Results for Hyperparameter Tuning

**-Hyperparameter tuning using GridSearchCV**

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

# Separate features and target

X = cleaned\_hard\_data\_df.drop(columns=['challenge\_rating']) # Features

y = cleaned\_hard\_data\_df['challenge\_rating'] # Target

# Identify categorical and numeric columns

categorical\_columns = X.select\_dtypes(include=['object']).columns

numeric\_columns = X.select\_dtypes(include=['float64', 'int64']).columns

# Preprocessing pipelines for numeric and categorical data

numeric\_transformer = StandardScaler() # Feature scaling (standardization)

categorical\_transformer = OneHotEncoder(handle\_unknown='ignore')

# Combine preprocessors in a column transformer

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_columns), # Scaling numeric features

('cat', categorical\_transformer, categorical\_columns) # One-hot encoding categorical features

])

# Create a pipeline for Ridge Regression

ridge\_model = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', Ridge()) # Ridge regression model

])

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Hyperparameter grid for Ridge regression

param\_grid = {

'regressor\_\_alpha': np.logspace(-5, 5, 10) # Search between 10^-5 to 10^5 with 10 points

}

# Perform grid search with cross-validation

grid\_search = GridSearchCV(ridge\_model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

# Best hyperparameters from GridSearchCV

print("Best Hyperparameters:", grid\_search.best\_params\_)

# Train the model using the best found hyperparameters

best\_ridge\_model = grid\_search.best\_estimator\_

# Make predictions on the test set

y\_pred = best\_ridge\_model.predict(X\_test)

# Evaluate the Ridge Regression model

print("Ridge Regression Model Performance:")

print("Mean Absolute Error (MAE):", mean\_absolute\_error(y\_test, y\_pred))

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

print("R² Score:", r2\_score(y\_test, y\_pred))

Best Hyperparameters: {'preprocessor\_\_num\_\_scaler': StandardScaler(), 'regressor': Ridge(), 'regressor\_\_alpha': 1}

Best Cross-validation score (negative MSE): -2.732665644115481

Evaluation on Test Set:

Mean Absolute Error (MAE): 0.8560240934731251

Mean Squared Error (MSE): 1.2787944443858021

R² Score: 0.9593575246436504

**- Hyperparameter Tuning with Ridge Regression: Hyperparameter Tuning Using GridSearchCV**

#Hyperparameter Tuning with Ridge Regression: Hyperparameter Tuning Using GridSearchCV

# from sklearn.linear\_model import Ridge

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

# Separate features and target

X = cleaned\_hard\_data\_df.drop(columns=['challenge\_rating']) # Features

y = cleaned\_hard\_data\_df['challenge\_rating'] # Target

# Identify categorical and numeric columns

categorical\_columns = X.select\_dtypes(include=['object']).columns

numeric\_columns = X.select\_dtypes(include=['float64', 'int64']).columns

# Preprocessing pipelines for numeric and categorical data

numeric\_transformer = StandardScaler() # Feature scaling (standardization)

categorical\_transformer = OneHotEncoder(handle\_unknown='ignore')

# Combine preprocessors in a column transformer

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_columns), # Scaling numeric features

('cat', categorical\_transformer, categorical\_columns) # One-hot encoding categorical features

])

# Create a pipeline for Ridge Regression

ridge\_model = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', Ridge()) # Ridge regression model

])

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Hyperparameter grid for Ridge regression

param\_grid = {

'regressor\_\_alpha': [0.01, 0.1, 1, 10, 100, 1000], # Range of alpha values

}

# Perform grid search with cross-validation

grid\_search = GridSearchCV(ridge\_model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

# Best hyperparameters from GridSearchCV

print("Best Hyperparameters:", grid\_search.best\_params\_)

# Train the model using the best found hyperparameters

best\_ridge\_model = grid\_search.best\_estimator\_

# Make predictions on the test set

y\_pred = best\_ridge\_model.predict(X\_test)

# Evaluate the Ridge Regression model

print("Ridge Regression Model Performance:")

print("Mean Absolute Error (MAE):", mean\_absolute\_error(y\_test, y\_pred))

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

print("R² Score:", r2\_score(y\_test, y\_pred))

Best Hyperparameters: {'regressor\_\_alpha': 1}

Ridge Regression Model Performance:

Mean Absolute Error (MAE): 0.8560232445008368

Mean Squared Error (MSE): 1.278799153265392

R² Score: 0.9593573749866641

**- Hyperparameter Tuning with Ridge Regression: Hyperparameter Tuning Using GridSearchCV**

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

# Separate features and target

X = cleaned\_hard\_data\_df.drop(columns=['challenge\_rating']) # Features

y = cleaned\_hard\_data\_df['challenge\_rating'] # Target

# Identify categorical and numeric columns

categorical\_columns = X.select\_dtypes(include=['object']).columns

numeric\_columns = X.select\_dtypes(include=['float64', 'int64']).columns

# Preprocessing pipelines for numeric and categorical data

numeric\_transformer = StandardScaler() # Feature scaling (standardization)

categorical\_transformer = OneHotEncoder(handle\_unknown='ignore')

# Combine preprocessors in a column transformer

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_columns), # Scaling numeric features

('cat', categorical\_transformer, categorical\_columns) # One-hot encoding categorical features

])

# Create a pipeline for Ridge Regression

ridge\_model = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', Ridge()) # Ridge regression model

])

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Hyperparameter grid for Ridge regression

param\_grid = {

'regressor\_\_alpha': np.logspace(-5, 5, 10) # Search between 10^-5 to 10^5 with 10 points

}

# Perform grid search with cross-validation

grid\_search = GridSearchCV(ridge\_model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

# Best hyperparameters from GridSearchCV

print("Best Hyperparameters:", grid\_search.best\_params\_)

# Train the model using the best found hyperparameters

best\_ridge\_model = grid\_search.best\_estimator\_

# Make predictions on the test set

y\_pred = best\_ridge\_model.predict(X\_test)

# Evaluate the Ridge Regression model

print("Ridge Regression Model Performance:")

print("Mean Absolute Error (MAE):", mean\_absolute\_error(y\_test, y\_pred))

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred))

print("R² Score:", r2\_score(y\_test, y\_pred))

Best Hyperparameters: {'regressor\_\_alpha': 3.593813663804626}

Ridge Regression Model Performance:

Mean Absolute Error (MAE): 0.8806310525094502

Mean Squared Error (MSE): 1.327645456560991

R² Score: 0.9578049482564294

**- Using RandomizedSearchCV for Faster Hyperparameter Tuning**

from sklearn.model\_selection import RandomizedSearchCV

import numpy as np

# Define the parameter distribution for alpha

param\_dist = {

'regressor\_\_alpha': np.logspace(-5, 5, 100) # Random search over a log scale of alpha values

}

# Perform RandomizedSearchCV

random\_search = RandomizedSearchCV(ridge\_model, param\_dist, n\_iter=10, cv=5, scoring='neg\_mean\_squared\_error', random\_state=42)

random\_search.fit(X\_train, y\_train)

# Get best hyperparameters

best\_alpha\_random = random\_search.best\_params\_['regressor\_\_alpha']

print(f"Best Alpha (Randomized Search): {best\_alpha\_random}")

# Evaluate the best model

best\_model\_random = random\_search.best\_estimator\_

y\_pred\_random = best\_model\_random.predict(X\_test)

print("Evaluation on Test Set (Randomized Search):")

print("Mean Absolute Error (MAE):", mean\_absolute\_error(y\_test, y\_pred\_random))

print("Mean Squared Error (MSE):", mean\_squared\_error(y\_test, y\_pred\_random))

print("R² Score:", r2\_score(y\_test, y\_pred\_random))

Best Alpha (Randomized Search): 2.2570197196339215

Evaluation on Test Set (Randomized Search):

Mean Absolute Error (MAE): 0.8708208350493573

Mean Squared Error (MSE): 1.3053797237078977

R² Score: 0.9585125948236676