Import the library

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

upload the dataset

```python
# Load the data
train_data= pd.read_csv("train.csv")
test_data= pd.read_csv("test.csv")
```

train_data

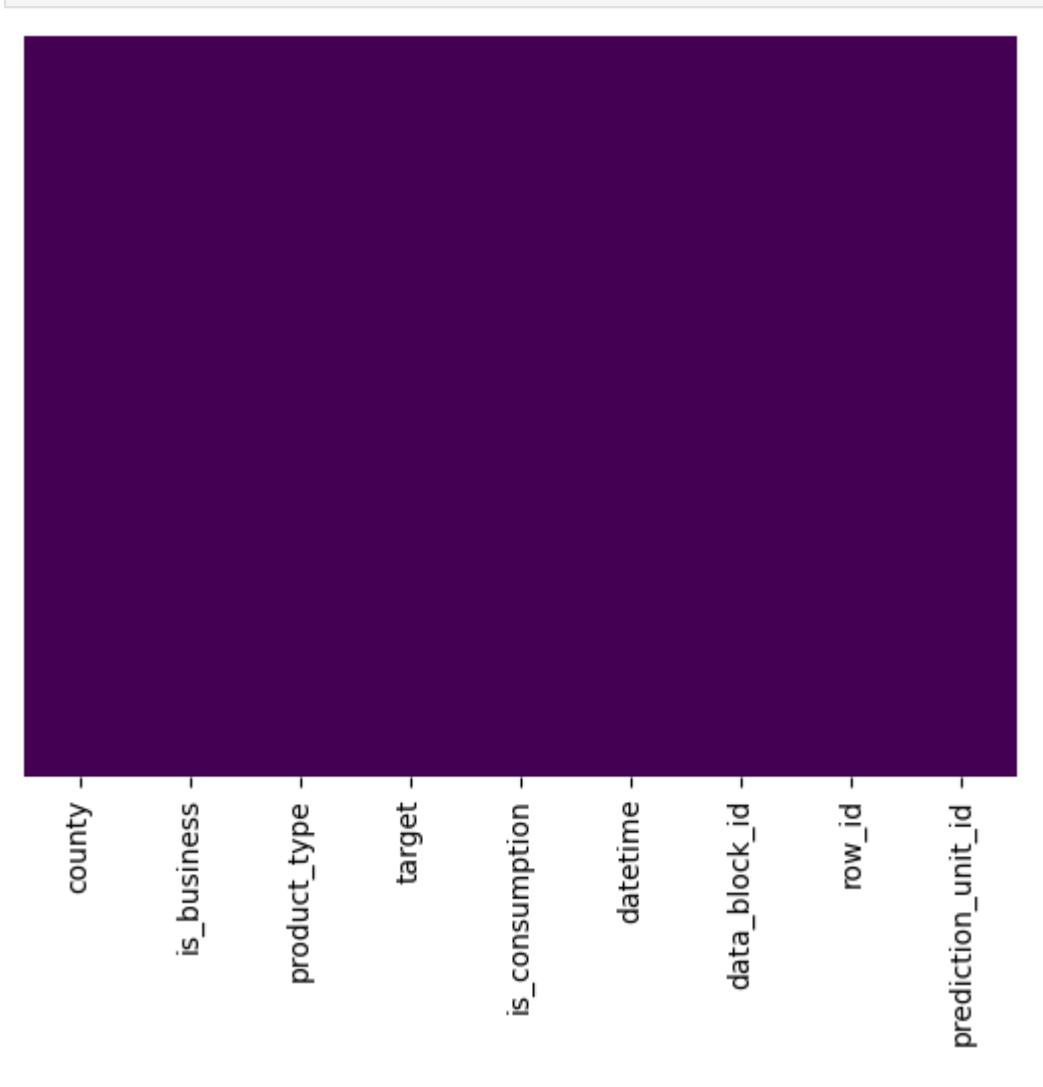
```python
# Display basic information about the training data
print(train_data.head())
print(train_data.info())
print(train_data.describe())
print(train_data.isnull().sum())
print(train_data.shape)
print(train_data.columns)
```

```
   county  is_business  product_type  target  is_consumption  \
0       0            0             1   0.713               0
1       0            0             1  96.590               1
2       0            0             2   0.000               0
3       0            0             2  17.314               1
4       0            0             3   2.904               0

              datetime  data_block_id  row_id  prediction_unit_id
0  2021-09-01 00:00:00              0       0                   0
1  2021-09-01 00:00:00              0       1                   0
2  2021-09-01 00:00:00              0       2                   1
3  2021-09-01 00:00:00              0       3                   1
4  2021-09-01 00:00:00              0       4                   2
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2018352 entries, 0 to 2018351
Data columns (total 9 columns):
 #   Column              Dtype
---  ------              -----
 0   county              int64
 1   is_business         int64
 2   product_type        int64
 3   target              float64
 4   is_consumption      int64
 5   datetime            object
 6   data_block_id       int64
 7   row_id              int64
 8   prediction_unit_id  int64
dtypes: float64(1), int64(7), object(1)
memory usage: 138.6+ MB
None
             county   is_business  product_type        target  is_consumption  \
count  2.018352e+06  2.018352e+06  2.018352e+06  2.017824e+06       2018352.0
mean   7.297034e+00  5.368261e-01  1.898927e+00  2.748556e+02             0.5
std    4.780990e+00  4.986421e-01  1.081766e+00  9.095024e+02             0.5
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00             0.0
25%    3.000000e+00  0.000000e+00  1.000000e+00  3.780000e-01             0.0
50%    7.000000e+00  1.000000e+00  2.000000e+00  3.113300e+01             0.5
75%    1.100000e+01  1.000000e+00  3.000000e+00  1.802062e+02             1.0
max    1.500000e+01  1.000000e+00  3.000000e+00  1.548027e+04             1.0

       data_block_id        row_id  prediction_unit_id
count   2.018352e+06  2.018352e+06        2.018352e+06
mean    3.218746e+02  1.009176e+06        3.304538e+01
std     1.826343e+02  5.826482e+05        1.959059e+01
min     0.000000e+00  0.000000e+00        0.000000e+00
25%     1.660000e+02  5.045878e+05        1.600000e+01
50%     3.230000e+02  1.009176e+06        3.300000e+01
75%     4.790000e+02  1.513763e+06        5.000000e+01
max     6.370000e+02  2.018351e+06        6.800000e+01
county                  0
is_business             0
product_type            0
target                528
is_consumption          0
datetime                0
data_block_id           0
row_id                  0
prediction_unit_id      0
dtype: int64
(2018352, 9)
Index(['county', 'is_business', 'product_type', 'target', 'is_consumption',
       'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'],
      dtype='object')
```

```python
sample_train_data = train_data.sample(frac=0.1)  # Adjust the fraction as needed
sns.heatmap(sample_train_data.isnull(), yticklabels=False, cbar=False, cmap="viridis")
plt.show()
```



Handle missing values

```python
train_data.fillna(0, inplace=True)
test_data.fillna(0, inplace=True)
```

Split features and target variable

```python
X = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                     'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
y = train_data['prediction_unit_id']
```

Split the data into training and validation sets

```python
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Check for NaN or Infinity values in y_train

```python
print("NaN values in y_train:", pd.DataFrame(y_train).isnull().sum())
print("Infinity values in y_train:", np.isinf(y_train).sum())
```

```
NaN values in y_train: prediction_unit_id    0
dtype: int64
Infinity values in y_train: 0
```

```python
print(X_train.dtypes)
print(y_train.dtypes)
```

```
Series([], dtype: object)
int64
```

```python
print("NaN values in X_train:", np.isnan(X_train).sum())
print("Infinity values in X_train:", np.isinf(X_train).sum())

print("NaN values in y_train:", np.isnan(y_train).sum())
print("Infinity values in y_train:", np.isinf(y_train).sum())
```

```
NaN values in X_train: Series([], dtype: float64)
Infinity values in X_train: Series([], dtype: float64)
NaN values in y_train: 0
Infinity values in y_train: 0
```

```python
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```python
y_train = y_train.astype(float)  # Convert to float if needed
```

```python
print(X_train.shape)
print(y_train.shape)
```

```
(1614681, 0)
(1614681,)
```

```python
print("NaN values in X_train:", np.isnan(X_train).sum())
print("Infinity values in X_train:", np.isinf(X_train).sum())
```

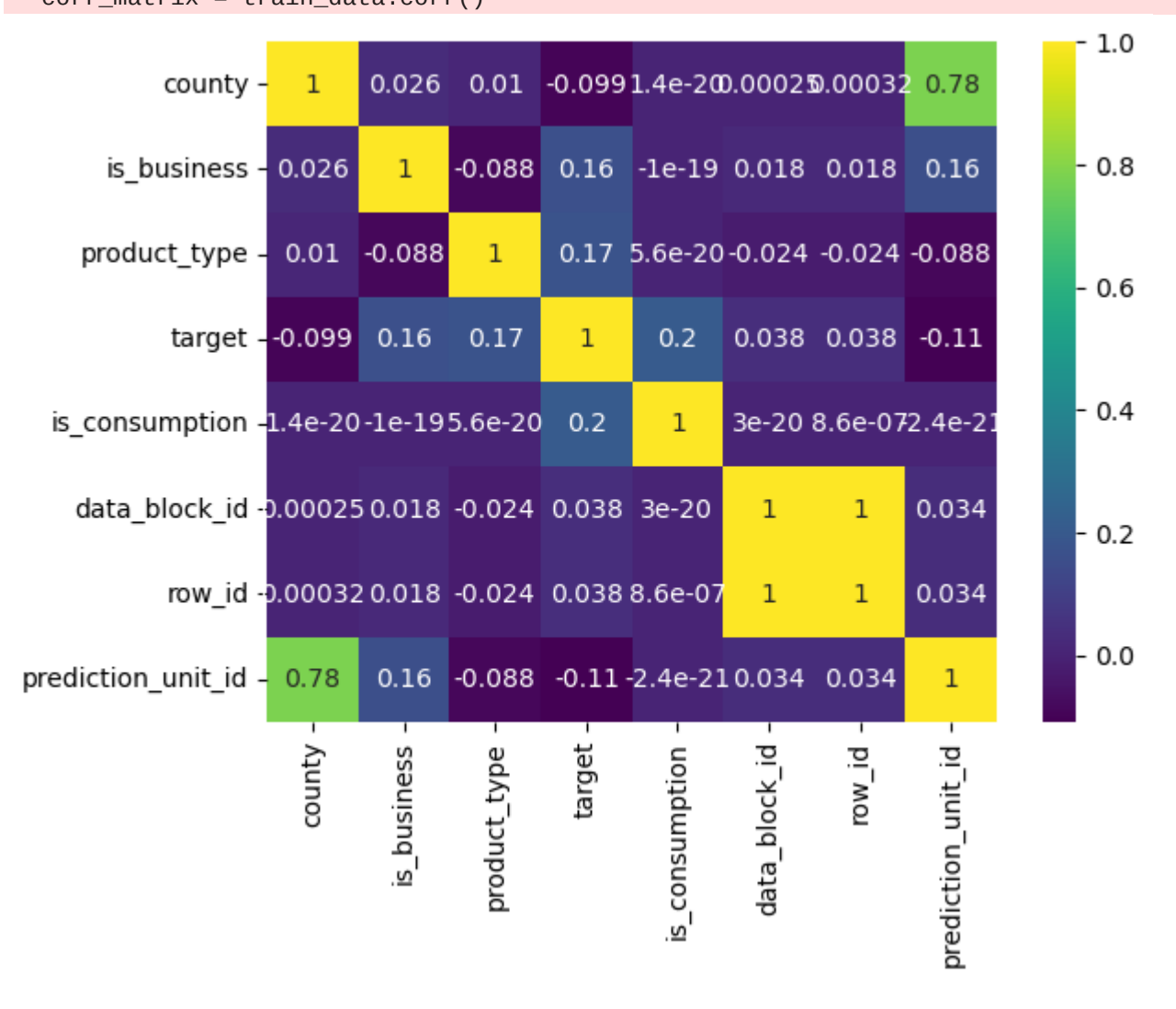
```
NaN values in X_train: 0
Infinity values in X_train: 0
```

```python
print("NaN values in y_train:", np.isnan(y_train).sum())
print("Infinity values in y_train:", np.isinf(y_train).sum())
```

```
NaN values in y_train: 0
Infinity values in y_train: 0
```

```python
corr_matrix = train_data.corr()
sns.heatmap(corr_matrix, annot=True, cmap="viridis")
plt.show()
```

```
C:\Users\jmair\AppData\Local\Temp\ipykernel_16460\3738285398.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr_matrix = train_data.corr()
```




```python
X = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                     'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
```

```python
# Split features and target variable
X = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                     'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
y = train_data['prediction_unit_id']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Drop specified columns from features
X = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                     'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
y = train_data['prediction_unit_id']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
from sklearn.impute import SimpleImputer

# Display information about X_train before imputation
print("Before Imputation:")
print(X_train.info())

# Check if there are any missing values in X_train
print("\nMissing Values in X_train:")
print(X_train.isnull().sum())

# Handle missing values in X_train only if it's not empty
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)

    # Display information about X_train_imputed after imputation
    print("\nAfter Imputation:")
    print(X_train_imputed.info())

    # Perform grid search
    grid_search = GridSearchCV(model, param_grid, cv=3, scoring='neg_mean_absolute_error')
    grid_search.fit(X_train_imputed, y_train)

    # Get the best hyperparameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)
else:
    print("X_train is empty. Check your data filtering or subsetting.")
```

```
Before Imputation:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1614681 entries, 1336406 to 121958
Empty DataFrame
None

Missing Values in X_train:
Series([], dtype: float64)
X_train is empty. Check your data filtering or subsetting.
```

```python
print("Before Imputation:")
print(X_train.info())
```

```
Before Imputation:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1614681 entries, 1336406 to 121958
Empty DataFrame
None
```

```python
print("\nMissing Values in X_train:")
print(X_train.isnull().sum())
```

```
Missing Values in X_train:
Series([], dtype: float64)
```

```python
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
```

```python
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
```

```python
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)
```

```python
# Display information about X_train before imputation
print("Before Imputation:")
print("Shape of X_train:", X_train.shape)
print("Is X_train empty?", X_train.empty)
print("\nMissing Values in X_train:")
print(X_train.isnull().sum())

# Handle missing values in X_train only if it's not empty
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)

    # Display information about X_train_imputed after imputation
    print("\nAfter Imputation:")
    print("Shape of X_train_imputed:", X_train_imputed.shape)
    print(X_train_imputed.info())

    # Perform grid search
    grid_search = GridSearchCV(model, param_grid, cv=3, scoring='neg_mean_absolute_error')
    grid_search.fit(X_train_imputed, y_train)

    # Get the best hyperparameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)
else:
    print("X_train is empty. Check your data filtering or subsetting.")
```

```
Before Imputation:
Shape of X_train: (1614681, 0)
Is X_train empty? True

Missing Values in X_train:
Series([], dtype: float64)
X_train is empty. Check your data filtering or subsetting.
```

```python
# Split features and target variable
X = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                     'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
y = train_data['prediction_unit_id']
```

```python
# Check the columns in the original DataFrame
print("Columns in train_data:", train_data.columns)

# Split features and target variable
X = train_data.drop(['prediction_unit_id'], axis=1)  # Keep all other columns except 'prediction_unit_id'
y = train_data['prediction_unit_id']

# Display the first few rows of X to verify it contains the expected features
print("X:")
print(X.head())
```

```
Columns in train_data: Index(['county', 'is_business', 'product_type', 'target', 'is_consumption',
       'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'],
      dtype='object')
X:
   county  is_business  product_type  target  is_consumption  \
0       0            0             1   0.713               0
1       0            0             1  96.590               1
2       0            0             2   0.000               0
3       0            0             2  17.314               1
4       0            0             3   2.904               0

              datetime  data_block_id  row_id
0  2021-09-01 00:00:00              0       0
1  2021-09-01 00:00:00              0       1
2  2021-09-01 00:00:00              0       2
3  2021-09-01 00:00:00              0       3
4  2021-09-01 00:00:00              0       4
```

```python
# Handle missing values in X_train only if it's not empty
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)

    # Display information about X_train_imputed after imputation
    print("\nAfter Imputation:")
    print("Shape of X_train_imputed:", X_train_imputed.shape)
    print(X_train_imputed.info())

    # Perform grid search
    grid_search = GridSearchCV(model, param_grid, cv=3, scoring='neg_mean_absolute_error')
    grid_search.fit(X_train_imputed, y_train)

    # Get the best hyperparameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)
else:
    print("X_train is empty. Check your data filtering or subsetting.")
```

```
X_train is empty. Check your data filtering or subsetting.
```

```python
print("Shape of X_train:", X_train.shape)
```

```
Shape of X_train: (1614681, 0)
```

```python
print(X_train.head())
```

```
Empty DataFrame
Columns: []
Index: [1336406, 913275, 1439080, 1257627, 1242815]
```

```python
print("Columns in train_data:", train_data.columns)
```

```
Columns in train_data: Index(['county', 'is_business', 'product_type', 'target', 'is_consumption',
       'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'],
      dtype='object')
```

```python
print("First few rows of train_data:")
print(train_data.head())
```

```
First few rows of train_data:
   county  is_business  product_type  target  is_consumption  \
0       0            0             1   0.713               0
1       0            0             1  96.590               1
2       0            0             2   0.000               0
3       0            0             2  17.314               1
4       0            0             3   2.904               0

              datetime  data_block_id  row_id  prediction_unit_id
0  2021-09-01 00:00:00              0       0                   0
1  2021-09-01 00:00:00              0       1                   0
2  2021-09-01 00:00:00              0       2                   1
3  2021-09-01 00:00:00              0       3                   1
4  2021-09-01 00:00:00              0       4                   2
```

```python
# Split features and target variable
X_train = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                           'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
y_train = train_data['prediction_unit_id']
```

```python
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
```

```
Shape of X_train: (2018352, 0)
Shape of y_train: (2018352,)
```

```python
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("Infinity values in X_train:", np.isinf(X_train).sum().sum())

print("NaN values in y_train:", pd.DataFrame(y_train).isnull().sum().sum())
print("Infinity values in y_train:", np.isinf(y_train).sum())
```

```
NaN values in X_train: 0.0
Infinity values in X_train: 0.0
NaN values in y_train: 0
Infinity values in y_train: 0
```

```python
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
```

```
Shape of X_train: (2018352, 0)
Shape of y_train: (2018352,)
```

```python
print("NaN values in X_train:", np.isnan(X_train).sum())
print("Infinity values in X_train:", np.isinf(X_train).sum())

print("NaN values in y_train:", pd.DataFrame(y_train).isnull().sum())
print("Infinity values in y_train:", np.isinf(y_train).sum())
```

```
NaN values in X_train: Series([], dtype: float64)
Infinity values in X_train: Series([], dtype: float64)
NaN values in y_train: prediction_unit_id    0
dtype: int64
Infinity values in y_train: 0
```

```python
print("Shape of X_train:", X_train.shape)
print("Data types in X_train:", X_train.dtypes)

print("\nShape of y_train:", y_train.shape)
print("Data type of y_train:", y_train.dtypes)
```

```
Shape of X_train: (2018352, 0)
Data types in X_train: Series([], dtype: object)

Shape of y_train: (2018352,)
Data type of y_train: int64
```

```python
X_train = train_data.drop(['county', 'is_business', 'product_type', 'target', 'is_consumption',
                           'datetime', 'data_block_id', 'row_id', 'prediction_unit_id'], axis=1)
```

```python
print("Shape of X_train after drop:", X_train.shape)
print("Is X_train empty?", X_train.empty)
```

```
Shape of X_train after drop: (2018352, 0)
Is X_train empty? True
```

```python
print("Column names of X_train:")
print(X_train.columns)
```

```
Column names of X_train:
Index([], dtype='object')
```

```python
# Handle missing values in X_train only if it's not empty
if not X_train.empty:
    imputer = SimpleImputer(strategy='mean')
    X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.columns)

    # Display information about X_train_imputed after imputation
    print("\nAfter Imputation:")
    print("Shape of X_train_imputed:", X_train_imputed.shape)
    print(X_train_imputed.info())

    # Perform grid search (assuming you have defined param_grid previously)
    grid_search = GridSearchCV(model, param_grid, cv=3, scoring='neg_mean_absolute_error')
    grid_search.fit(X_train_imputed, y_train)

    # Get the best hyperparameters
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)
else:
    print("X_train is empty. Check your data filtering or subsetting.")
```

```
X_train is empty. Check your data filtering or subsetting.
```