse:12.63647
[50]	validation_0-rmse:12.58119
[51]	validation_0-rmse:12.59170
[52]	validation_0-rmse:12.61335
[53]	validation_0-rmse:12.62897
[54]	validation_0-rmse:12.62011
[55]	validation_0-rmse:12.56420
[56]	validation_0-rmse:12.48863
[57]	validation_0-rmse:12.46631
[58]	validation_0-rmse:12.48386
[59]	validation_0-rmse:12.45601
[60]	validation_0-rmse:12.45776
[61]	validation_0-rmse:12.44131
[62]	validation_0-rmse:12.50152
[63]	validation_0-rmse:12.49750
[64]	validation_0-rmse:12.51028
[65]	validation_0-rmse:12.51954
[66]	validation_0-rmse:12.51048
[67]	validation_0-rmse:12.55699
[68]	validation_0-rmse:12.46605
[69]	validation_0-rmse:12.34799
[70]	validation_0-rmse:12.34792
[71]	validation_0-rmse:12.35496
[72]	validation_0-rmse:12.35717
[73]	validation_0-rmse:12.29108
[74]	validation_0-rmse:12.24197
[75]	validation_0-rmse:12.24440
[76]	validation_0-rmse:12.24401
[77]	validation_0-rmse:12.25421
[78]	validation_0-rmse:12.26423
[79]	validation_0-rmse:12.17381
[80]	validation_0-rmse:12.18054
[81]	validation_0-rmse:12.16817
```

```
[82]    validation_0-rmse:12.16121
[83]    validation_0-rmse:12.17259
[84]    validation_0-rmse:12.17096
[85]    validation_0-rmse:12.13431
[86]    validation_0-rmse:12.12734
[87]    validation_0-rmse:12.13144
[88]    validation_0-rmse:12.07335
[89]    validation_0-rmse:12.07187
[90]    validation_0-rmse:12.06345
[91]    validation_0-rmse:12.05346
[92]    validation_0-rmse:12.04924
[93]    validation_0-rmse:12.05250
[94]    validation_0-rmse:12.04332
[95]    validation_0-rmse:12.04349
[96]    validation_0-rmse:11.99958
[97]    validation_0-rmse:11.99755
[98]    validation_0-rmse:11.97719
[99]    validation_0-rmse:11.96176
[100]   validation_0-rmse:11.90561
[101]   validation_0-rmse:11.89568
[102]   validation_0-rmse:11.86229
[103]   validation_0-rmse:11.87343
[104]   validation_0-rmse:11.93056
[105]   validation_0-rmse:11.93115
[106]   validation_0-rmse:11.93338
[107]   validation_0-rmse:11.79412
[108]   validation_0-rmse:11.78143
[109]   validation_0-rmse:11.66949
[110]   validation_0-rmse:11.67563
[111]   validation_0-rmse:11.65606
[112]   validation_0-rmse:11.63606
[113]   validation_0-rmse:11.63871
[114]   validation_0-rmse:11.65056
[115]   validation_0-rmse:11.64819
[116]   validation_0-rmse:11.60264
[117]   validation_0-rmse:11.64512
[118]   validation_0-rmse:11.63476
[119]   validation_0-rmse:11.59142
[120]   validation_0-rmse:11.58779
[121]   validation_0-rmse:11.58683
[122]   validation_0-rmse:11.58692
[123]   validation_0-rmse:11.58501
[124]   validation_0-rmse:11.58650
[125]   validation_0-rmse:11.59385
[126]   validation_0-rmse:11.59440
[127]   validation_0-rmse:11.24660
[128]   validation_0-rmse:11.23050
[129]   validation_0-rmse:11.22969
```

```
[130]	validation_0-rmse:11.23080
[131]	validation_0-rmse:11.24547
[132]	validation_0-rmse:11.24442
[133]	validation_0-rmse:11.24304
[134]	validation_0-rmse:11.23913
[135]	validation_0-rmse:11.24244
[136]	validation_0-rmse:11.17558
[137]	validation_0-rmse:11.17613
[138]	validation_0-rmse:11.12257
[139]	validation_0-rmse:10.77217
[140]	validation_0-rmse:10.80583
[141]	validation_0-rmse:10.80616
[142]	validation_0-rmse:10.80640
[143]	validation_0-rmse:10.75727
[144]	validation_0-rmse:10.74497
[145]	validation_0-rmse:10.74945
[146]	validation_0-rmse:10.76559
[147]	validation_0-rmse:10.69490
[148]	validation_0-rmse:10.69760
[149]	validation_0-rmse:10.69649
[150]	validation_0-rmse:10.69058
[151]	validation_0-rmse:10.69028
[152]	validation_0-rmse:10.68820
[153]	validation_0-rmse:10.68968
[154]	validation_0-rmse:10.69391
[155]	validation_0-rmse:10.71765
[156]	validation_0-rmse:10.73585
[157]	validation_0-rmse:10.73769
[158]	validation_0-rmse:10.73735
[159]	validation_0-rmse:10.74831
[160]	validation_0-rmse:10.74448
[161]	validation_0-rmse:10.73949
[162]	validation_0-rmse:10.76039
[163]	validation_0-rmse:10.90721
[164]	validation_0-rmse:10.90607
[165]	validation_0-rmse:10.91344
[166]	validation_0-rmse:11.02097
[167]	validation_0-rmse:10.96051
[168]	validation_0-rmse:10.97271
[169]	validation_0-rmse:10.99476
[170]	validation_0-rmse:10.99410
[171]	validation_0-rmse:11.00806
[172]	validation_0-rmse:11.00852
[173]	validation_0-rmse:11.00317
[174]	validation_0-rmse:10.99935
[175]	validation_0-rmse:11.00699
[176]	validation_0-rmse:10.99899
[177]	validation_0-rmse:10.99563
```

```
[178]	validation_0-rmse:10.98847
[179]	validation_0-rmse:10.98537
[180]	validation_0-rmse:10.98636
[181]	validation_0-rmse:10.99275
[182]	validation_0-rmse:11.00718
[183]	validation_0-rmse:11.01002
[184]	validation_0-rmse:11.01185
[185]	validation_0-rmse:10.98804
[186]	validation_0-rmse:10.97407
[187]	validation_0-rmse:10.99669
[188]	validation_0-rmse:10.95985
[189]	validation_0-rmse:10.96367
[190]	validation_0-rmse:10.96212
[191]	validation_0-rmse:10.95016
[192]	validation_0-rmse:10.95171
[193]	validation_0-rmse:10.95667
[194]	validation_0-rmse:10.95856
[195]	validation_0-rmse:10.93135
[196]	validation_0-rmse:10.93472
[197]	validation_0-rmse:10.93198
[198]	validation_0-rmse:10.95688
[199]	validation_0-rmse:10.93941
[200]	validation_0-rmse:10.85878
[201]	validation_0-rmse:10.87828
[202]	validation_0-rmse:10.87758
```

[20]:

```
pred_vs_actual["XGBoost Regressor"] = pd.DataFrame({
    "Actual": y,
    "Predicted": model.predict(X),
})

# Calculate R² and MSE
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

full_results = pd.concat(
    [
        full_results,
        pd.DataFrame(
            {
                "Model": "XGBoost Regressor",
                "Dataset": ["Test", "Train", "Full"],
                "R²": [r2, r2_score(y_train, model.predict(X_train)),
 ↪r2_score(y, model.predict(X))],
                "MSE": [
```

```
                    mse,
                    mean_squared_error(y_train, model.predict(X_train)),
                    mean_squared_error(y, model.predict(X)),
                ],
                "RMSE": [
                    rmse,
                    root_mean_squared_error(y_train, model.predict(X_train)),
                    root_mean_squared_error(y, model.predict(X)),
                ],
                "MAE": [
                    mae,
                    mean_absolute_error(y_train, model.predict(X_train)),
                    mean_absolute_error(y, model.predict(X)),
                ],
            }
        ),
    ]
)


# Output the results
print(f"R²: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
R²: 0.5191
MSE: 201.7432
RMSE: 14.2036
MAE: 9.6627
```

1.4.5 XGBoost Random Forest Regressor

[21]:
```
# Assuming 'df' is your DataFrame and 'features' is a list of feature column␣
 ↪names
X = df[df.columns.difference(["popularity"])]
y = df["popularity"]

# Split the dataset into train and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=42)

# Initialize the XGBRFRegressor model
model = XGBRFRegressor(
    tree_method="hist", n_estimators=200, n_jobs=-1, random_state=42,␣
 ↪enable_categorical=False
)
```

```
# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate R² and MSE
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

pred_vs_actual["XGBoost Random Forest Regressor"] = pd.DataFrame({
    "Actual": y,
    "Predicted": model.predict(X),
})

full_results = pd.concat(
    [
        full_results,
        pd.DataFrame(
            {
                "Model": "XGBoost Random Forest Regressor",
                "Dataset": ["Test", "Train", "Full"],
                "R²": [r2, r2_score(y_train, model.predict(X_train)),
 ↪r2_score(y, model.predict(X))],
                "MSE": [
                    mse,
                    mean_squared_error(y_train, model.predict(X_train)),
                    mean_squared_error(y, model.predict(X)),
                ],
                "RMSE": [
                    rmse,
                    root_mean_squared_error(y_train, model.predict(X_train)),
                    root_mean_squared_error(y, model.predict(X)),
                ],
                "MAE": [
                    mae,
                    mean_absolute_error(y_train, model.predict(X_train)),
                    mean_absolute_error(y, model.predict(X)),
                ],
            }
        ),
    ]
)
```

```
# Output the results
print(f"R²: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
R²: 0.2044
MSE: 334.8450
RMSE: 18.2988
MAE: 14.7479
```

1.5 Plots

Plot of RMSE and MAE for each model

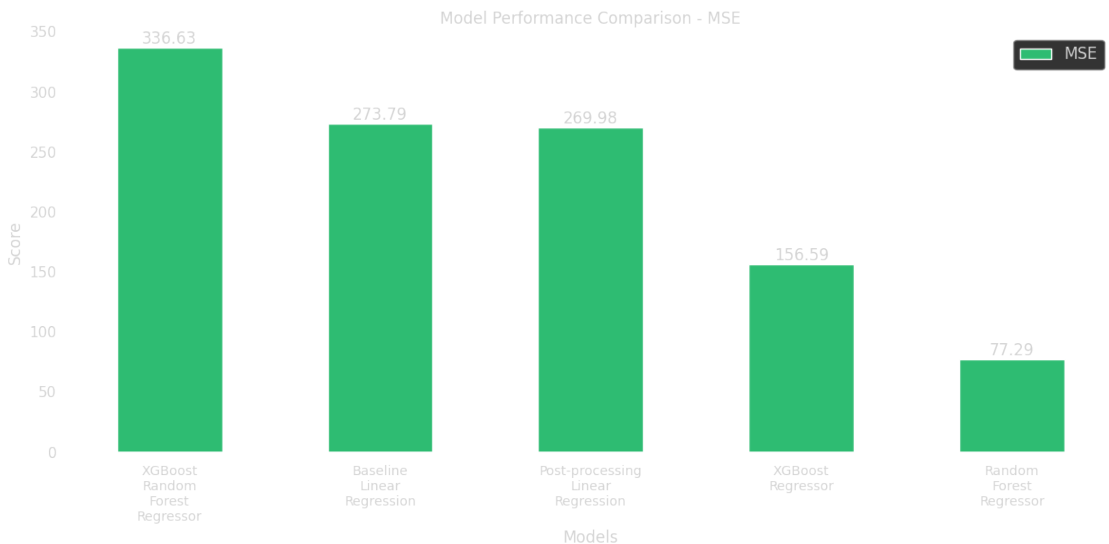
[22]:
```
# Set figure size
plt.figure(figsize=(12, 6))

plt_df = full_results[full_results["Dataset"] == "Full"].sort_values("MSE",␣
↪ascending=False)

# Create bar positions
x = range(len(plt_df))

# Create grouped bars for each metric with adjusted positions
bar_width = 0.2
bars1 = plt.bar(
    [i - bar_width / 2 for i in x],
    plt_df["RMSE"],
    bar_width,
    label="RMSE",
    color=spotify_colors[-1],
    edgecolor="black",
    linewidth=0.5,
)
bars2 = plt.bar(
    [i + bar_width / 2 for i in x],
    plt_df["MAE"],
    bar_width,
    label="MAE",
    color=spotify_colors[0],
    edgecolor="black",
    linewidth=0.5,
)

# Add value labels on top of each bar
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
```

```
        plt.text(
            bar.get_x() + bar.get_width() / 2.0,
            height + 0.3,
            f"{height:.2f}",
            ha="center",
            va="bottom",
            fontsize=9,
        )

# Customize the plot
plt.xlabel("Models", color="lightgray", fontsize=12)
plt.ylabel("Score", color="lightgray", fontsize=12)
plt.title("Model Performance Comparison - RMSE and MAE", fontsize=14)

# Position x-ticks at the center of each model's bar group
# Format labels with newlines between words
labels = plt_df["Model"].str.replace(" ", "\n", regex=False)
plt.xticks(x, labels, rotation=0, ha="center", fontsize=10)

# Improve legend readability
plt.legend(loc="upper right", fontsize=12, frameon=True, facecolor="black",␣
 ↪edgecolor="gray")
plt.grid(True, axis="y", linestyle="--", alpha=0.7)
plt.grid(False, axis="x")  # Disable grid for x-axis
plt.ylim(0, 20)  # Set y-axis limit

# Adjust layout to prevent label cutoff
plt.tight_layout()

plt.savefig("images/rmse-mae.png", dpi=300, bbox_inches="tight",␣
 ↪transparent=True)

plt.show()
```

Plot of MSE for each model

[23]:
```python
# Set figure size
plt.figure(figsize=(12, 6))

plt_df = full_results[full_results["Dataset"] == "Full"].sort_values("MSE", ascending=False)

# Create bar positions
x = range(len(plt_df))

# Create bar plot for MSE
bars = plt.bar(x, plt_df["MSE"], width=0.5, color=spotify_colors[-1], label="MSE")

# Add value labels on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2.0, height, f"{height:.2f}", ha="center", va="bottom")

# Customize the plot
plt.xlabel("Models", color="lightgray", fontsize=12)
plt.ylabel("Score", color="lightgray", fontsize=12)
plt.title("Model Performance Comparison - MSE")

# Position x-ticks at the center of each model's bar group
# Format labels with newlines between words
labels = plt_df["Model"].str.replace(" ", "\n", regex=False)
```

```
plt.xticks(x, labels, rotation=0, ha="center", fontsize=10)

plt.legend(loc="upper right", fontsize=12, frameon=True, facecolor="black",
 ↪edgecolor="gray")
plt.grid(True, axis="y", linestyle="--", alpha=0.7)
plt.grid(False, axis="x")  # Disable grid for x-axis

plt.ylim(0, 350)  # Set y-axis limit to 10% above max value

# Adjust layout to prevent label cutoff
plt.tight_layout()

plt.savefig("images/mse.png", dpi=300, bbox_inches="tight", transparent=True)

plt.show()
```

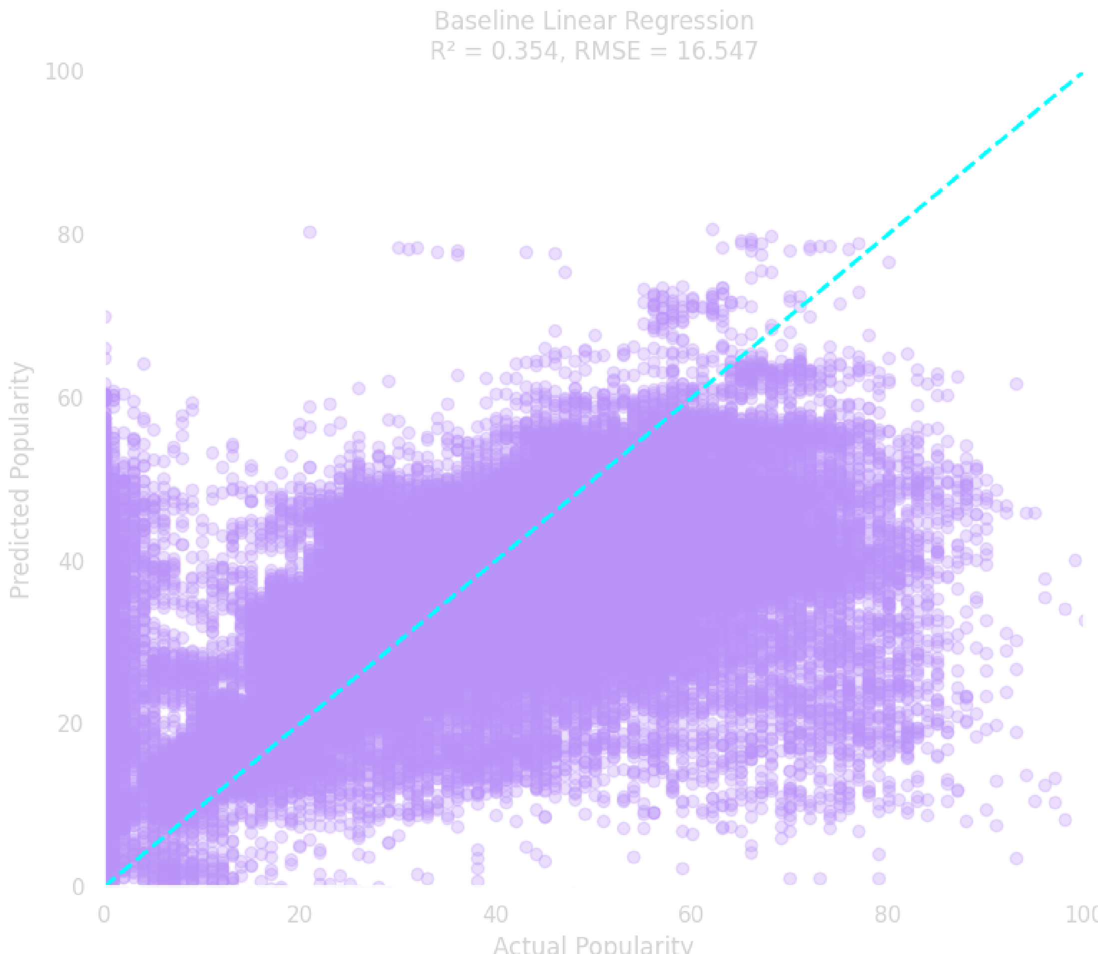


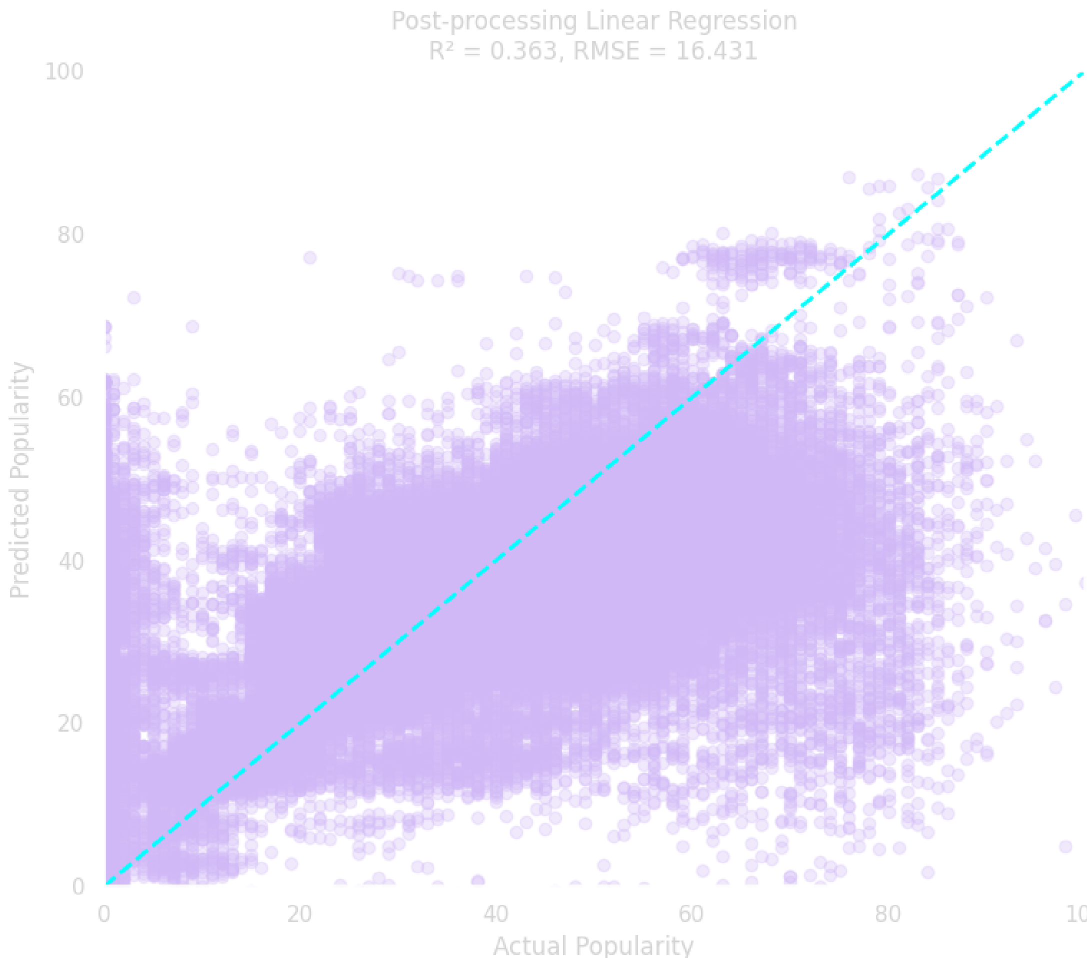
Predictions vs Actual

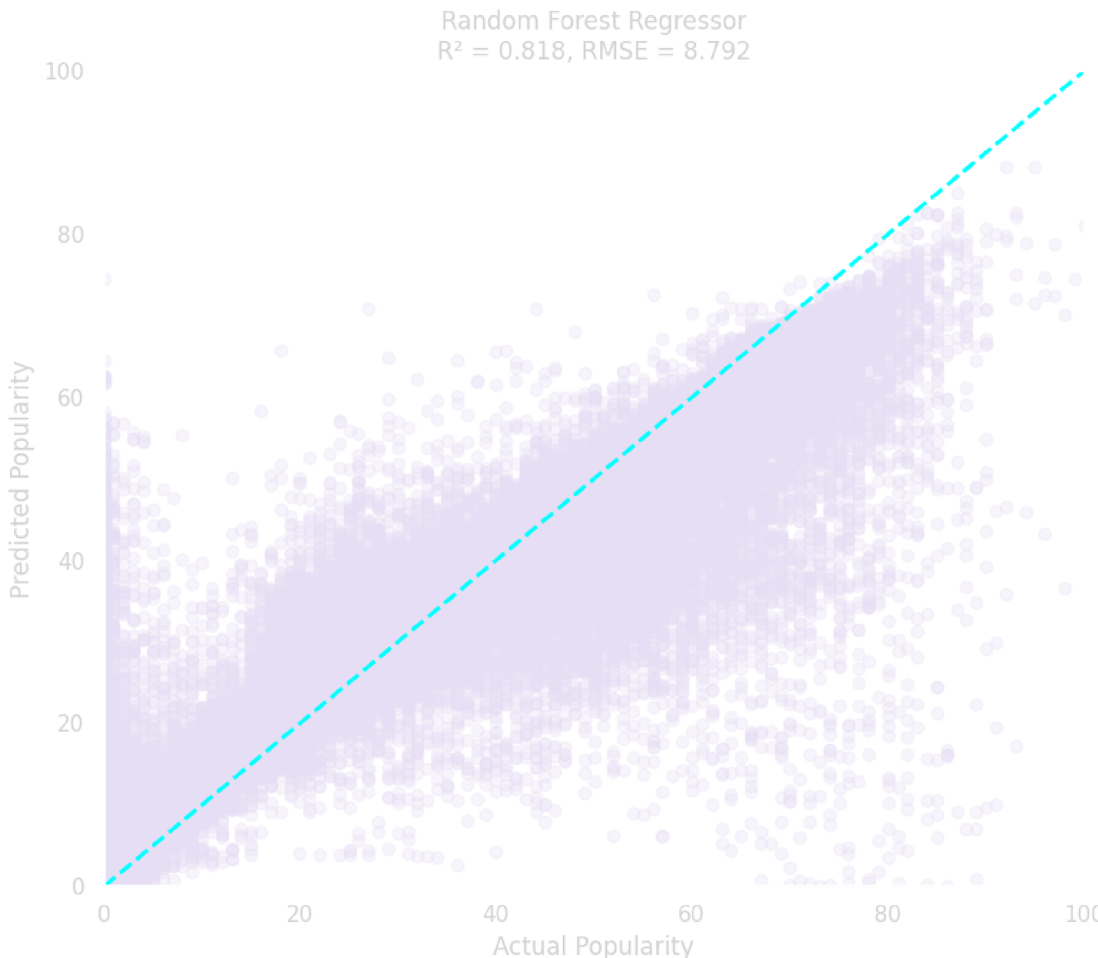
[24]:
```
for i, model in enumerate(pred_vs_actual):
    # Create a new figure for each model
    plt.figure(figsize=(8, 7))

    # Create scatter plot of actual vs predicted values
    plt.scatter(pred_vs_actual[model]["Actual"],
 ↪pred_vs_actual[model]["Predicted"], alpha=0.3, color=spotify_colors[i %
 ↪len(spotify_colors)])

    # Add perfect prediction line
    min_val = 0
```

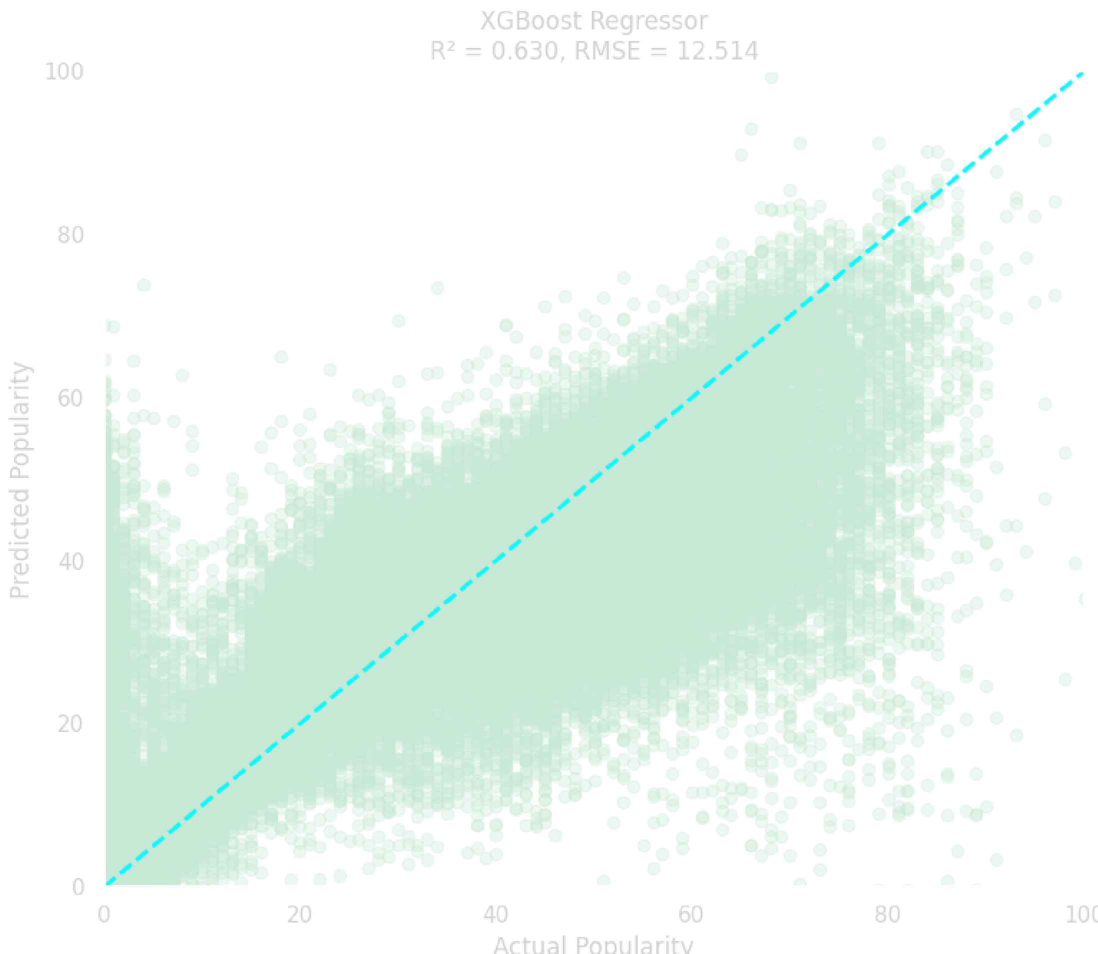
```
    max_val = 100
    plt.plot([min_val, max_val], [min_val, max_val], "--", lw=2, color="cyan")

    # Calculate metrics for the title
    r2 = r2_score(pred_vs_actual[model]["Actual"],␣
↪pred_vs_actual[model]["Predicted"])
    rmse = root_mean_squared_error(pred_vs_actual[model]["Actual"],␣
↪pred_vs_actual[model]["Predicted"])

    # Set title and labels
    plt.title(f"{model}\nR² = {r2:.3f}, RMSE = {rmse:.3f}", fontsize=12,␣
↪color="lightgray")
    plt.xlabel("Actual Popularity", color="lightgray")
    plt.ylabel("Predicted Popularity", color="lightgray")

    # Set tick colors
    plt.tick_params(colors="lightgray")

    # Add grid
    plt.grid(True, linestyle="--", alpha=0.7)

    # Set equal limits for better comparison
    plt.xlim(0, 100)
    plt.ylim(0, 100)

    # Adjust layout
    plt.tight_layout()

    # Save the figure
    plt.savefig(
        f"images/predicted_vs_actual_{model.replace(' ', '_').lower()}.png",
        dpi=300,
        bbox_inches="tight",
        transparent=True,
    )

# Show a message instead of displaying all plots at once
print(f"Created and saved {len(pred_vs_actual)} individual model plots to␣
↪images/ directory")
```

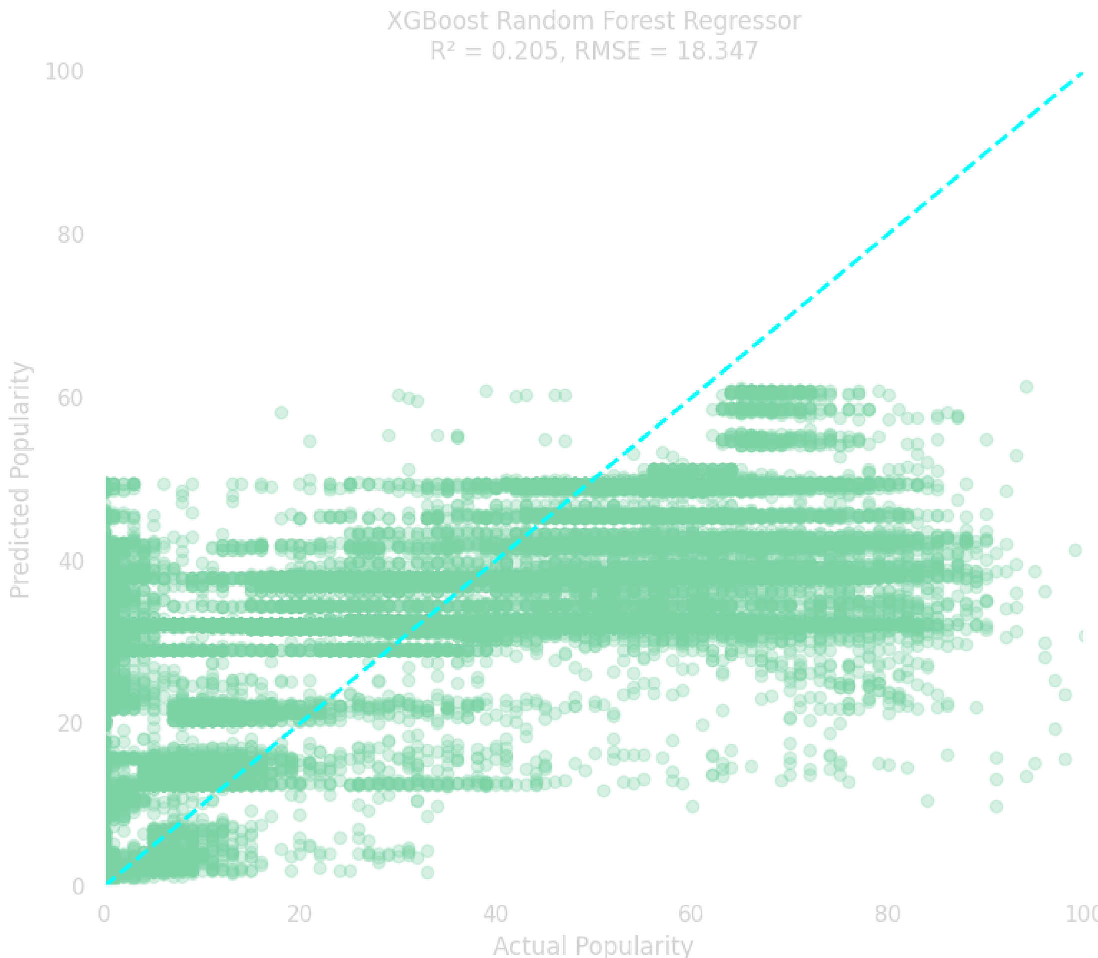
```
Created and saved 5 individual model plots to images/ directory
```


Baseline Linear Regression
R² = 0.354, RMSE = 16.547
Predicted Popularity
Actual Popularity
0
20
40
60
80
100
0
20
40
60
80
100


Post-processing Linear Regression
R² = 0.363, RMSE = 16.431
Predicted Popularity
Actual Popularity
0
20
40
60
80
100


Random Forest Regressor
R² = 0.818, RMSE = 8.792
Predicted Popularity
Actual Popularity
0
20
40
60
80
100
0
20
40
60
80
100


XGBoost Regressor
R² = 0.630, RMSE = 12.514
Predicted Popularity
Actual Popularity
0
20
40
60
80
100
0
20
40
60
80
100



[25]:
```
full_results.to_csv("data/model_results.csv", index=False)
full_results
```

[25]:

	Model	Dataset	R²	MSE	RMSE	\
0	Baseline Linear Regression	Test	0.349180	273.897624	16.549853	
1	Baseline Linear Regression	Train	0.355471	273.743524	16.545196	
2	Baseline Linear Regression	Full	0.353596	273.789754	16.546593	
0	Post-processing Linear Regression	Test	0.356594	270.777686	16.455324	
1	Post-processing Linear Regression	Train	0.365137	269.637920	16.420655	
2	Post-processing Linear Regression	Full	0.362591	269.979850	16.431064	
0	Random Forest Regressor	Test	0.561858	184.392086	13.579105	
1	Random Forest Regressor	Train	0.926086	31.392637	5.602913	
2	Random Forest Regressor	Full	0.817516	77.292471	8.791614	
0	XGBoost Regressor	Test	0.519100	201.743164	14.203632	
1	XGBoost Regressor	Train	0.666832	141.588531	11.899098	
2	XGBoost Regressor	Full	0.630303	156.588181	12.513520	
0	XGBoost Random Forest Regressor	Test	0.204361	334.844971	18.298769	

```
1    XGBoost Random Forest Regressor   Train  0.205610  337.392120  18.368237
2    XGBoost Random Forest Regressor    Full  0.205238  336.627960  18.347424

         MAE
0  11.890188
1  11.893864
2  11.892761
0  11.727340
1  11.704718
2  11.711505
0   9.126881
1   3.553841
2   5.225753
0   9.662746
1   8.235728
2   8.591683
0  14.747857
1  14.788769
2  14.776496
```

1.6 KFold Cross Validation

Because the Random Forest Regressor performed the best, we will use it for KFold Cross Validation.

[26]:
```python
# Assuming 'df' is already loaded and preprocessed
target = "popularity"
features = df.columns.difference(["popularity"])


# Preprocessing function (No scaling needed for Random Forest)
def preprocess_data(df, features, target):
    X = df[features]
    y = df[target]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=42)
    return X_train, X_test, y_train, y_test


# Nested Cross-Validation function with GridSearchCV
def nested_cv(model, param_grid, X, y, k_outer=5, k_inner=3):
    outer_kf = KFold(n_splits=k_outer, shuffle=True, random_state=42)
    outer_mses = []

    # Store all parameter combinations and their corresponding MSE for␣
 ↪visualization
    param_combinations = []
    mse_values = []
```

```
    # Outer loop for cross-validation with tqdm progress bar
    for train_index, test_index in tqdm(outer_kf.split(X), total=k_outer,␣
 ↪desc="Outer loop"):
        X_train_outer, X_test_outer = X.iloc[train_index], X.iloc[test_index]
        y_train_outer, y_test_outer = y.iloc[train_index], y.iloc[test_index]

        # Inner loop for hyperparameter tuning using GridSearchCV
        inner_kf = KFold(n_splits=k_inner, shuffle=True, random_state=42)
        grid_search = GridSearchCV(
            model, param_grid, cv=inner_kf, scoring="neg_mean_squared_error",␣
 ↪verbose=3
        )
        grid_search.fit(X_train_outer, y_train_outer)

        # Store the grid search results
        param_combinations.extend(grid_search.cv_results_["params"])
        mse_values.extend(grid_search.cv_results_["mean_test_score"])

        # Get the best model
        best_model = grid_search.best_estimator_

        # Predictions on the outer fold's test set
        y_pred_outer = best_model.predict(X_test_outer)
        outer_mses.append(mean_squared_error(y_test_outer, y_pred_outer))

    return np.mean(outer_mses), param_combinations, mse_values


# Parameter grid for RandomForest
param_grid = {
    "n_estimators": [100, 200, 300],  # Number of trees
    "max_depth": [40, 50, 60],  # Depth of trees
    "min_samples_split": [2, 5, 10],  # Minimum samples to split
    "n_jobs": [-1],  # Use all processors
    "max_features": ["sqrt"],  # Number of features to consider for the best␣
 ↪split
}

# Preprocess the data
X_train, X_test, y_train, y_test = preprocess_data(df, features, target)

# Perform Nested Cross-Validation for RandomForest
nested_mse, param_combinations, mse_values = nested_cv(
    RandomForestRegressor(), param_grid, X_train, y_train
)
```

```python
# Output the best parameters and the nested cross-validation MSE
print(f"Nested CV Mean MSE: {nested_mse}")

# Convert the results into a DataFrame for easier plotting
results_df = pd.DataFrame(param_combinations)
results_df["mse"] = mse_values
```

```
Outer loop:   0%|          | 0/5 [00:00<?, ?it/s]

Fitting 3 folds for each of 27 candidates, totalling 81 fits
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.504 total time=   1.7s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.968 total time=   1.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.265 total time=   1.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-213.660 total time=   3.2s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-215.994 total time=   2.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-215.113 total time=   2.8s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-212.828 total time=   4.2s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-214.997 total time=   4.0s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-215.473 total time=   4.0s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-215.788 total time=   1.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-218.694 total time=   1.3s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-218.468 total time=   1.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-215.356 total time=   2.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-217.987 total time=   2.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-217.326 total time=   2.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-215.232 total time=   3.8s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-217.587 total time=   4.0s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-217.622 total time=   3.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-217.694 total time=   1.2s
```

```
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-220.707 total time=   1.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-221.360 total time=   1.1s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-218.102 total time=   2.1s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-220.019 total time=   2.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-219.984 total time=   2.2s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-217.851 total time=   3.2s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-219.468 total time=   3.2s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-219.610 total time=   3.2s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-209.011 total time=   1.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-211.628 total time=   1.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.890 total time=   1.4s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-209.512 total time=   2.7s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-211.371 total time=   2.7s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-212.176 total time=   2.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-208.901 total time=   3.9s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-210.629 total time=   4.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-212.255 total time=   3.9s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-211.095 total time=   1.3s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.787 total time=   1.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-214.295 total time=   1.2s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-211.067 total time=   2.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-212.645 total time=   2.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-213.176 total time=   2.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-210.142 total time=   3.6s
```

```
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.477 total time=   3.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-212.754 total time=   3.8s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-212.741 total time=   1.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.435 total time=   1.3s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.953 total time=   1.3s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-212.506 total time=   2.1s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-215.620 total time=   2.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-216.054 total time=   2.3s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.076 total time=   3.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-215.221 total time=   4.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-215.600 total time=   3.4s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-207.790 total time=   1.5s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.925 total time=   1.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-212.288 total time=   1.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-207.911 total time=   2.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-208.202 total time=   2.8s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-209.811 total time=   2.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-206.907 total time=   4.9s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-208.510 total time=   4.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-210.147 total time=   4.1s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-209.629 total time=   1.6s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-210.904 total time=   1.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.202 total time=   1.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-208.439 total time=   2.7s
```

```
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-210.952 total time=   2.9s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-211.507 total time=   2.9s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-207.665 total time=   4.3s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-210.005 total time=   5.1s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-210.842 total time=   5.0s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-211.811 total time=   1.6s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-214.097 total time=   1.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.782 total time=   1.6s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-211.416 total time=   3.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-213.257 total time=   3.0s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-212.927 total time=   3.0s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-210.513 total time=   4.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-212.763 total time=   4.4s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.429 total time=   4.2s
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.715 total time=   1.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.470 total time=   1.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.578 total time=   1.6s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-215.727 total time=   3.0s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-212.183 total time=   3.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-214.996 total time=   3.0s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-215.829 total time=   4.2s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-211.171 total time=   4.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-214.912 total time=   4.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
```

```
n_estimators=100, n_jobs=-1;, score=-218.049 total time=   1.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-213.712 total time=   1.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-217.550 total time=   1.6s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-218.612 total time=   2.8s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-212.791 total time=   2.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-217.898 total time=   2.7s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-217.744 total time=   4.2s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.680 total time=   3.9s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-217.297 total time=   4.1s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-220.804 total time=   1.3s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.733 total time=   1.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-220.733 total time=   1.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-221.179 total time=   2.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-215.359 total time=   2.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-219.641 total time=   2.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-220.610 total time=   3.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-215.001 total time=   3.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-220.447 total time=   4.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-212.625 total time=   1.6s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-208.241 total time=   1.7s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.133 total time=   1.7s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-211.629 total time=   3.3s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-207.237 total time=   3.2s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-211.625 total time=   2.9s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
```

```
n_estimators=300, n_jobs=-1;, score=-211.494 total time=   4.7s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-206.773 total time=   5.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-211.019 total time=   5.3s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-214.423 total time=   1.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-209.612 total time=   1.8s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.930 total time=   1.8s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-214.561 total time=   2.9s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-208.461 total time=   3.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-213.160 total time=   3.1s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-213.274 total time=   4.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-207.853 total time=   4.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-213.175 total time=   4.3s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-217.330 total time=   1.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-211.648 total time=   1.5s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-216.626 total time=   1.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-216.929 total time=   2.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-210.735 total time=   2.8s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-215.026 total time=   2.9s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-216.743 total time=   4.0s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-211.006 total time=   4.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-215.869 total time=   4.0s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.364 total time=   1.9s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-206.960 total time=   1.9s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.611 total time=   1.9s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
```

```
n_estimators=200, n_jobs=-1;, score=-209.711 total time=   3.5s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-205.334 total time=   3.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-210.264 total time=   3.4s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-209.877 total time=   5.3s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-205.318 total time=   5.1s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-209.909 total time=   5.0s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.125 total time=   1.6s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-207.995 total time=   1.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-211.799 total time=   1.6s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-212.033 total time=   3.0s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-206.147 total time=   3.0s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-211.780 total time=   3.0s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.133 total time=   4.4s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-206.112 total time=   4.3s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.207 total time=   4.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.684 total time=   1.4s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-209.664 total time=   1.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.029 total time=   1.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-215.091 total time=   2.8s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-208.601 total time=   2.8s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-213.662 total time=   2.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.950 total time=   4.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-208.909 total time=   4.1s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.667 total time=   4.1s
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```

```
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-211.491 total time=   1.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-212.729 total time=   1.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.397 total time=   1.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-211.256 total time=   2.9s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-211.839 total time=   2.8s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-212.187 total time=   2.9s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-211.284 total time=   4.3s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-210.856 total time=   4.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-212.203 total time=   4.2s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-213.719 total time=   1.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-214.286 total time=   1.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-216.206 total time=   1.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-212.564 total time=   2.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-213.397 total time=   2.8s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-215.292 total time=   2.8s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-212.668 total time=   4.0s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-212.823 total time=   4.0s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-215.056 total time=   4.0s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-214.955 total time=   1.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-217.464 total time=   1.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-219.345 total time=   1.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-215.062 total time=   2.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-216.717 total time=   2.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-218.908 total time=   2.5s
```

```
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-214.491 total time=   3.7s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-216.738 total time=   3.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-218.119 total time=   3.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-207.951 total time=   1.7s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-207.045 total time=   1.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.000 total time=   1.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-206.291 total time=   3.0s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-206.495 total time=   3.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-208.590 total time=   3.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-206.650 total time=   4.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-206.828 total time=   4.7s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-208.261 total time=   4.7s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-208.077 total time=   1.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-209.086 total time=   1.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-210.748 total time=   1.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-207.020 total time=   2.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-209.226 total time=   2.8s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-210.003 total time=   2.8s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-208.030 total time=   4.0s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-208.504 total time=   4.1s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-209.549 total time=   3.9s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-211.195 total time=   1.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-212.173 total time=   1.3s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-213.525 total time=   1.4s
```

```
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-210.583 total time=   2.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-211.916 total time=   2.7s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-214.145 total time=   2.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-210.079 total time=   3.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-211.671 total time=   3.8s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.865 total time=   3.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-205.789 total time=   1.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-205.595 total time=   1.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-207.131 total time=   1.7s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-205.764 total time=   3.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-205.151 total time=   3.2s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-207.257 total time=   3.3s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-205.113 total time=   4.6s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-204.461 total time=   4.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-205.970 total time=   4.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-206.564 total time=   1.6s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-206.552 total time=   1.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-209.781 total time=   1.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-206.175 total time=   2.9s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-206.377 total time=   3.1s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-208.897 total time=   2.9s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-205.322 total time=   4.4s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-205.812 total time=   4.4s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-207.819 total time=   4.3s
```

```
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-209.167 total time=   1.5s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-210.259 total time=   1.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-211.930 total time=   1.4s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-209.077 total time=   2.8s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-209.554 total time=   2.9s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-211.713 total time=   2.7s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-207.947 total time=   4.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-209.453 total time=   4.2s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-211.144 total time=   4.0s
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.307 total time=   1.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.846 total time=   1.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-217.644 total time=   1.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-216.159 total time=   2.8s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-212.982 total time=   2.9s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-216.208 total time=   3.0s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-214.867 total time=   4.3s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-212.088 total time=   4.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-216.379 total time=   4.3s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-217.290 total time=   1.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-214.522 total time=   1.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-218.639 total time=   1.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-216.649 total time=   2.8s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-214.449 total time=   2.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
```

```
n_estimators=200, n_jobs=-1;, score=-218.447 total time=   2.7s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-216.416 total time=   3.9s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-213.745 total time=   4.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-217.795 total time=   3.9s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-219.866 total time=   1.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-217.594 total time=   1.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-222.195 total time=   1.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-219.545 total time=   2.6s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-216.896 total time=   2.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-220.782 total time=   2.6s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-219.065 total time=   3.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-216.495 total time=   3.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-220.644 total time=   3.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-211.781 total time=   1.6s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.144 total time=   1.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.684 total time=   1.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-209.985 total time=   3.3s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-208.672 total time=   3.1s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-212.442 total time=   3.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-209.926 total time=   4.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-207.839 total time=   4.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-212.491 total time=   4.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.346 total time=   1.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-211.193 total time=   1.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
```

```
n_estimators=100, n_jobs=-1;, score=-214.354 total time=   1.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-211.266 total time=   2.7s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-209.491 total time=   2.7s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-213.701 total time=   2.7s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.721 total time=   4.1s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-209.551 total time=   4.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-213.488 total time=   4.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-214.887 total time=   1.4s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-213.369 total time=   1.4s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-217.448 total time=   1.4s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-214.703 total time=   2.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-212.104 total time=   2.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-216.435 total time=   2.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-214.079 total time=   3.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-212.069 total time=   3.8s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-216.646 total time=   3.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-209.477 total time=   1.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-207.450 total time=   1.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-212.553 total time=   1.7s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-207.517 total time=   3.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-206.311 total time=   3.2s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-210.583 total time=   3.2s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-207.507 total time=   4.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-205.762 total time=   4.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
```

```
n_estimators=300, n_jobs=-1;, score=-210.478 total time=   4.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-210.661 total time=   1.5s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-207.904 total time=   1.6s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.968 total time=   1.6s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-209.534 total time=   2.9s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-207.390 total time=   2.9s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-212.106 total time=   2.9s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-208.985 total time=   4.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-207.604 total time=   4.2s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-211.435 total time=   4.2s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-212.610 total time=   1.4s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-210.896 total time=   1.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-215.148 total time=   1.4s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-212.082 total time=   2.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-210.961 total time=   2.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-214.330 total time=   2.7s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-211.910 total time=   4.0s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-209.792 total time=   4.0s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-213.716 total time=   4.0s
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-220.892 total time=   1.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-210.571 total time=   1.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-213.789 total time=   1.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-220.693 total time=   2.7s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-209.903 total time=   2.7s
```

```
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-213.164 total time=   2.7s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-219.583 total time=   4.1s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-209.165 total time=   4.1s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-213.086 total time=   4.0s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-221.975 total time=   1.3s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-212.586 total time=   1.3s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-215.903 total time=   1.3s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-221.908 total time=   2.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-211.285 total time=   2.6s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-215.477 total time=   2.5s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-220.967 total time=   3.7s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-210.630 total time=   3.7s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-214.716 total time=   3.7s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-225.790 total time=   1.3s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-214.872 total time=   1.3s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-218.342 total time=   1.3s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-225.077 total time=   2.4s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-214.751 total time=   2.4s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-217.723 total time=   2.4s
[CV 1/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-224.233 total time=   3.5s
[CV 2/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-214.784 total time=   3.5s
[CV 3/3] END max_depth=40, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-217.872 total time=   3.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-216.472 total time=   1.6s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-205.608 total time=   1.6s
```

```
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-208.930 total time=   1.7s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-215.847 total time=   3.2s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-204.839 total time=   3.2s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-209.509 total time=   3.2s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-214.931 total time=   4.8s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-203.913 total time=   4.9s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-208.335 total time=   5.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-217.812 total time=   1.6s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-206.499 total time=   1.6s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-210.705 total time=   1.6s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-217.013 total time=   2.9s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-206.047 total time=   3.0s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-210.823 total time=   3.0s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-216.665 total time=   4.3s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-205.985 total time=   4.3s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-210.505 total time=   4.4s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-220.465 total time=   1.5s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-209.386 total time=   1.5s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-214.105 total time=   1.5s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-220.277 total time=   2.7s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-208.845 total time=   2.9s
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-213.543 total time=   2.9s
[CV 1/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-219.920 total time=   4.2s
[CV 2/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-209.281 total time=   4.1s
```

```
[CV 3/3] END max_depth=50, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-212.892 total time=   4.1s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-215.051 total time=   1.8s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-203.413 total time=   1.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=100, n_jobs=-1;, score=-208.023 total time=   1.8s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-214.165 total time=   3.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-202.781 total time=   3.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=200, n_jobs=-1;, score=-206.756 total time=   3.6s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-214.366 total time=   5.4s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-202.253 total time=   5.5s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=2,
n_estimators=300, n_jobs=-1;, score=-206.719 total time=   5.5s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-215.401 total time=   1.7s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-204.134 total time=   1.7s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=100, n_jobs=-1;, score=-210.453 total time=   1.7s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-215.768 total time=   3.2s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-204.012 total time=   3.3s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=200, n_jobs=-1;, score=-207.732 total time=   3.1s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-215.134 total time=   4.0s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-203.758 total time=   3.8s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=5,
n_estimators=300, n_jobs=-1;, score=-208.707 total time=   3.4s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-218.437 total time=   1.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-208.077 total time=   1.2s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=100, n_jobs=-1;, score=-212.172 total time=   1.1s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-218.374 total time=   2.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-206.948 total time=   2.1s
```

```
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=200, n_jobs=-1;, score=-211.413 total time=   2.1s
[CV 1/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-217.411 total time=   3.1s
[CV 2/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-206.932 total time=   3.1s
[CV 3/3] END max_depth=60, max_features=sqrt, min_samples_split=10,
n_estimators=300, n_jobs=-1;, score=-211.183 total time=   3.1s
Nested CV Mean MSE: 198.1884508838352
```