1.0 Importing Libraries

- Setting various of jupyter notebook for better readablity

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure, xticks
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler, StandardScaler, scale, PolynomialFeatures
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, RandomizedSearchCV,cross_validate,KFold
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV, LinearRegression, ElasticNet
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier,export_graphviz
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, BaggingRegressor,GradientBoostingRegressor
from sklearn.metrics import r2_score,precision_score, recall_score, mean_squared_error, mean_absolute_error
from sklearn.feature_selection import RFE
from sklearn import preprocessing, utils

import xgboost as xgb
from xgboost import XGBRegressor
import lightgbm as lgbm
from lightgbm import LGBMRegressor

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

from IPython.display import Image
from six import StringIO
from io import StringIO
import pydotplus
import graphviz


pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

2.0 Importing Dataframe and evaluating basic properties

In [2]:
```
base=pd.read_csv('CTGAN Generated data.csv')
base=base.rename(columns=lambda x: x.strip())
```

In [3]:
```
base.head()
```

Out[3]:

	Feed N Plus 2A content	Reactor WAIT	H2 to HC	Reactor 1 Inlet Temp	Reactor 2 Inlet Temp	Reactor 3 Inlet Temp	Reactor 4 Inlet Temp	Reactor 1 Delta T	Reactor 2 Delta T	Reactor 3 Delta T	Reactor 4 Delta T	Reactor 1 Delta P	Reactor 2 Delta P	Reactor 3 Delta P	Reactor 4 Delta P	Seperator Pressure	Seperator Temperature	Recycle gas purity	Net gas Hydrogen Purity	Coke on Spent Catalyst	Chloride Injection rate	Total Paraffins in feed	Total Naphthenes in feed	Total Aromatics in feed	Tot olefin i Fee
0	45.46	1004.48	3.28	998.41	1000.46	1006.41	1011.81	166.26	108.39	72.02	39.17	1.17	2.73	2.77	3.17	30.83	100.62	78.67	88.24	3.82	2.59	64.91	24.60	10.43	0.0
1	44.22	1004.54	3.17	999.00	999.57	1003.23	1012.77	166.37	105.33	68.00	38.45	1.17	2.80	2.78	3.22	30.32	103.02	78.67	88.24	3.56	2.64	65.89	23.58	10.32	0.2
2	44.22	1004.54	3.17	996.97	999.72	1004.46	1012.81	165.14	107.33	69.92	39.13	1.16	2.74	2.77	3.25	30.27	103.55	78.67	88.24	3.56	2.58	65.89	23.58	10.32	0.2
3	45.49	1002.61	3.46	998.34	997.32	1001.66	1011.66	172.18	106.72	70.46	40.10	1.25	2.72	2.76	3.18	32.56	103.20	77.88	88.40	3.82	2.60	63.35	27.07	9.21	0.3
4	45.46	1004.34	3.31	998.17	999.71	1004.88	1013.84	165.81	107.55	69.72	39.09	1.16	2.77	2.76	3.09	30.39	103.05	78.67	88.24	3.82	2.60	64.91	24.60	10.43	0.0

In [4]:
```
base.shape
```

Out[4]:
```
(5962, 30)
```

3.0 Data Cleaning

In [5]:
```
base.describe()
```

Out[5]:

	Feed N Plus 2A content	Reactor WAIT	H2 to HC	Reactor 1 Inlet Temp	Reactor 2 Inlet Temp	Reactor 3 Inlet Temp	Reactor 4 Inlet Temp	Reactor 1 Delta T	Reactor 2 Delta T	Reactor 3 Delta T	Reactor 4 Delta T	Reactor 1 Delta P	Reactor 2 Delta P	Reactor 3 Delta P	Reactor 4 Delta P	Seperator Pressure	Seperator Temperature	Recycle pu
count	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000000	5962.000
mean	47.972068	982.563648	3.814648	975.295392	982.172801	983.606872	986.789906	161.030943	106.133452	71.095792	41.762420	0.992466	2.587736	2.640976	2.805659	33.360704	105.697828	81.232
std	2.046191	14.284562	0.335320	13.532721	15.434145	13.933120	15.674754	7.303361	6.186524	3.766175	4.140208	0.341575	0.091843	0.132266	0.240884	1.310147	3.924914	9.579
min	40.380000	963.040000	3.100000	945.740000	951.400000	960.060000	964.350000	134.380000	86.240000	52.290000	22.690000	0.450000	2.330000	2.150000	2.130000	28.650000	92.960000	0.860
25%	47.000000	968.530000	3.560000	964.770000	966.470000	970.472500	971.550000	156.232500	101.560000	69.530000	39.670000	0.650000	2.520000	2.530000	2.640000	32.510000	103.290000	80.880
50%	48.150000	980.890000	3.770000	970.845000	981.850000	981.610000	986.385000	161.160000	106.120000	71.350000	42.580000	1.100000	2.570000	2.650000	2.840000	33.490000	105.805000	82.550
75%	49.380000	991.927500	4.030000	984.447500	994.270000	995.450000	996.162500	166.407500	110.450000	73.137500	44.670000	1.310000	2.670000	2.750000	2.980000	34.300000	108.307500	83.450
max	59.890000	1009.490000	5.660000	1004.350000	1015.790000	1009.320000	1015.620000	176.950000	128.670000	83.260000	51.740000	1.880000	2.820000	2.920000	3.280000	36.500000	119.300000	86.820

In [6]:
```
### Removing outliers using Q1-1.5IQR & Q3+1.5IQR
def mod_outlier(df):

        df1 = df.copy()
        df1 = df1._get_numeric_data()

        df2 = df.copy()

        q1 = df1.quantile(0.1)
        q3 = df1.quantile(0.9)

        iqr = q3 - q1

        lower_bound = q1 -(1.5 * iqr)
        upper_bound = q3 +(1.5 * iqr)

        for col in df.columns:
            for i in range(0,len(df1[col])):
                if df1[col][i] < lower_bound[col]:
                    df1[col][i] = np.nan

                if df1[col][i] > upper_bound[col]:
                    df1[col][i] = np.nan


            for col in df.columns:
                df2[col] = df1[col]

        return(df2)


df_outlier_removed = mod_outlier(base) #Passing the base dataframe through the mod_outlier function
print(df_outlier_removed.isna().sum())
df_outlier_removed=df_outlier_removed.dropna() ## Dropping blank rows
```

```
Feed N Plus 2A content       18
Reactor WAIT                  0
H2 to HC                     13
Reactor 1 Inlet Temp          0
Reactor 2 Inlet Temp          0
Reactor 3 Inlet Temp          0
Reactor 4 Inlet Temp          0
Reactor 1 Delta T             0
Reactor 2 Delta T             0
Reactor 3 Delta T             2
Reactor 4 Delta T             0
Reactor 1 Delta P             0
Reactor 2 Delta P             0
Reactor 3 Delta P             0
Reactor 4 Delta P             0
Seperator Pressure            0
Seperator Temperature         0
Recycle gas purity           80
Net gas Hydrogen Purity       0
Coke on Spent Catalyst        0
Chloride Injection rate       2
Total Paraffins in feed      21
Total Naphthenes in feed     18
Total Aromatics in feed       0
Total olefins in Feed       142
Reactor LHSV                 22
Feed IBP                      0
50% IBP                      69
WABT                          0
Plant C5PlusYield             0
dtype: int64
```

4.0 Identifying Features/Independent variables for building models

4.1 Feature Selection Based on RFE using DecisionTreeRegressor and LinearRegression

In [7]:
```
# RFE using DecisionTree Regressor

def rfe_DecissionTreeRegressor(df, target_col, k):
    # Create a copy of the original dataframe
    df_new = df.copy()

    # Get the target column values
    y = df_new[target_col]

    # Initialize a decision tree regressor model
    model = DecisionTreeRegressor()

    # Initialize RFE with the decision tree regressor model and the number of desired features (k)
    rfe = RFE(estimator=model, n_features_to_select=k)

    # Perform recursive feature elimination
    rfe.fit(df_new.drop(target_col, axis=1), y)

    # Get the selected feature indices
    selected_indices = rfe.get_support(indices=True)

    # Get the names of the selected features
    important_vars = df_new.drop(target_col, axis=1).columns[selected_indices].tolist()

    return important_vars

# Dataframe is named 'df_outlier_removed' and the target column is named 'Plant C5PlusYield'
k = 15  # Select the top 15 most important variables
important_vars_dectree = rfe_DecissionTreeRegressor(df_outlier_removed, 'Plant C5PlusYield', k)

# Print the list of important variables
print(important_vars_dectree)
```

```
['Feed N Plus 2A content', 'Reactor WAIT', 'H2 to HC', 'Reactor 1 Inlet Temp', 'Reactor 1 Delta T', 'Reactor 3 Delta T', 'Seperator Pressure', 'Net gas Hydrogen Purity', 'Coke on Spent Catalys
t', 'Chloride Injection rate', 'Total Aromatics in feed', 'Total olefins in Feed', 'Reactor LHSV', '50% IBP', 'WABT']
```

In [8]:
```
# Recrussive Feature Engineering using Linear Regression

def rfe_LinearRegression(df, target_col, k):
    # Create a copy of the original dataframe
    df_new = df.copy()

    # Get the target column values
    y = df_new[target_col]

    # Initialize a linear regression model
    model = LinearRegression()

    # Initialize RFE with the linear regression model and the number of desired features (k)
    rfe = RFE(estimator=model, n_features_to_select=k)

    # Perform recursive feature elimination
    rfe.fit(df_new.drop(target_col, axis=1), y)

    # Get the selected feature indices
    selected_indices = rfe.get_support(indices=True)

    # Get the names of the selected features
    important_vars = df_new.drop(target_col, axis=1).columns[selected_indices].tolist()

    return important_vars

# Dataframe is named 'df_outlier_removed' and the target column is named 'Plant C5PlusYield'
k = 15  # Select the top 15 most important variables
important_vars_lr = rfe_LinearRegression(df_outlier_removed, 'Plant C5PlusYield', k)

# Print the list of important variables
print(important_vars_lr)
```

```
['Feed N Plus 2A content', 'Reactor WAIT', 'H2 to HC', 'Reactor 1 Delta P', 'Reactor 2 Delta P', 'Reactor 3 Delta P', 'Reactor 4 Delta P', 'Seperator Pressure', 'Coke on Spent Catalyst', 'Chlor
ide Injection rate', 'Total Paraffins in feed', 'Total Naphthenes in feed', 'Total Aromatics in feed', 'Total olefins in Feed', 'Reactor LHSV']
```

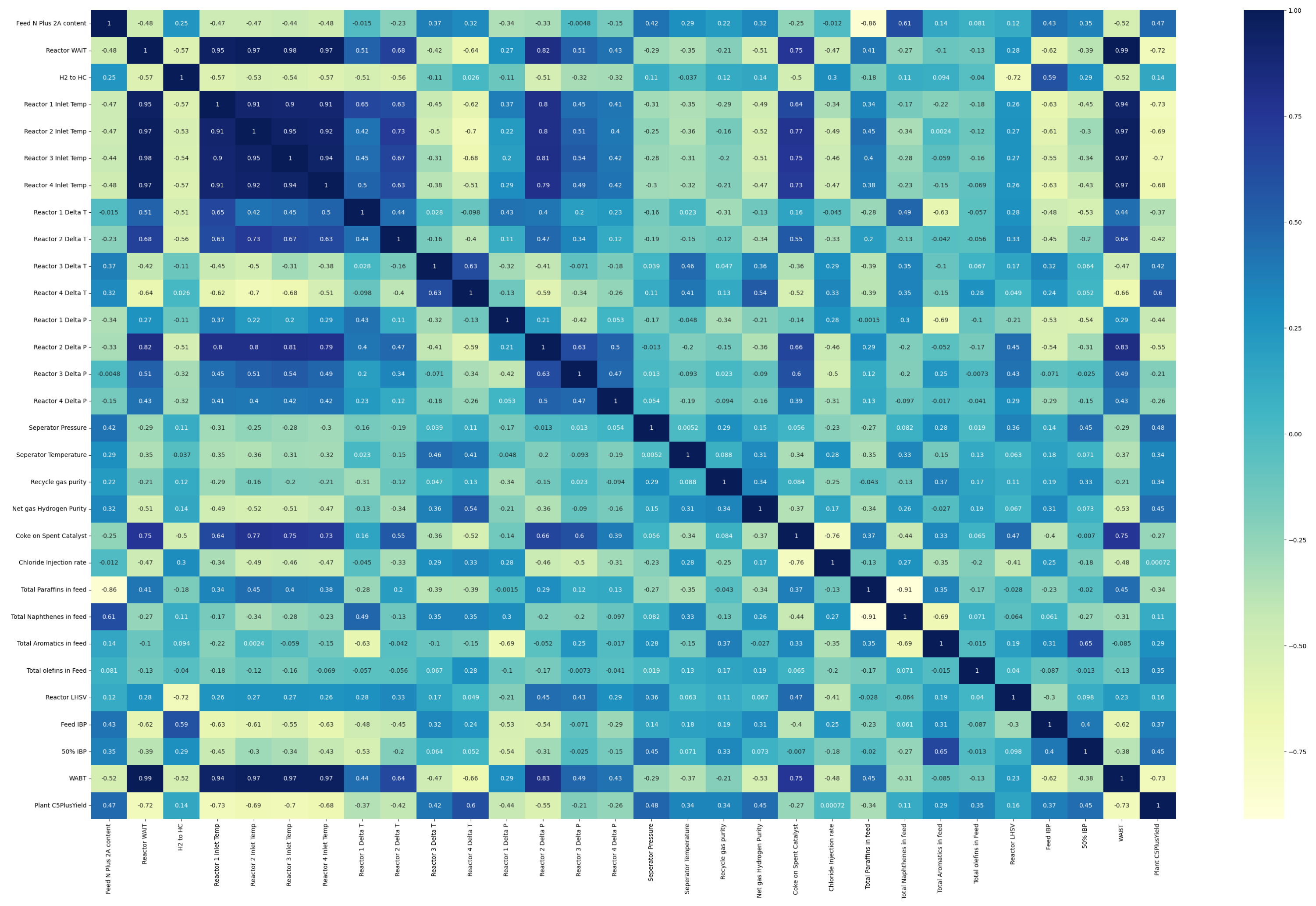
In [9]:
```
max_length = max(len(important_vars_lr), len(important_vars_dectree))

common_vars = list(set(important_vars_lr) & set(important_vars_dectree))
common_vars += [None] * (max_length - len(common_vars))

lr_only_vars = list(set(important_vars_lr) - set(important_vars_dectree))
lr_only_vars += [None] * (max_length - len(lr_only_vars))

dectree_only_vars = list(set(important_vars_dectree) - set(important_vars_lr))
dectree_only_vars += [None] * (max_length - len(dectree_only_vars))

df_rfe = pd.DataFrame({'Common': common_vars, 'Linear Regression RFE': lr_only_vars, 'Decision Tree RFE': dectree_only_vars})

df_rfe
```

Out[9]:

	Common	Linear Regression RFE	Decision Tree RFE
0	Seperator Pressure	Total Paraffins in feed	WABT
1	H2 to HC	Reactor 3 Delta P	Reactor 1 Delta T
2	Reactor LHSV	Reactor 4 Delta P	Reactor 3 Delta T
3	Reactor WAIT	Reactor 1 Delta P	Net gas Hydrogen Purity
4	Feed N Plus 2A content	Reactor 2 Delta P	Reactor 1 Inlet Temp
5	Coke on Spent Catalyst	Total Naphthenes in feed	50% IBP
6	Chloride Injection rate	None	None
7	Total Aromatics in feed	None	None
8	Total olefins in Feed	None	None
9	None	None	None
10	None	None	None
11	None	None	None
12	None	None	None
13	None	None	None
14	None	None	None

4.2 Feature selection based on Subject knowledge and VIF

In [10]:
```
## Building a Heat map first and identifying highly correlated variable
figure(figsize=(40, 24))
dataplot = sns.heatmap(df_outlier_removed.corr(), cmap="YlGnBu", annot=True)

# displaying heatmap
plt.show()
```



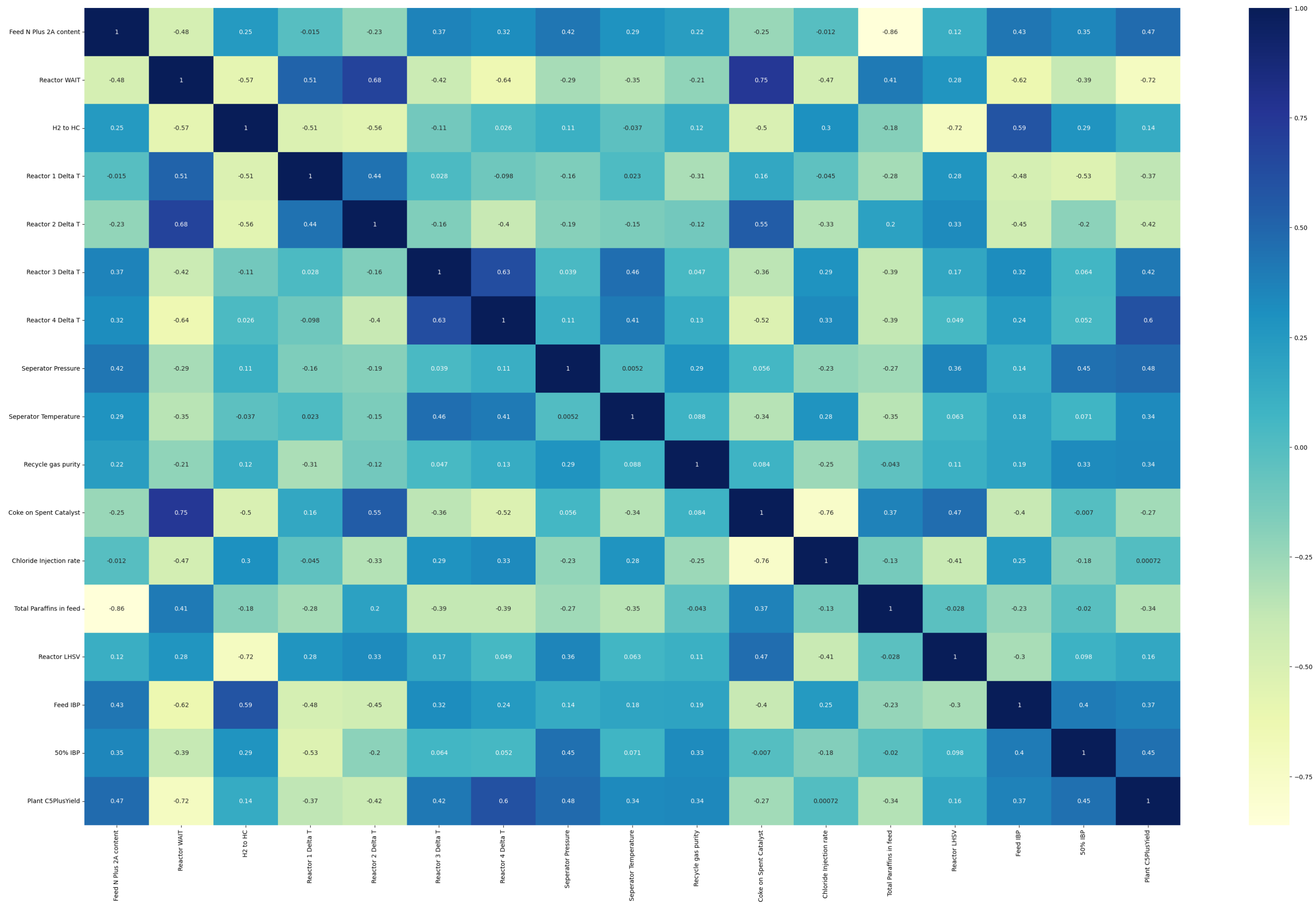
In [11]:
```
cols_to_drop=['Reactor 1 Inlet Temp','Reactor 2 Inlet Temp', 'Reactor 3 Inlet Temp','Reactor 4 Inlet Temp',
             'Reactor 1 Delta P','Reactor 2 Delta P', 'Reactor 3 Delta P', 'Reactor 4 Delta P','Net gas Hydrogen Purity',
             'Total Naphthenes in feed', 'Total Aromatics in feed','Total olefins in Feed','WABT']
len(cols_to_drop)
```

Out[11]:
```
13
```

In [12]:
```
df_trimmed=df_outlier_removed.drop(cols_to_drop, axis=1)
number_of_columns_dropped = df_outlier_removed.shape[1] - df_trimmed.shape[1]
number_of_columns_dropped
```

Out[12]:
```
13
```

```
In [13]: figure(figsize=(40, 24))
         dataplot = sns.heatmap(df_trimmed.corr(), cmap="YlGnBu", annot=True)

         # displaying heatmap
         plt.show()
```



5.0 Creating a Linear regression model and checking VIF

```
In [14]: # Splitting the dataset into X and y
         X = df_trimmed.drop(['Plant C5PlusYield'], axis = 1)
         y = df_trimmed[['Plant C5PlusYield']]

         # Splitting data into Train and Test (80-20)
         X_train, X_test,y_train, y_test = train_test_split(X,y ,random_state=104, test_size=0.2, shuffle=True)

         # Scaling the X data using StandardScaler
         ss = StandardScaler()
         X_train = pd.DataFrame(ss.fit_transform(X_train),columns = X_train.columns)
         X_test = pd.DataFrame(ss.fit_transform(X_test),columns = X_test.columns)

         #Identifying total number of independent variables
         print(f'Total number of independent variables are :  {X_train.shape[1]}')

         Total number of independent variables are :  16
```

```
In [15]: ## Applying RFE on df_trimmed to get a starting point for the model
         k = 10  # Select the top 15 most important variables
         important_vars_lr = rfe_LinearRegression(df_trimmed, 'Plant C5PlusYield', k)

         # Print the list of important variables
         print(important_vars_lr)

         ['Reactor WAIT', 'H2 to HC', 'Reactor 4 Delta T', 'Seperator Pressure', 'Seperator Temperature', 'Coke on Spent Catalyst', 'Chloride Injection rate', 'Reactor LHSV', 'Feed IBP', '50% IBP']
```

In [16]:
```python
# Concised model- First model using all columns
relevant_cols= X_train.columns
#Model 1 (Linear regression)
X_train_sm = sm.add_constant(X_train[relevant_cols])
lr = sm.OLS(y_train.values.reshape(-1,1), X_train_sm).fit()
#lr = sm.OLS(y_train, X_train_sm).fit()
lr.summary()
```

Out[16]:
```
OLS Regression Results

      Dep. Variable:                 y        R-squared:    0.797
             Model:                OLS   Adj. R-squared:    0.797
            Method:      Least Squares      F-statistic:    1103.
              Date:   Wed, 24 May 2023 Prob (F-statistic):   0.00
              Time:           19:43:09   Log-Likelihood:  -3547.8
  No. Observations:               4499              AIC:    7130.
      Df Residuals:               4482              BIC:    7239.
          Df Model:                 16
   Covariance Type:          nonrobust

                            coef  std err         t   P>|t|  [0.025   0.975]
                  const  84.9682    0.008  1.07e+04   0.000  84.953   84.984
 Feed N Plus 2A content   0.0071    0.025     0.285   0.775  -0.042    0.056
           Reactor WAIT  -1.2812    0.025   -51.255   0.000  -1.330   -1.232
               H2 to HC  -0.3396    0.020   -17.002   0.000  -0.379   -0.300
      Reactor 1 Delta T   0.0233    0.016     1.487   0.137  -0.007    0.054
      Reactor 2 Delta T  -0.0164    0.012    -1.391   0.164  -0.040    0.007
      Reactor 3 Delta T   0.0144    0.012     1.179   0.239  -0.010    0.038
      Reactor 4 Delta T   0.1396    0.014     9.838   0.000   0.112    0.167
     Seperator Pressure   0.1455    0.012    12.611   0.000   0.123    0.168
  Seperator Temperature   0.0933    0.010     9.606   0.000   0.074    0.112
     Recycle gas purity   0.0121    0.009     1.292   0.196  -0.006    0.031
  Coke on Spent Catalyst  0.4090    0.020    20.549   0.000   0.370    0.448
 Chloride Injection rate -0.2346    0.014   -16.753   0.000  -0.262   -0.207
 Total Paraffins in feed  0.0220    0.026     0.836   0.403  -0.030    0.074
            Reactor LHSV -0.0658    0.016    -4.022   0.000  -0.098   -0.034
                Feed IBP -0.0369    0.013    -2.855   0.004  -0.062   -0.012
                 50% IBP  0.0296    0.012     2.423   0.015   0.006    0.054

        Omnibus: 722.817     Durbin-Watson:    2.019
 Prob(Omnibus):    0.000  Jarque-Bera (JB): 8947.121
           Skew:  -0.364          Prob(JB):     0.00
       Kurtosis:   9.870         Cond. No.      10.5


Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [17]:
```python
# P values are high for some variables. Checking VIF
vif = pd.DataFrame()
vif['Features'] = X_train[relevant_cols].columns
vif['VIF'] = [variance_inflation_factor(X_train[relevant_cols].values, i) for i in range(X_train[relevant_cols].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[17]:

	Features	VIF
12	Total Paraffins in feed	10.93
0	Feed N Plus 2A content	9.89
1	Reactor WAIT	9.88
2	H2 to HC	6.31
10	Coke on Spent Catalyst	6.26
13	Reactor LHSV	4.24
3	Reactor 1 Delta T	3.88
6	Reactor 4 Delta T	3.19
11	Chloride Injection rate	3.10
14	Feed IBP	2.63
15	50% IBP	2.37
5	Reactor 3 Delta T	2.36
4	Reactor 2 Delta T	2.20
7	Seperator Pressure	2.10
8	Seperator Temperature	1.49
9	Recycle gas purity	1.40

In [18]:
```python
#Dropping a few variables
variables_to_be_dropped=['Reactor 1 Delta T','Reactor 2 Delta T','Reactor 3 Delta T','Reactor 4 Delta T',
                         'Seperator Temperature'] #dropping these as Reactor delta T have high p Value and separator temp
                                                  #is highly collinear to last reactor temperature
relevant_cols = [item for item in relevant_cols if item not in variables_to_be_dropped]
```

In [19]:
```python
# Fitting 2nd Model
X_train_sm = sm.add_constant(X_train[relevant_cols])
lr = sm.OLS(y_train.values.reshape(-1,1), X_train_sm).fit()
#lr = sm.OLS(y_train, X_train_sm).fit()
lr.summary()
```

Out[19]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.787
Model:	OLS	**Adj. R-squared:**	0.786
Method:	Least Squares	**F-statistic:**	1506.
Date:	Wed, 24 May 2023	**Prob (F-statistic):**	0.00
Time:	19:43:10	**Log-Likelihood:**	-3662.3
No. Observations:	4499	**AIC:**	7349.
Df Residuals:	4487	**BIC:**	7426.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	84.9682	0.008	1.04e+04	0.000	84.952	84.984
Feed N Plus 2A content	-0.0635	0.024	-2.607	0.009	-0.111	-0.016
Reactor WAIT	-1.4291	0.018	-78.366	0.000	-1.465	-1.393
H2 to HC	-0.4505	0.017	-25.773	0.000	-0.485	-0.416
Seperator Pressure	0.1224	0.012	10.572	0.000	0.100	0.145
Recycle gas purity	0.0354	0.009	3.750	0.000	0.017	0.054
Coke on Spent Catalyst	0.3859	0.020	19.480	0.000	0.347	0.425
Chloride Injection rate	-0.2219	0.014	-15.770	0.000	-0.249	-0.194
Total Paraffins in feed	-0.0901	0.023	-3.993	0.000	-0.134	-0.046
Reactor LHSV	-0.0587	0.017	-3.518	0.000	-0.091	-0.026
Feed IBP	-0.0190	0.012	-1.535	0.125	-0.043	0.005
50% IBP	0.0273	0.012	2.240	0.025	0.003	0.051

Omnibus:	641.059	**Durbin-Watson:**	2.019
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6649.723
Skew:	-0.320	**Prob(JB):**	0.00
Kurtosis:	8.921	**Cond. No.**	8.18

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [20]:
```python
#p value for Feed IBP is high. Lets check VIF
vif = pd.DataFrame()
vif['Features'] = X_train[relevant_cols].columns
vif['VIF'] = [variance_inflation_factor(X_train[relevant_cols].values, i) for i in range(X_train[relevant_cols].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[20]:

	Features	**VIF**
0	Feed N Plus 2A content	8.94
7	Total Paraffins in feed	7.66
5	Coke on Spent Catalyst	5.90
1	Reactor WAIT	5.00
2	H2 to HC	4.60
8	Reactor LHSV	4.19
6	Chloride Injection rate	2.98
9	Feed IBP	2.31
10	50% IBP	2.24
3	Seperator Pressure	2.02
4	Recycle gas purity	1.34

In [21]:
```python
#Dropping a few more variables
variables_to_be_dropped=['Feed IBP'] #dropping as it has high correlation with Feed composition and 50% boiling point
relevant_cols = [item for item in relevant_cols if item not in variables_to_be_dropped]
```

In [22]:
```python
# Fitting Final Model
X_train_sm = sm.add_constant(X_train[relevant_cols])
lr = sm.OLS(y_train.values.reshape(-1,1), X_train_sm).fit()
#lr = sm.OLS(y_train, X_train_sm).fit()
lr.summary()
```

Out[22]:
```
OLS Regression Results

        Dep. Variable:                y          R-squared:      0.787
                Model:              OLS     Adj. R-squared:      0.786
               Method:    Least Squares         F-statistic:     1656.
                 Date: Wed, 24 May 2023  Prob (F-statistic):      0.00
                 Time:         19:43:10     Log-Likelihood:    -3663.5
     No. Observations:             4499                 AIC:     7349.
         Df Residuals:             4488                 BIC:     7420.
             Df Model:               10
      Covariance Type:        nonrobust

                              coef  std err         t  P>|t|  [0.025   0.975]
                   const   84.9682    0.008  1.04e+04  0.000  84.952   84.984
   Feed N Plus 2A content  -0.0762    0.023    -3.324  0.001  -0.121   -0.031
             Reactor WAIT  -1.4212    0.017   -81.271  0.000  -1.455   -1.387
                 H2 to HC  -0.4578    0.017   -27.205  0.000  -0.491   -0.425
       Seperator Pressure   0.1258    0.011    11.068  0.000   0.104    0.148
       Recycle gas purity   0.0358    0.009     3.795  0.000   0.017    0.054
   Coke on Spent Catalyst   0.3835    0.020    19.417  0.000   0.345    0.422
  Chloride Injection rate  -0.2235    0.014   -15.921  0.000  -0.251   -0.196
   Total Paraffins in feed -0.0997    0.022    -4.594  0.000  -0.142   -0.057
              Reactor LHSV -0.0600    0.017    -3.596  0.000  -0.093   -0.027
                  50% IBP   0.0273    0.012     2.241  0.025   0.003    0.051

         Omnibus:  629.745     Durbin-Watson:     2.019
   Prob(Omnibus):    0.000  Jarque-Bera (JB):  6514.378
            Skew:   -0.303          Prob(JB):      0.00
        Kurtosis:    8.864          Cond. No.      7.42

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

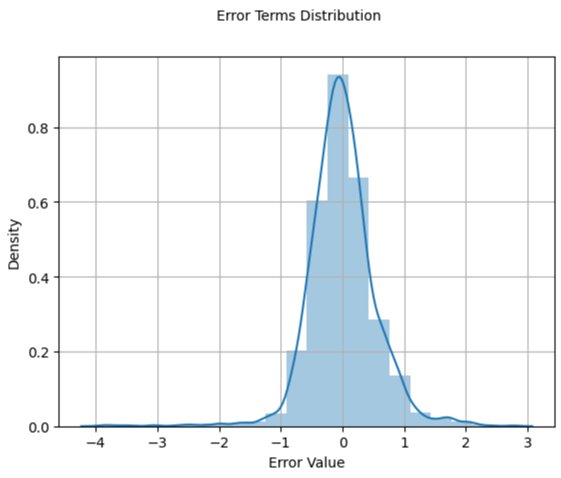
In [23]:
```python
# defining a function to Test the error of predicted vs actual value
def calculate_prediction_error(X_train, y_train, lr_model):
    # Predict the target variable using the trained model
    y_train_pred = lr_model.predict(X_train)

    # Create a DataFrame for predicted values
    y_train_pred = pd.DataFrame({'Predicted': y_train_pred})

    # Reset the index of y_train and drop the current index
    y_train_act = y_train.reset_index(drop=True)

    # Concatenate actual and predicted values
    y_train_error = pd.concat([y_train_act, y_train_pred], axis=1, join="inner")

    # Calculate the error
    y_train_error['Error'] = y_train_error['Plant C5PlusYield'] - y_train_error['Predicted']

    # Calculate the R2 score
    r2_score_value = r2_score(y_train, y_train_pred)

    # Plot the distribution of error terms
    fig = plt.figure()
    sns.distplot(y_train_error['Error'], bins=20)
    fig.suptitle('Error Terms Distribution', fontsize=10)
    plt.xlabel('Error Value', fontsize=10)
    plt.grid(True)

    return y_train_error, r2_score_value
```

```
In [24]: # the above defined function on X_train_sm and y_train as the DataFrames, and lr is the trained Linear Regression model
         y_train_error, r2_score_value = calculate_prediction_error(X_train_sm, y_train, lr)
         print("R2 Score:", r2_score_value)
         y_train_error.head()
```

```
R2 Score: 0.7867498508303788
```

Out[24]:

	Plant C5PlusYield	Predicted	Error
0	86.67	86.530114	0.139886
1	82.25	82.833754	-0.583754
2	82.98	83.318286	-0.338286
3	86.97	86.296869	0.673131
4	83.95	83.033286	0.916714



Checkpoint

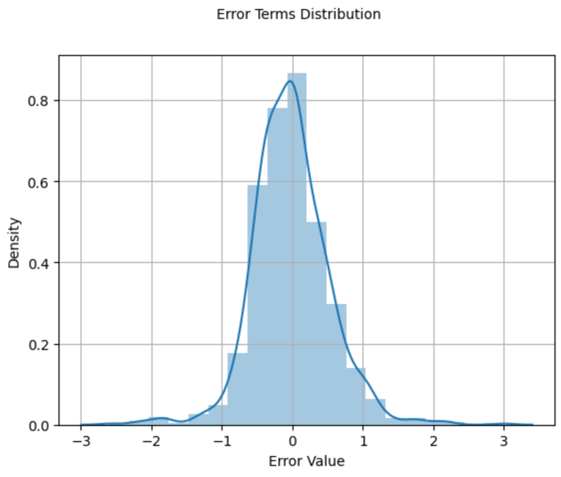
Error terms are normally distributed.

```
In [25]: ## Fitting model on Test data and analysisng
         X_test_final=X_test[relevant_cols]
         X_test_final.head()
         X_test_sm=sm.add_constant(X_test_final)
         X_test_sm.head()
         # using X_test_sm and y_test as the DataFrames, and lr is the trained Linear Regression model
         y_test_error, r2_score_value = calculate_prediction_error(X_test_sm, y_test, lr)
         print("R2 Score:", r2_score_value)
         y_test_error.head()
```

```
R2 Score: 0.7870789456093632
```

Out[25]:

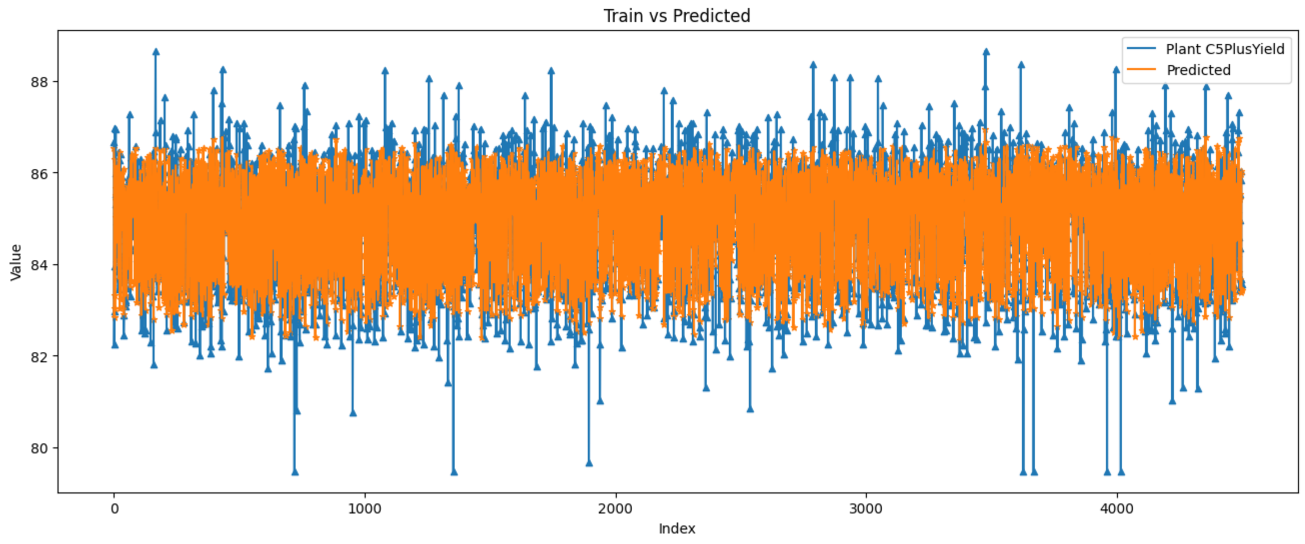
	Plant C5PlusYield	Predicted	Error
0	85.72	84.497737	1.222263
1	86.27	85.877906	0.392094
2	85.94	86.052058	-0.112058
3	85.07	85.311617	-0.241617
4	84.00	83.106448	0.893552

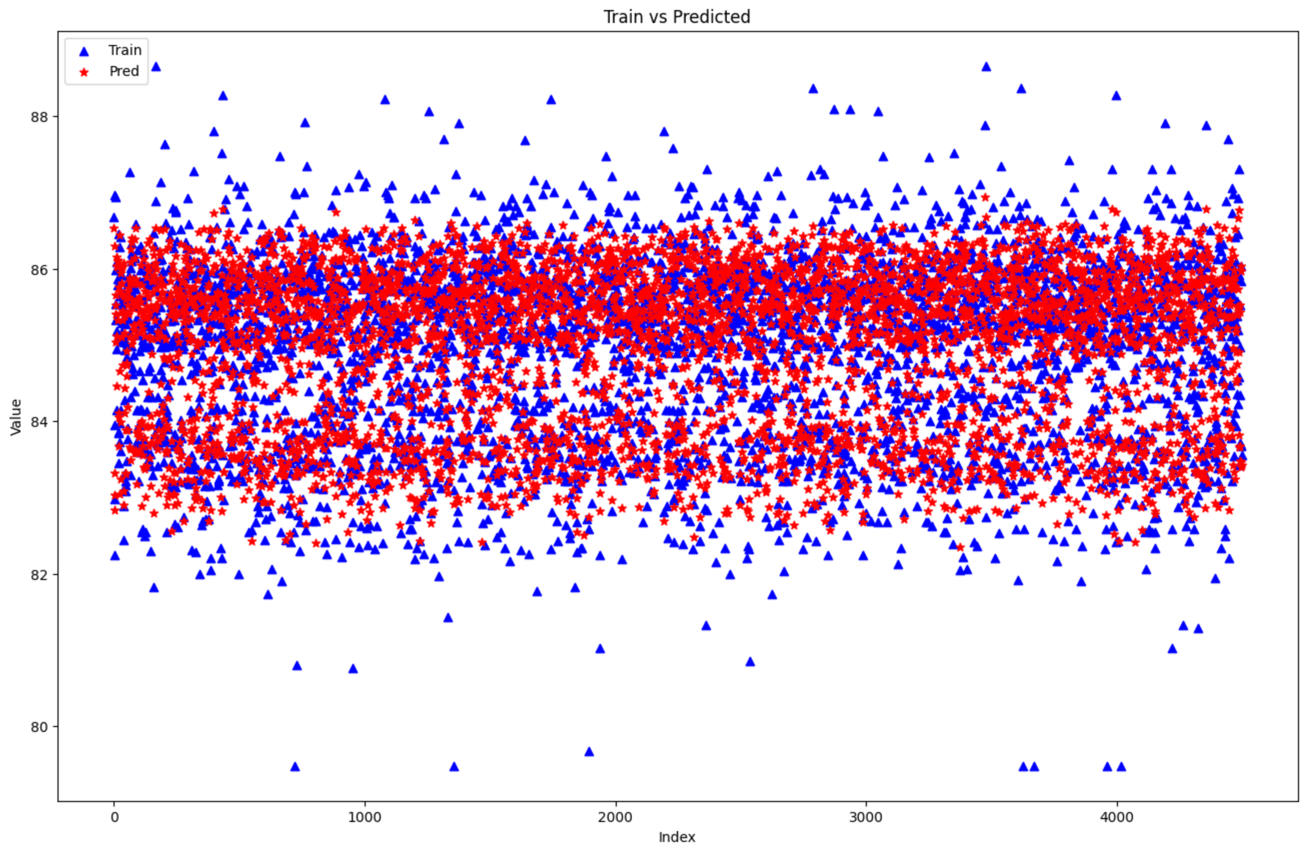


Visualization

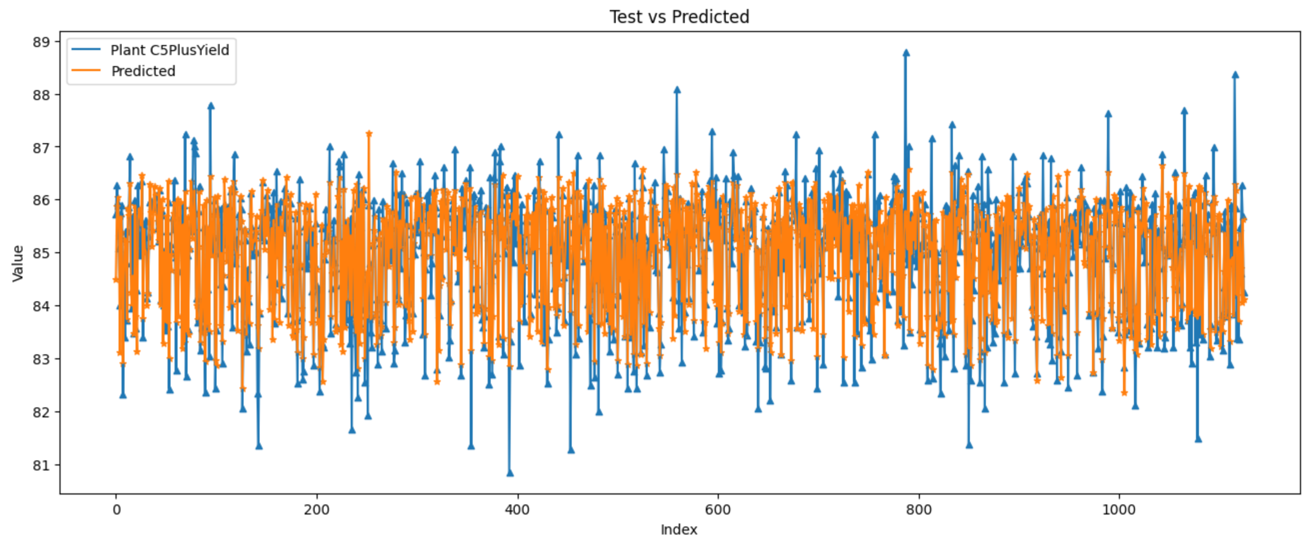
In [26]:
```python
## Defining a function to plot the error analysis

def plot_error_analysis(df_error,title):


    # Plot 1
    df_plot = df_error.drop('Error', axis=1)
    df_plot['x1'] = df_plot.index
    ax1 = df_plot.plot(x='x1', figsize=(16, 6))
    ax2 = df_plot.plot.scatter(x='x1', y=['Plant C5PlusYield'], marker='^', ax=ax1)
    ax3 = df_plot.plot.scatter(x='x1', y=['Predicted'], marker='*', ax=ax1, color='#ff7f0e')
    plt.title(title)
    plt.xlabel('Index')
    plt.ylabel('Value')

    # Plot 2
    plt.figure(figsize=(16, 10))
    df_error['x1'] = df_error.index
    plt.scatter(df_error['x1'], df_error['Plant C5PlusYield'], c='b', marker='^', label='Train')
    plt.scatter(df_error['x1'], df_error['Predicted'], c='r', marker='*', label='Pred')
    plt.title(title)
    plt.xlabel('Index')
    plt.ylabel('Value')
    plt.legend(loc='upper left')
    plt.show()  # Add this line to show the second plot
```

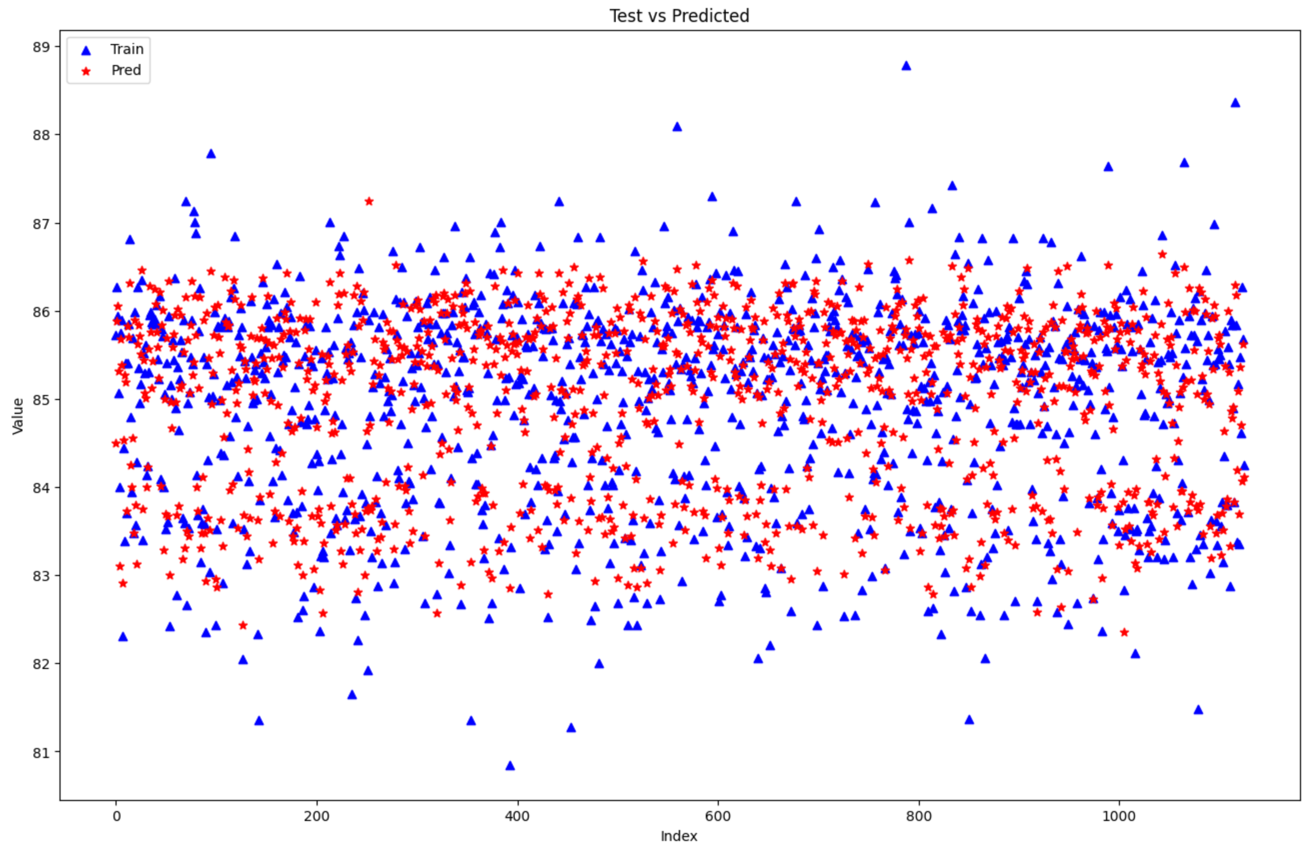
In [27]:
```python
# Plotting for Actual vs Predicted for Train data
plot_error_analysis(y_train_error,'Train vs Predicted')
```





```
In [28]: # Plotting for Actual vs Predicted for Test data
         plot_error_analysis(y_test_error, 'Test vs Predicted')
```





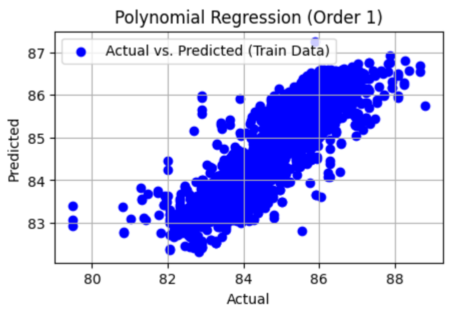
6.0 Polynomial Regression Model

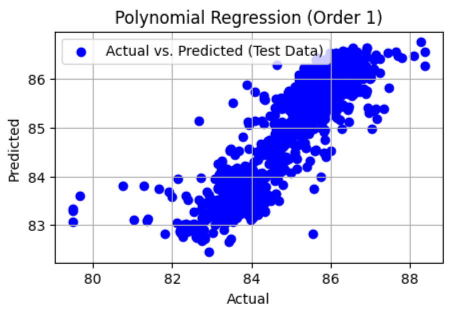
```
In [29]: X_relevant=X_train_sm.drop('const',axis=1)
         x=list(X_relevant.columns)
         y=('Plant C5PlusYield')
         x.append(y)
         df_final=df_trimmed[x]
         df_final.head()
```

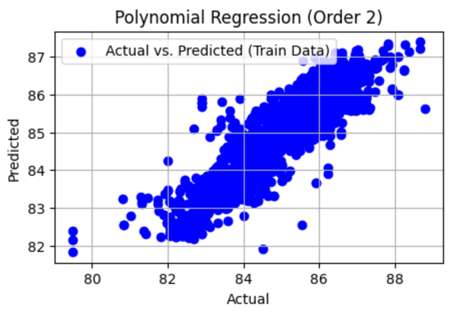
Out[29]:

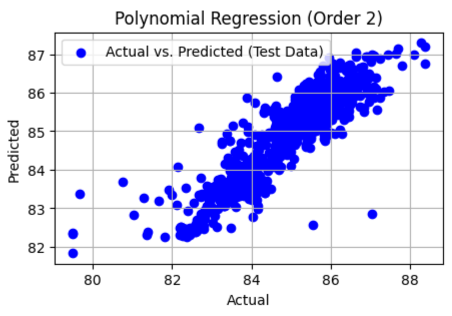
	Feed N Plus 2A content	Reactor WAIT	H2 to HC	Seperator Pressure	Recycle gas purity	Coke on Spent Catalyst	Chloride Injection rate	Total Paraffins in feed	Reactor LHSV	50% IBP	Plant C5PlusYield
0	45.46	1004.48	3.28	30.83	78.67	3.82	2.59	64.91	1.62	250.57	83.24
1	44.22	1004.54	3.17	30.32	78.67	3.56	2.64	65.89	1.62	247.78	83.20
2	44.22	1004.54	3.17	30.27	78.67	3.56	2.58	65.89	1.62	247.78	83.20
3	45.49	1002.61	3.46	32.56	77.88	3.82	2.60	63.35	1.62	249.98	83.46
4	45.46	1004.34	3.31	30.39	78.67	3.82	2.60	64.91	1.62	250.57	83.33

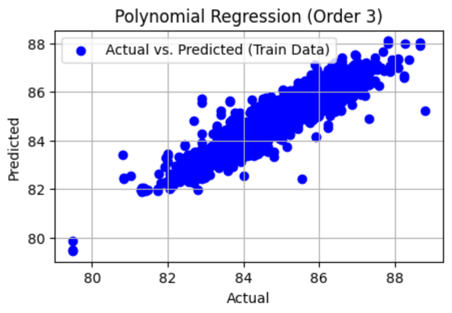
```
In [30]: # Create a list of dictionaries to store the results
         results = []

         # Test polynomial regression models from 1st to 4th order
         for order in range(1, 5):
             X_train, X_test, y_train, y_test = train_test_split(df_final.drop('Plant C5PlusYield', axis=1), df_final['Plant C5PlusYield'], test_size=0.2, random_state=42)

             # Create polynomial features
             poly_features = PolynomialFeatures(degree=order)
             X_train_poly = poly_features.fit_transform(X_train)
             X_test_poly = poly_features.transform(X_test)

             # Fit the polynomial regression model
             model = LinearRegression()
             model.fit(X_train_poly, y_train)

             # Predict the target variable for train and test data
             y_train_pred = model.predict(X_train_poly)
             y_test_pred = model.predict(X_test_poly)

             # Calculate mean squared error
             mse = mean_squared_error(y_test, y_test_pred)

             # Calculate R-squared score for train and test data
             r2_train = r2_score(y_train, y_train_pred)
             r2_test = r2_score(y_test, y_test_pred)

             # Plot the actual vs. predicted values for train data
             plt.figure(figsize=(5, 3))
             plt.scatter(y_train, y_train_pred, color='blue', label='Actual vs. Predicted (Train Data)')
             plt.title(f'Polynomial Regression (Order {order})')
             plt.xlabel('Actual')
             plt.ylabel('Predicted')
             plt.legend()
             plt.grid(True)
             plt.show()

             # Plot the actual vs. predicted values for test data
             plt.figure(figsize=(5, 3))
             plt.scatter(y_test, y_test_pred, color='blue', label='Actual vs. Predicted (Test Data)')
             plt.title(f'Polynomial Regression (Order {order})')
             plt.xlabel('Actual')
             plt.ylabel('Predicted')
             plt.legend()
             plt.grid(True)
             plt.show()

             # Append the results to the list
             results.append({'Order': order, 'MSE': mse, 'R2 (Train)': r2_train, 'R2 (Test)': r2_test})

         # Convert the list of dictionaries into a DataFrame
         poly_r2_results = pd.DataFrame(results)

         # Print the DataFrame
         print(poly_r2_results)
```

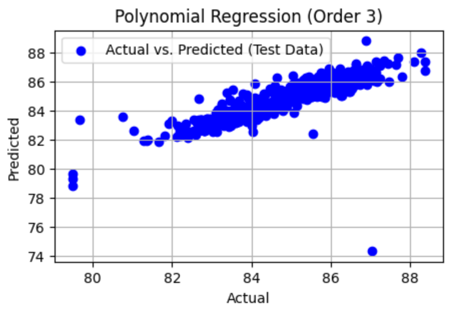


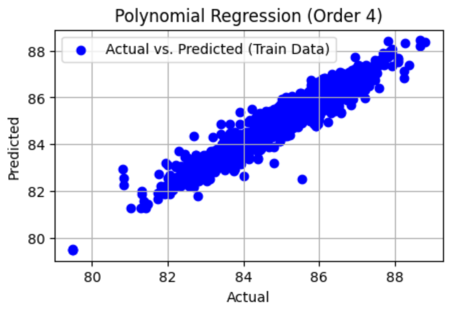


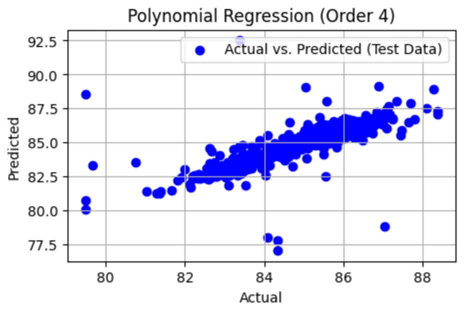










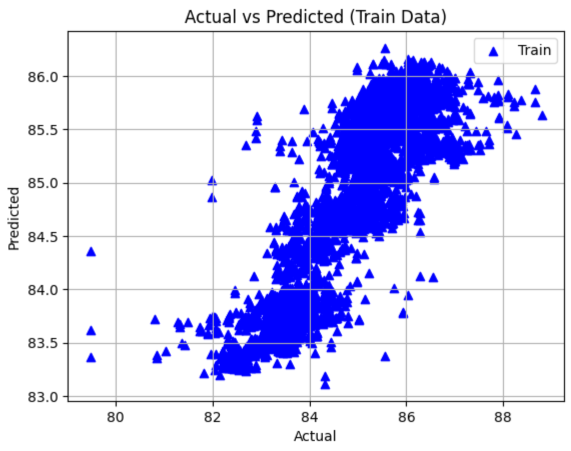


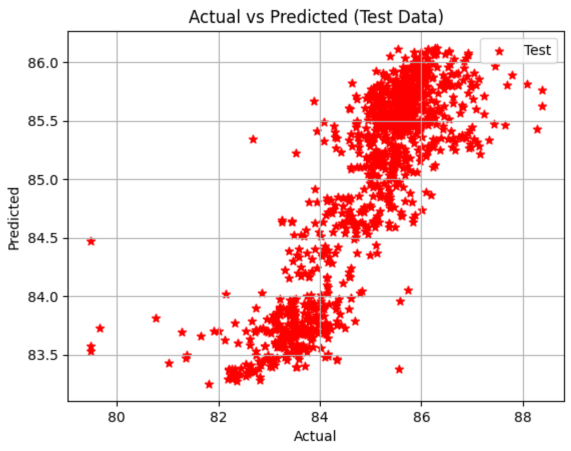
```
   Order       MSE  R2 (Train)  R2 (Test)
0      1  0.358073    0.793415   0.763524
1      2  0.278951    0.844922   0.815778
2      3  0.342419    0.898153   0.773862
3      4  0.523736    0.940705   0.654118
```

7.0 Ridge Regression, Lasso and Elastic Net

In [31]:
```
## Defining a function Ridge, Lasso and Elastic net
def grid_search_regression(model_name, param_grid):
    df_temp = df_final.copy()
    X = df_temp.drop('Plant C5PlusYield', axis=1)
    y = df_temp['Plant C5PlusYield']

    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Create the regressor object based on the model name
    if model_name == "Lasso":
        model = Lasso()
    elif model_name == "Ridge":
        model = Ridge()
    elif model_name == "ElasticNet":
        model = ElasticNet()
    else:
        raise ValueError("Invalid model name. Supported models: Lasso, Ridge, ElasticNet")

    # Perform grid search
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='r2', cv=10)
    grid_search.fit(X_train, y_train)



    # Train the Ridge regression model with the best hyperparameters
    best_temp = grid_search.best_estimator_
    best_temp.fit(X_train, y_train)

    # Perform cross-validation to evaluate the model
    cross_val_scores = cross_val_score(best_temp, X_scaled, y, cv=10, scoring='r2')

    # Predict on train and test data
    y_train_pred = best_temp.predict(X_train)
    y_test_pred = best_temp.predict(X_test)

    # Calculate R2 scores for train and test data
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    # Plot actual vs predicted for train data
    plt.scatter(y_train, y_train_pred, c='b', marker='^', label='Train')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title("Actual vs Predicted (Train Data)")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Plot actual vs predicted for test data
    plt.scatter(y_test, y_test_pred, c='r', marker='*', label='Test')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title("Actual vs Predicted (Test Data)")
    plt.legend()
    plt.grid(True)
    plt.show()

    print("Best parameters: ", grid_search.best_params_)
    print("Best R-squared score: %.3f" % grid_search.best_score_)
    print("Train R2 Score: ", train_r2)
    print("Test R2 Score: ", test_r2)
    print("Cross-validation R2 Scores: ", cross_val_scores)
    print("Average Cross-validation R2 Score: ", np.mean(cross_val_scores))
    print('-'*100)
```

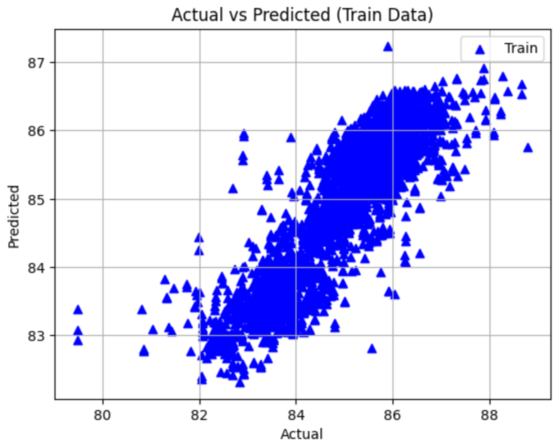
In [32]:
```
lasso_param_grid = {'alpha': [0.1, 1.0, 10.0]}
grid_search_regression("Lasso", lasso_param_grid)
```

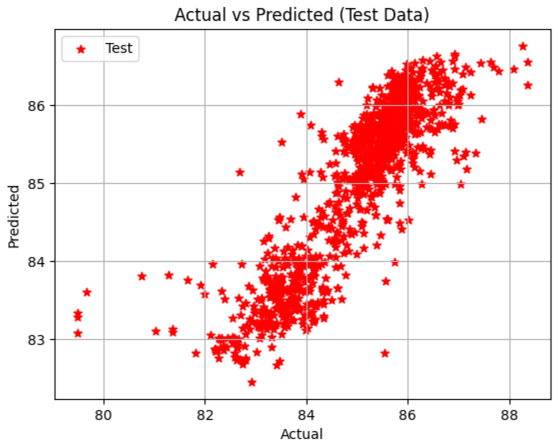




```
Best parameters:  {'alpha': 0.1}
Best R-squared score: 0.713
Train R2 Score:  0.7151533014265671
Test R2 Score:  0.6987587245766793
Cross-validation R2 Scores:  [-0.1831571   0.46838543  0.68551024  0.70536005  0.64167855  0.49062216
  0.47784368  0.49200273  0.0509346  -0.36409407]
Average Cross-validation R2 Score:  0.3465086251595856
----------------------------------------------------------------------------------------------------
```

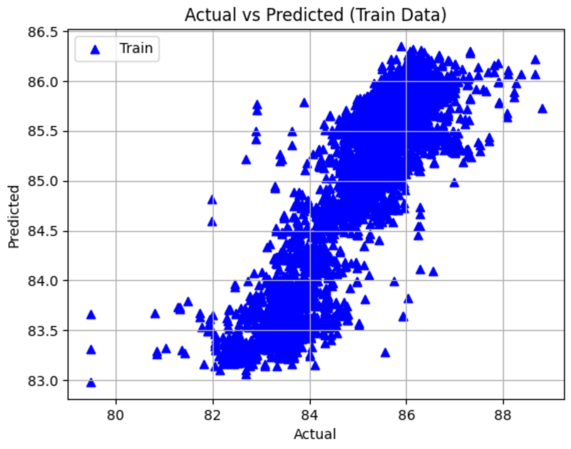
```
In [33]: ridge_param_grid = {'alpha': [0.1, 1.0, 10.0]}
         grid_search_regression("Ridge", ridge_param_grid)
```

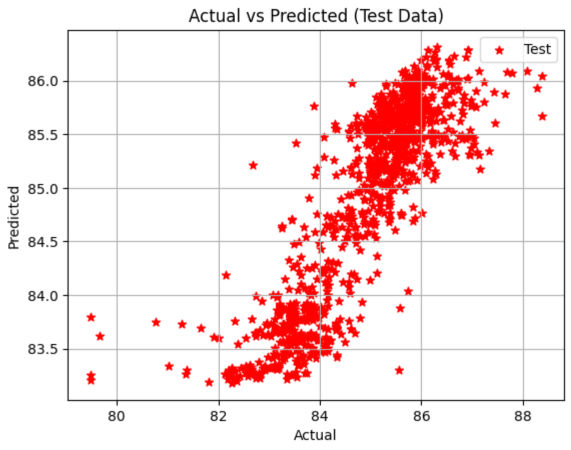




```
Best parameters:  {'alpha': 1.0}
Best R-squared score: 0.791
Train R2 Score:  0.7934140759196404
Test R2 Score:  0.7635365808475798
Cross-validation R2 Scores:  [0.36495254 0.53653727 0.74918257 0.77914956 0.71381132 0.64491639
 0.6168163  0.63089484 0.1917863  0.25169991]
Average Cross-validation R2 Score:  0.5479747000196314
----------------------------------------------------------------------------------------------------
```

```
In [34]: elasticnet_param_grid = {'alpha': [0.1, 1.0, 10.0], 'l1_ratio': [0.25, 0.5, 0.75]}
         grid_search_regression("ElasticNet", elasticnet_param_grid)
```





```
Best parameters:  {'alpha': 0.1, 'l1_ratio': 0.25}
Best R-squared score: 0.748
Train R2 Score:  0.7503607356611781
Test R2 Score:  0.7291652338958736
Cross-validation R2 Scores:  [ 0.02818046  0.49564942  0.73235725  0.73181379  0.6788015   0.57199081
  0.55591487  0.56718647  0.0965154  -0.13714748]
Average Cross-validation R2 Score:  0.43212624882293527
----------------------------------------------------------------------------------------------------
```

8.0 Bagging and Boosting

8.1 XGBoost, LGBM and Bagging- Without Cross Validation

```
In [35]: def XRegressor_TrainTest(regressor):
             # Loading Data
             df_temp = df_final.copy()
             X = df_temp.drop('Plant C5PlusYield', axis=1)
             y = df_temp['Plant C5PlusYield']

             # Standardize the features
             scaler = StandardScaler()
             X_scaled = scaler.fit_transform(X)

             # Split the data into train and test sets
             X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

             model = regressor()

             # Fit the BaggingRegressor model on the training set
             model.fit(X_train, y_train)

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

             model_name=str(regressor.__name__).upper()

             # Calculate the R-squared score on the test set
             test_r2 = r2_score(y_test, y_pred)*100
             print(f"R-squared score on unseen data (test set) using {model_name}: %.3f" % test_r2)
             print('-'*100)
```

```
In [36]: XRegressor_TrainTest(BaggingRegressor)
         XRegressor_TrainTest(XGBRegressor)
         XRegressor_TrainTest(LGBMRegressor)
```

```
R-squared score on unseen data (test set) using BAGGINGREGRESSOR: 90.918
----------------------------------------------------------------------------------------------------
R-squared score on unseen data (test set) using XGBREGRESSOR: 90.351
----------------------------------------------------------------------------------------------------
R-squared score on unseen data (test set) using LGBMREGRESSOR: 88.953
----------------------------------------------------------------------------------------------------
```

8.2 XGBoost, LGBM and Bagging- Using K-Fold Cross Validation

In [37]:
```
def XRegressor_KfoldCV(regressor):
    df_temp = df_final.copy()
    X = df_temp.drop('Plant C5PlusYield', axis=1)
    y = df_temp['Plant C5PlusYield']

    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Kfolds
    num_folds = 10
    seed = 7
    kfold = KFold(n_splits=num_folds, shuffle=True, random_state=seed)

    model_name=str(regressor.__name__).upper()

    # Bagging Regressor
    model = regressor()

    results = cross_val_score(model, X_scaled, y, cv=kfold, scoring='r2')
    print(f'RESULTS FOR {model_name}')
    print("R-squared scores:", results)
    cv_r2_mean = results.mean()
    cv_r2_std = results.std()
    print("Average R-squared: %.3f" % cv_r2_mean)
    print("Standard Deviation of R-squared: %.3f" % cv_r2_std)

    # Calculating Accuracy
    cv_accuracy_mean = cv_r2_mean * 100
    cv_accuracy_std = cv_r2_std * 100
    print("Accuracy: %.3f%% (%.3f%%)" % (cv_accuracy_mean, cv_accuracy_std))
    print('-'*30)
```

In [38]:
```
XRegressor_KfoldCV(BaggingRegressor)
XRegressor_KfoldCV(XGBRegressor)
XRegressor_KfoldCV(LGBMRegressor)
```

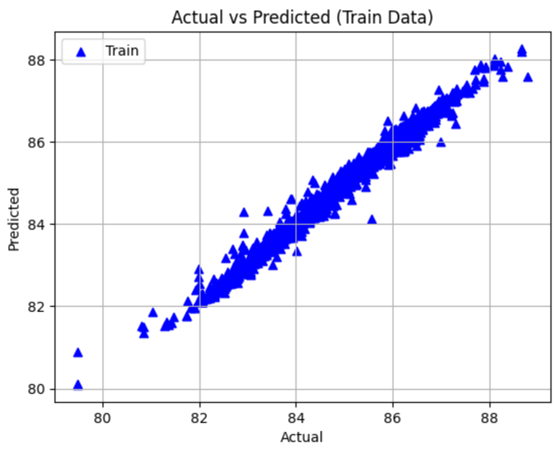
```
RESULTS FOR BAGGINGREGRESSOR
R-squared scores: [0.93815637 0.94817044 0.89732438 0.90641677 0.92236416 0.93329892
 0.89783462 0.92795107 0.92713232 0.87621344]
Average R-squared: 0.917
Standard Deviation of R-squared: 0.021
Accuracy: 91.749% (2.113%)
------------------------------
RESULTS FOR XGBREGRESSOR
R-squared scores: [0.93489351 0.93641588 0.86939604 0.90837882 0.92691926 0.93567633
 0.92265244 0.93162506 0.87669058 0.87511733]
Average R-squared: 0.912
Standard Deviation of R-squared: 0.026
Accuracy: 91.178% (2.615%)
------------------------------
RESULTS FOR LGBMREGRESSOR
R-squared scores: [0.91208591 0.91885377 0.89501824 0.90680604 0.9094335  0.9179317
 0.88553086 0.91316972 0.90531064 0.87398719]
Average R-squared: 0.904
Standard Deviation of R-squared: 0.014
Accuracy: 90.381% (1.388%)
------------------------------
```

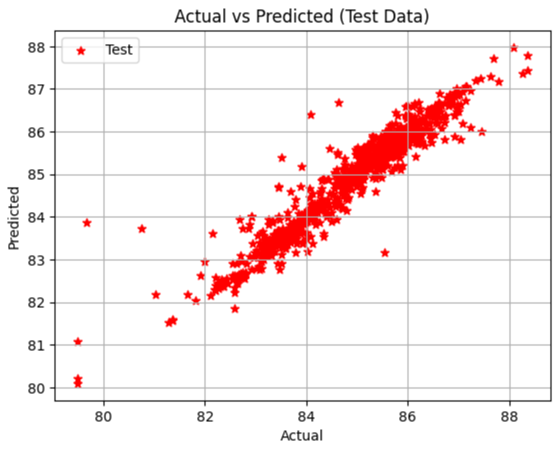
8.3 XGBoost, LGBM and Bagging- Using Grid Search Cross Validation

In [39]:
```
def grid_search_regression(model_name, param_grid):
    df_temp = df_final.copy()
    X = df_temp.drop('Plant C5PlusYield', axis=1)
    y = df_temp['Plant C5PlusYield']

    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Create the regressor object based on the model name
    if model_name == "BaggingRegressor":
        model = BaggingRegressor()
    elif model_name == "XGBRegressor":
        model = XGBRegressor()
    elif model_name == "LGBMRegressor":
        model = LGBMRegressor()
    else:
        raise ValueError("Invalid model name. Supported models: BaggingRegressor, XGBRegressor, LGBMRegressor")

    # Perform grid search
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='r2', cv=10)
    grid_search.fit(X_train, y_train)

    # Print the best parameters and score
    print("Best parameters: ", grid_search.best_params_)
    print("Best R-squared score: %.3f" % grid_search.best_score_)

    # Predict on the test set using the best model
    best_model = grid_search.best_estimator_
    y_train_pred = best_model.predict(X_train)
    y_test_pred = best_model.predict(X_test)

    # Calculate the R-squared score on the test set
    test_r2 = r2_score(y_test, y_test_pred)
    print("R-squared score on unseen data (test set): %.3f" % test_r2)

    # Plot actual vs predicted for train data
    plt.scatter(y_train, y_train_pred, c='b', marker='^', label='Train')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title("Actual vs Predicted (Train Data)")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Plot actual vs predicted for test data
    plt.scatter(y_test, y_test_pred, c='r', marker='*', label='Test')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title("Actual vs Predicted (Test Data)")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
In [40]: %%time
         bagging_param_grid = {'n_estimators': [50, 100, 200], 'max_samples': [0.5, 0.8, 1.0]}
         grid_search_regression("BaggingRegressor", bagging_param_grid)
```

```
Best parameters:  {'max_samples': 1.0, 'n_estimators': 200}
Best R-squared score: 0.912
R-squared score on unseen data (test set): 0.918
```

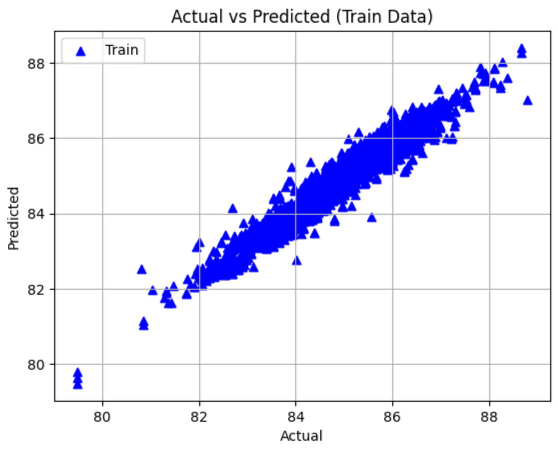


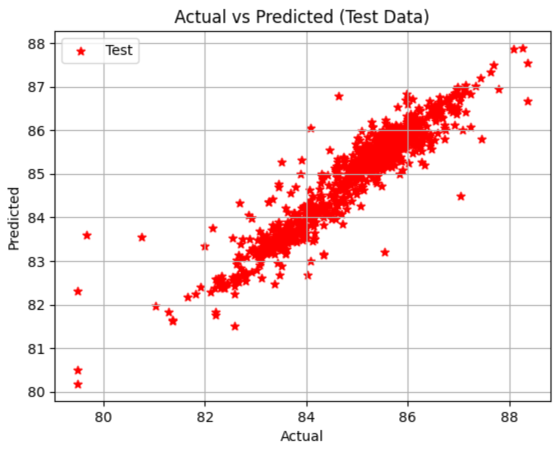


```
CPU times: total: 4min 11s
Wall time: 4min 12s
```

```
In [41]: xgb_param_grid = {'learning_rate': [0.1, 0.01, 0.001], 'max_depth': [3, 5, 7]}
         grid_search_regression("XGBRegressor", xgb_param_grid)
```

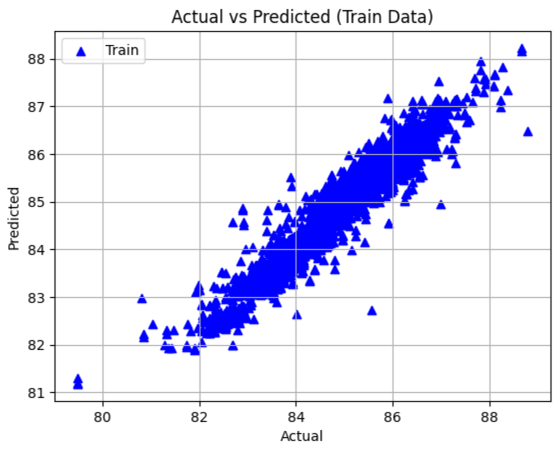
```
Best parameters:  {'learning_rate': 0.1, 'max_depth': 7}
Best R-squared score: 0.892
R-squared score on unseen data (test set): 0.891
```

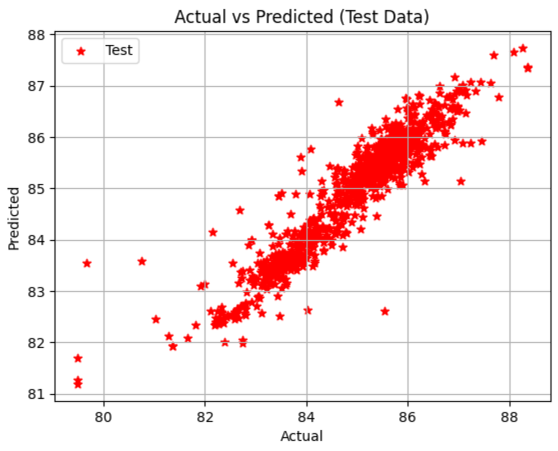




```
In [42]: lgbm_param_grid = {'learning_rate': [0.1, 0.01, 0.001], 'max_depth': [3, 5, 7]}
         grid_search_regression("LGBMRegressor", lgbm_param_grid)
```

```
Best parameters:  {'learning_rate': 0.1, 'max_depth': 7}
Best R-squared score: 0.892
R-squared score on unseen data (test set): 0.885
```





END