Step 4.4: Bi-Directional Elimination for Feature Selection

To perform another round of feature selection using a wrapper method called bi-directional elimination. This method iteratively adds and removes features to identify the optimal subset for predicting the target variable (Powerall).

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR

from sklearn.metrics import mean_squared_error,r2_score

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```
# Load the Data set
df=pd.read_csv('Group_14_Clean_Data.csv')
df.head()
```

Out[2]:

	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5
0	316.5855	223.9277	182.3434	551.5497	7.8641	243.1339	361.0877	115.9284	78.6087
1	530.3136	68.7031	31.5983	175.2582	516.1441	63.4652	67.0954	369.4486	14.0930
2	27.3967	399.0488	565.6854	394.0466	120.2245	558.1293	546.4520	27.3256	314.1051
3	346.1526	59.6375	226.2742	280.9095	402.2161	218.7181	207.0407	339.5676	280.2195
4	317.9144	551.8542	335.4745	40.0240	316.6285	365.6434	416.3060	562.1028	211.3577

5 rows × 49 columns

In [3]:
```
# Define features and target variable

X = df.drop(columns=['Powerall'])
y = df['Powerall']
```

In [4]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [5]:
```
# Perform scaling after splitting
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [37]:
```
from sklearn.feature_selection import SequentialFeatureSelector

linear_model = LinearRegression()
```

In [38]:
```
# Perform Bi-Directional Elimination
bi_directional_selector = SequentialFeatureSelector(
    estimator=linear_model,
    n_features_to_select="auto",
    direction='forward',
    scoring='r2',
    cv=3
)
```

In [39]:
```
# Fit the selector
bi_directional_selector.fit(X_train_scaled, y_train)

# Selected features
selected_features = X.columns[bi_directional_selector.get_support()]
print("Selected Features:", selected_features)
```

```
Selected Features: Index(['Y3', 'Y4', 'X7', 'X12', 'Y12', 'X14', 'Y15', 'X16', 'P1',
'P2', 'P3',
       'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14',
       'P15', 'P16'],
      dtype='object')
```

In [40]:
```
# Create reduced datasets
X_train_selected = X_train_scaled[:, bi_directional_selector.get_support()]
X_test_selected = X_test_scaled[:, bi_directional_selector.get_support()]
```

In [41]:
```
models_Feature_select= {
    "Linear_Regression": LinearRegression(),
    "K_Nearest_Neighbors": KNeighborsRegressor(n_neighbors=5),
    "Random_Forest": RandomForestRegressor(n_estimators=50,max_depth= 10,random_sta
    "SVM_Linear_Kernel": SVR(kernel="linear"),
    "SVM_RBF_Kernel": SVR(kernel='rbf',C=100,gamma=0.1),
    "Gradient_Boosting": GradientBoostingRegressor(n_estimators=50, max_depth=3, ra
}
```

In [42]:
```
# Train models on selected features
final_results = {}
for model_name, model in models_Feature_select.items():
    model.fit(X_train_selected, y_train)
    y_train_pred = model.predict(X_train_selected)
    y_test_pred = model.predict(X_test_selected)
```

```python
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    final_results[model_name] = {"Train_RMSE": train_rmse, "Test_RMSE": test_rmse,
                                 "Train_R2": train_r2, "Test_R2": test_r2}
```

In [43]:

```python
# Convert results to DataFrame
final_results_df = pd.DataFrame(final_results).T
final_results_df
```

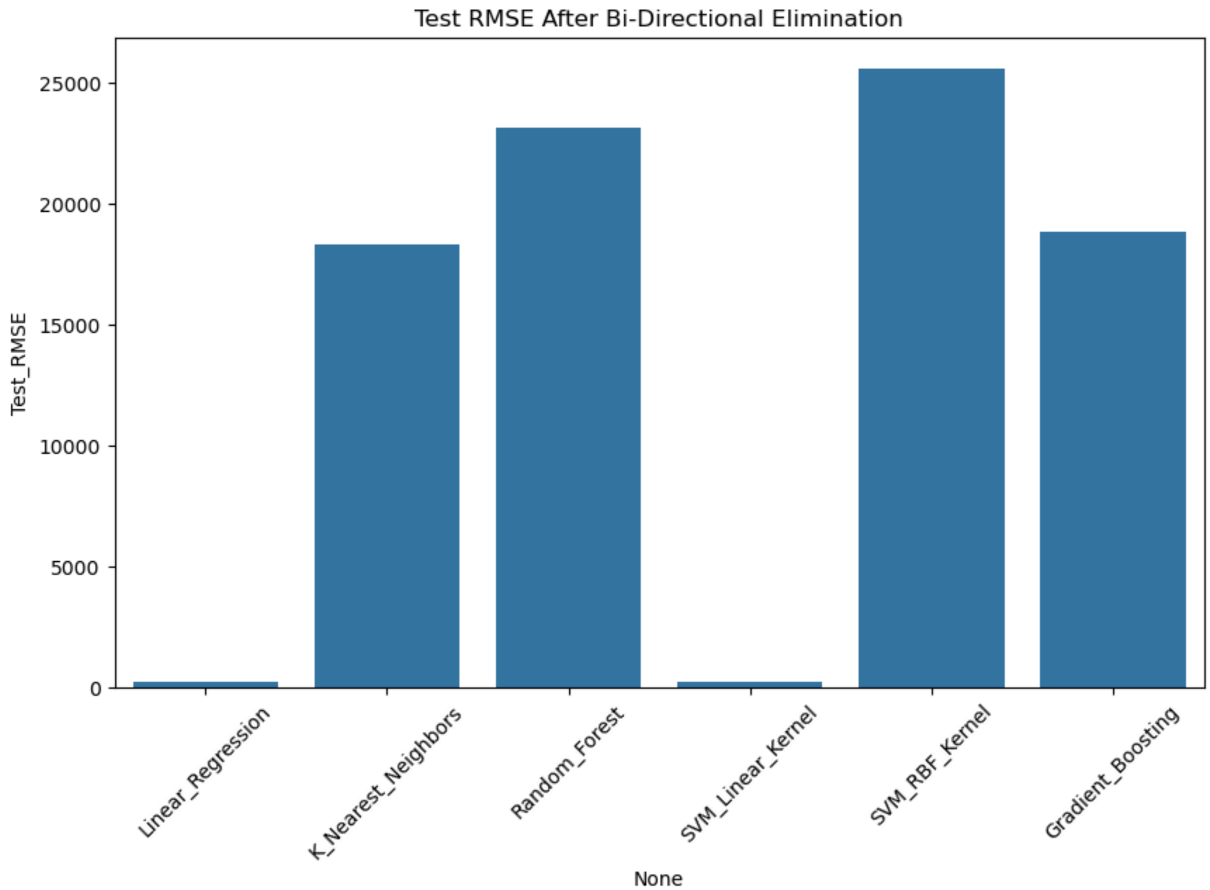
Out[43]:

	Train_RMSE	Test_RMSE	Train_R2	Test_R2
Linear_Regression	369.958181	230.296733	0.999955	0.999983
K_Nearest_Neighbors	14996.456335	18327.087436	0.926267	0.892384
Random_Forest	20847.298757	23107.040965	0.857511	0.828929
SVM_Linear_Kernel	370.072199	229.977526	0.999955	0.999983
SVM_RBF_Kernel	25315.516332	25562.867836	0.789885	0.790633
Gradient_Boosting	18310.368112	18843.290447	0.890080	0.886237

In [44]:

```python
# Visualize performance after feature selection

plt.figure(figsize=(10, 6))
sns.barplot(x=final_results_df.index, y=final_results_df["Test_RMSE"])
plt.title("Test RMSE After Bi-Directional Elimination")
plt.xticks(rotation=45)
plt.show()
```



Test RMSE After Bi-Directional Elimination
Test_RMSE
25000
20000
15000
10000
5000
0
Linear_Regression
K_Nearest_Neighbors
Random_Forest
SVM_Linear_Kernel
SVM_RBF_Kernel
Gradient_Boosting
None