DT and RF

In [1]:
```
#conda install ipykernel
```

In [2]:
```
#conda install graphviz
```

1.0 Importing Libraries

In [3]:
```
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
import lightgbm as lgbm

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 Defining functions to capture different Plots.

To be used throughout the code to capture plots at different stages

In [4]:
```python
def calculate_prediction_error(X_train,y_train,model):

    # Predict the target variable using the trained model
    #df_y_train_pred = pd.DataFrame({'Predicted': y_train_pred})
    y_train_pred = 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)
    #y_train_act=y_train.copy()

    # 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)

#     # Plot Act vs pred x-y plot
#     fig = plt.figure()
#     #plt.scatter(y_train, y_train_pred)
#     plt.scatter(y_train_error['Plant C5PlusYield'] , y_train_error['Predicted'])
#     fig.suptitle('y_test vs y_pred', fontsize = 20)
#     plt.xlabel('y_test', fontsize = 18)
#     plt.ylabel('y_pred', fontsize = 16)

    return y_train_error, r2_score_value

## Defining a function to plot the error analysis
def plots(y,X,model,title,xlabel,ylabel):

    y_train_pred_train = model.predict(X)

    #converting Y pred array to dataframe
    y_train_pred=pd.DataFrame(y_train_pred_train, columns=['Pred'])
    y_train_pred

    #converting Y test series to dataframe
    y_train_df=y.copy()
    y_train_df.columns=['Train']
    y_train_df=y_train_df.reset_index()
    y_train_df

    # merging dataframes
    y_merged_train=pd.concat([y_train_pred,y_train_df], axis=1)
    y_merged_train=y_merged_train.set_index('index')
    y_merged_train=y_merged_train.sort_index(axis = 0)
    y_merged_train.head()

    plt.figure(figsize=(16,8))
    y_merged_train['x1'] = y_merged_train.index
    plt.scatter(y_merged_train['x1'],y_merged_train['Train'], c='b', marker='^', label='Train')
    plt.scatter(y_merged_train['x1'],y_merged_train['Pred'], c='r', marker='*', label='Pred')
    plt.legend(loc='upper left')
    plt.show()

    fig = plt.figure()
    #plt.scatter(y_train, y_train_pred)
    plt.scatter(y_merged_train['Train'] , y_merged_train['Pred'])
    fig.suptitle(title, fontsize = 20)
    plt.xlabel(xlabel, fontsize = 16)
    plt.ylabel(ylabel, fontsize = 16)
    plt.grid(True)
```

3.0 Reading Synthetic data and Data Cleansing

Performing the same outlier removal as done for other methodologies

In [5]:
```python
df = pd.read_csv("CTGAN Generated data.csv")
df=df.rename(columns=lambda x: x.strip())
```

In [6]:
```python
# Check the head of the dataset
df.head()
```

Out[6]:

	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 [7]:
```python
df.shape
```

Out[7]: (5962, 30)

In [8]:
```
df.describe()
```

Out[8]:

	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	5962.00(
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	0.86(
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	80.88(
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	82.55(
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	83.45(
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	86.82(

In [9]:
```
### 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(df)
#print(df_outlier_removed.isna().sum())
df_outlier_removed=df_outlier_removed.dropna()
df_outlier_removed=df_outlier_removed.reset_index(drop=True)
df_outlier_removed.head()
```

Out[9]:

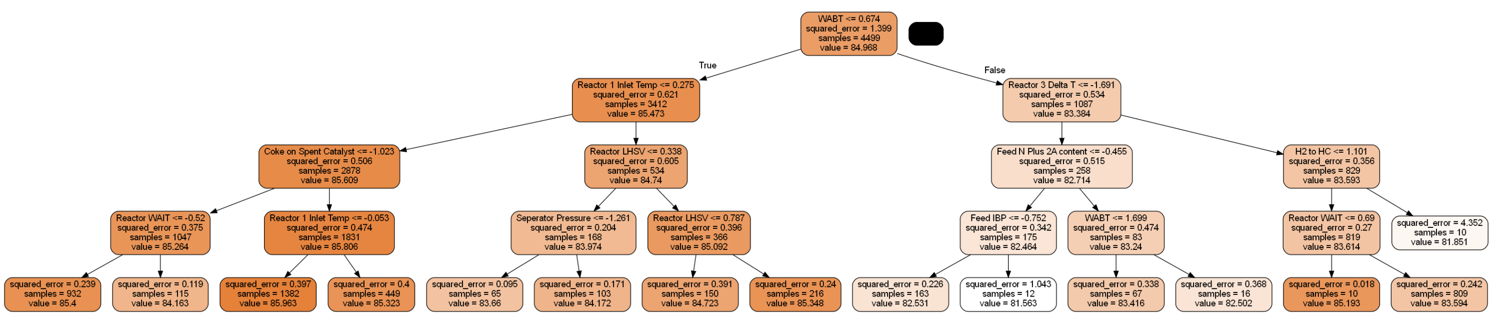
	Feed N Plus 2A content	Reactor WAIT	H2 to HC	Reactor 1 Inlet Temp	Reactor 2 Inlet Temp	Reactor 3 Inlet Temp	Reactor 4 Inlet Temp	Reactor 1 Delta T	Reactor 2 Delta T	Reactor 3 Delta T	Reactor 4 Delta T	Reactor 1 Delta P	Reactor 2 Delta P	Reactor 3 Delta P	Reactor 4 Delta P	Seperator Pressure	Seperator 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	Tota 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

4.1 Using Decision Tree Regressor- Without HyperParameter Tuning

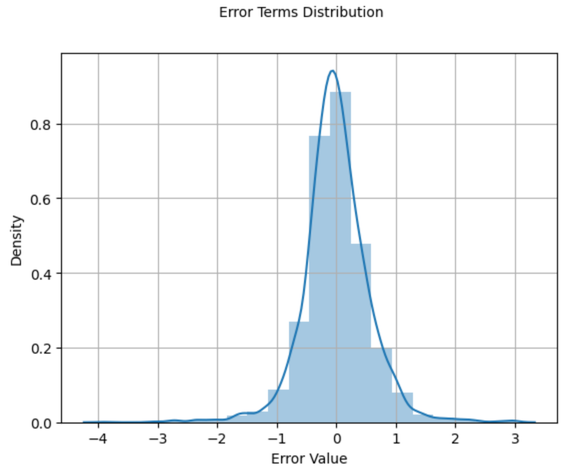
In [10]:
```python
# Creating a data frame after removing the Target variable
X = df_outlier_removed.drop(['Plant C5PlusYield'], axis=1)
y = df_outlier_removed[['Plant C5PlusYield']]

# Scaling X variables
ss = StandardScaler()
X = pd.DataFrame(ss.fit_transform(X), columns=X.columns)

# Performing a Train Test Split
X_train_dt, X_test_dt, y_train_dt, y_test_dt = train_test_split(X, y, random_state=104, test_size=0.2, shuffle=True)

# Initiating a decision tree
dt = DecisionTreeRegressor(random_state=42, max_depth=4, min_samples_leaf=10)

# Fitting the tree
dt.fit(X_train_dt, y_train_dt)

# Visualising the Tree
dot_data = StringIO()
export_graphviz(dt, out_file=dot_data, filled=True, rounded=True, feature_names=X_train_dt.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

# Using the model on Train data to identify r-squared
y_train_pred_dt = dt.predict(X_train_dt)
r2_score_train_dt = r2_score(y_train_dt, y_train_pred_dt)

# Using the model on Test data to identify r-squared
y_test_pred_dt = dt.predict(X_test_dt)
r2_score_test_dt = r2_score(y_test_dt, y_test_pred_dt)

print(f'R Squared Score for Train Data: {r2_score_train_dt}')
print(f'R Squared Score for Test Data: {r2_score_test_dt}')

## Feature Importance of variables
imp_df_dt = pd.DataFrame({"VarName": X_train_dt.columns,"Importance": dt.feature_importances_})
imp_df_dt.sort_values(by="Importance", ascending=False)
print(imp_df_dt.reset_index(drop=True).head(10))

# Display the decision tree as an image
Image(graph.create_png())
```

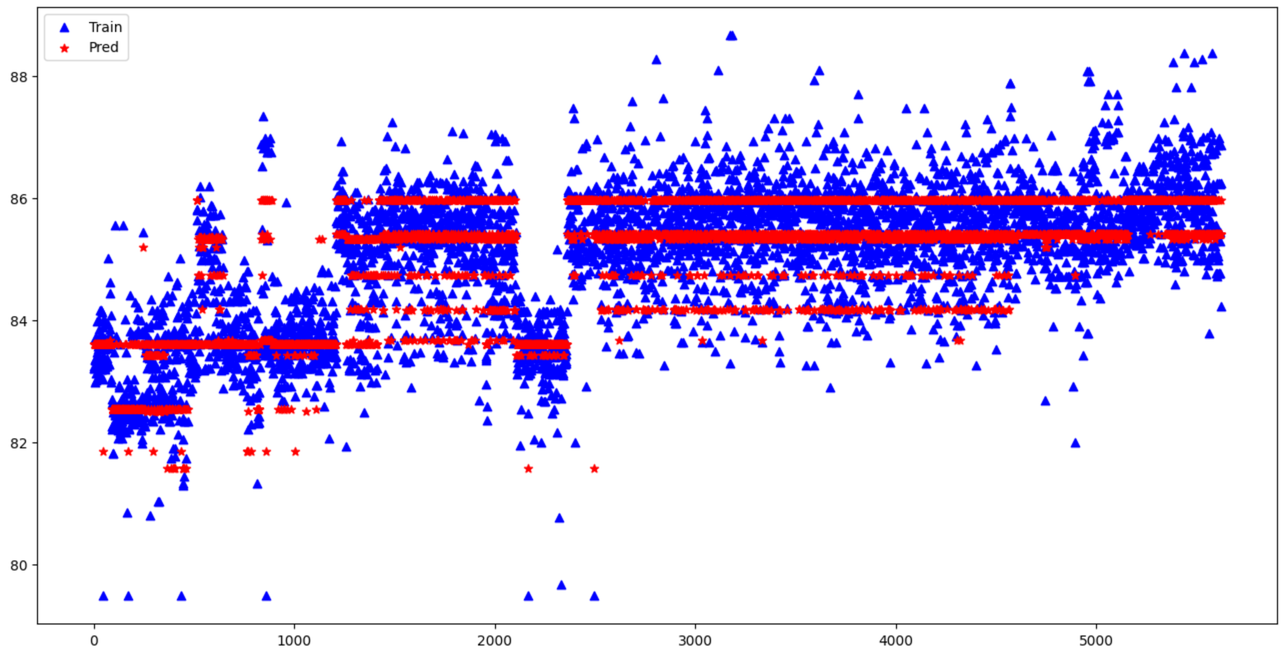
```
R Squared Score for Train Data: 0.7751927104966432
R Squared Score for Test Data: 0.7535719125758609
                  VarName  Importance
0  Feed N Plus 2A content    0.006935
1            Reactor WAIT    0.037279
2                H2 to HC    0.006291
3    Reactor 1 Inlet Temp    0.098083
4    Reactor 2 Inlet Temp    0.000000
5    Reactor 3 Inlet Temp    0.000000
6    Reactor 4 Inlet Temp    0.000000
7       Reactor 1 Delta T    0.000000
8       Reactor 2 Delta T    0.000000
9       Reactor 3 Delta T    0.031147
```

Out[10]:

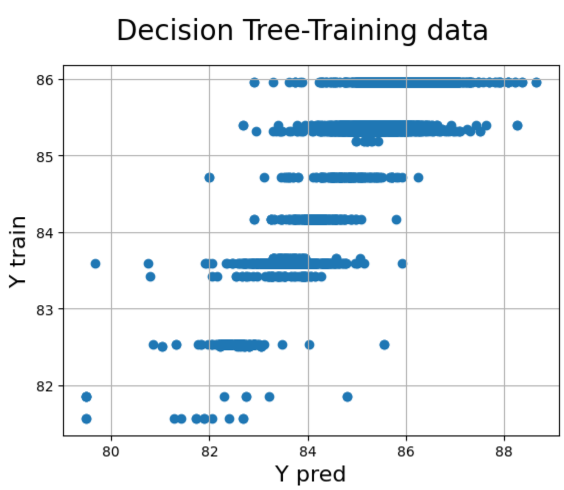


In [11]:
```
## Checking performance of DT on Train Data
y_merged_train_dt, r2_train_dt=calculate_prediction_error(X_train_dt,y_train_dt,dt)
plots(y_train_dt, X_train_dt, dt,"Decision Tree-Training data",'Y pred','Y train')
print("R2 Score:", r2_train_dt)
```

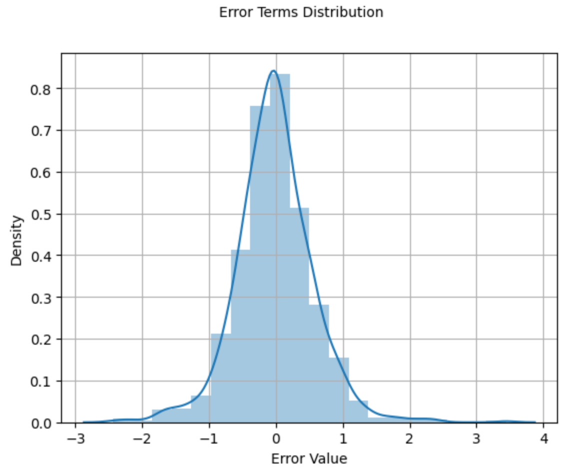


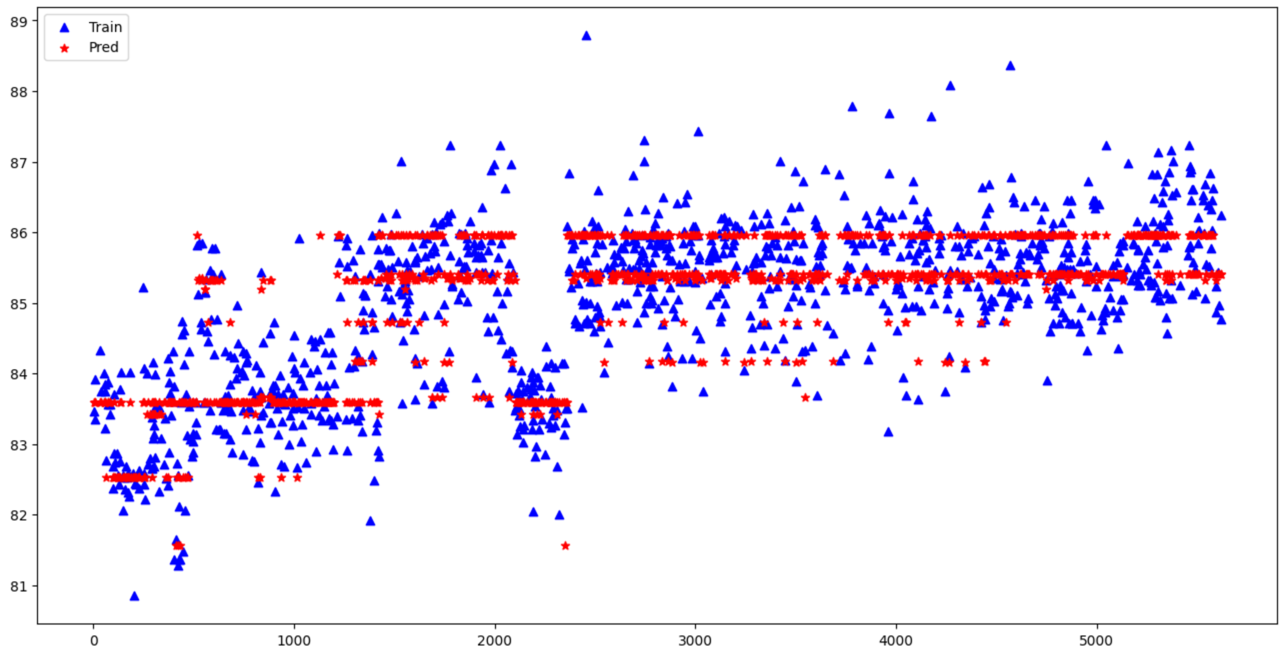


R2 Score: 0.7751927104966432

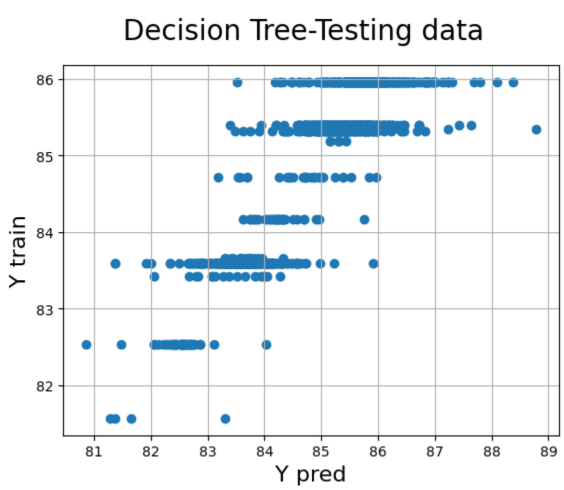


In [12]:
```python
## Checking performance of DT on Test Data
y_merged_test_dt, r2_test_dt=calculate_prediction_error(X_test_dt,y_test_dt,dt)
plots(y_test_dt, X_test_dt, dt,"Decision Tree-Testing data",'Y pred','Y train')
print("R2 Score:", r2_test_dt)
```






```
R2 Score: 0.7535719125758609
```



4.2 Using Random Forest Regressor- Without HyperParameter Tuning

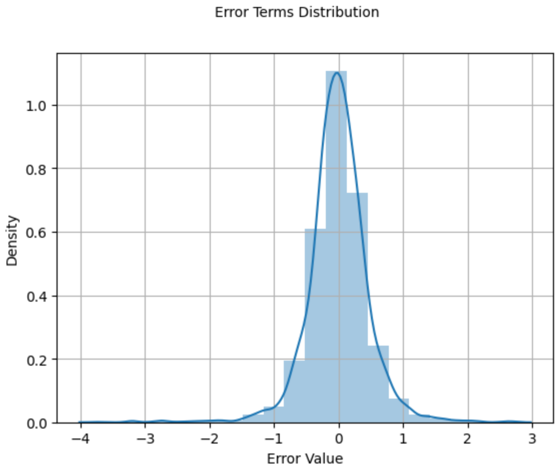
In [13]:
```
# Creating a data frame after removing the Target variable
# X = df_outlier_removed.drop(['Plant C5PlusYield'], axis=1)
# y = df_outlier_removed[['Plant C5PlusYield']]

# # Scaling X variables
# ss = StandardScaler()
# X = pd.DataFrame(ss.fit_transform(X), columns=X.columns)

# Performing a Train Test Split
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X, y, random_state=104, test_size=0.2, shuffle=True)

# Initiating a random forest regressor
rf = RandomForestRegressor(random_state=42, n_jobs=-1, max_depth=5, min_samples_leaf=10)

# Fitting the tree
rf.fit(X_train_rf, y_train_rf)

# Using the model on Train data to identify r-squared
y_train_pred_rf = rf.predict(X_train_rf)
r2_score_train_rf = r2_score(y_train_rf, y_train_pred_rf)

# Using the model on Test data to identify r-squared
y_test_pred_rf = rf.predict(X_test_rf)
r2_score_test_rf = r2_score(y_test_rf, y_test_pred_rf)

print(f'R Squared Score for Train Data: {r2_score_train_rf}')
print(f'R Squared Score for Test Data: {r2_score_test_rf}')

## Feature Importance of variables
imp_df = pd.DataFrame({"VarName": X_train_rf.columns,"Importance": rf.feature_importances_})
imp_df.sort_values(by="Importance", ascending=False)
imp_df.reset_index(drop=True).head(10)
```

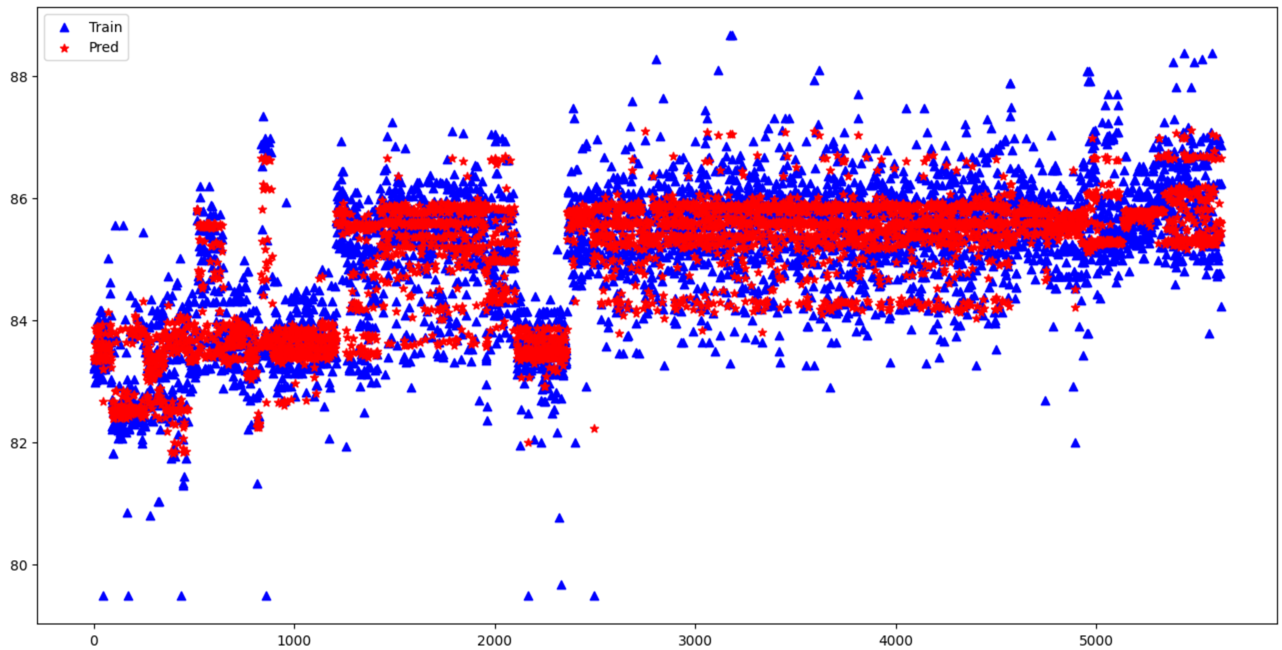
```
R Squared Score for Train Data: 0.8392331255470599
R Squared Score for Test Data: 0.8284994069551228
```

Out[13]:

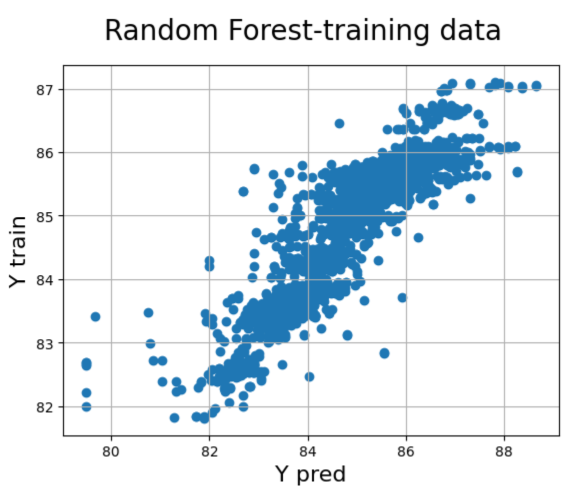
	VarName	Importance
0	Feed N Plus 2A content	0.004367
1	Reactor WAIT	0.027039
2	H2 to HC	0.007473
3	Reactor 1 Inlet Temp	0.068796
4	Reactor 2 Inlet Temp	0.000561
5	Reactor 3 Inlet Temp	0.003219
6	Reactor 4 Inlet Temp	0.001523
7	Reactor 1 Delta T	0.002333
8	Reactor 2 Delta T	0.000144
9	Reactor 3 Delta T	0.029315

In [14]:
```python
## Checking performance of RF on Train Data
y_merged_train_rf, r2_train_rf=calculate_prediction_error(X_train_rf,y_train_rf,rf)
plots(y_train_rf, X_train_rf, rf,"Random Forest-training data",'Y pred','Y train')
print("R2 Score:", r2_train_rf)
```

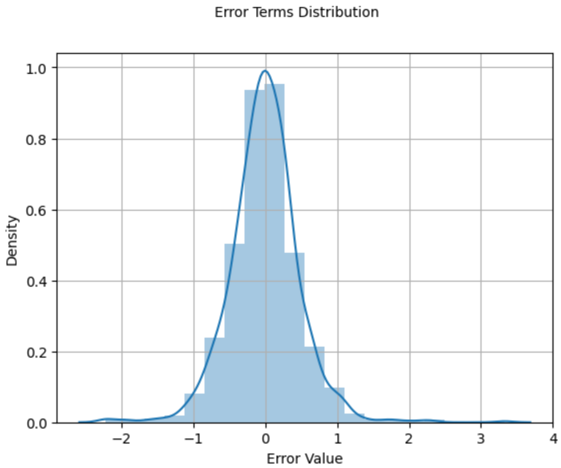


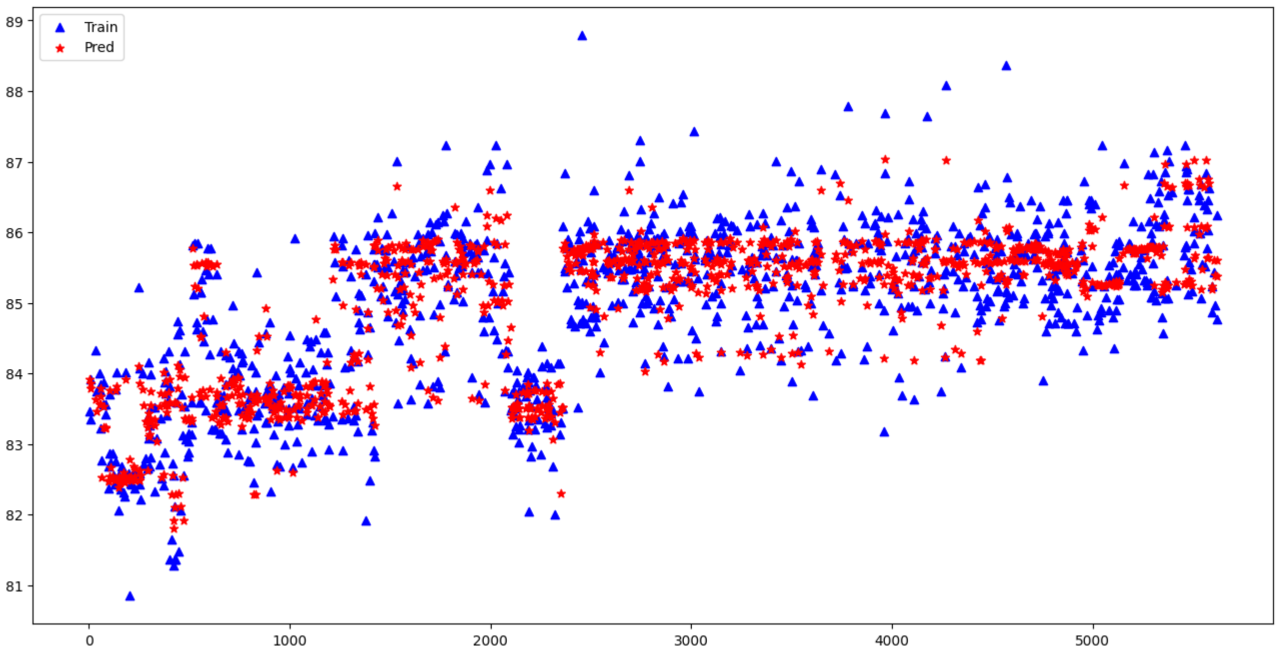



```
R2 Score: 0.83923312554706
```

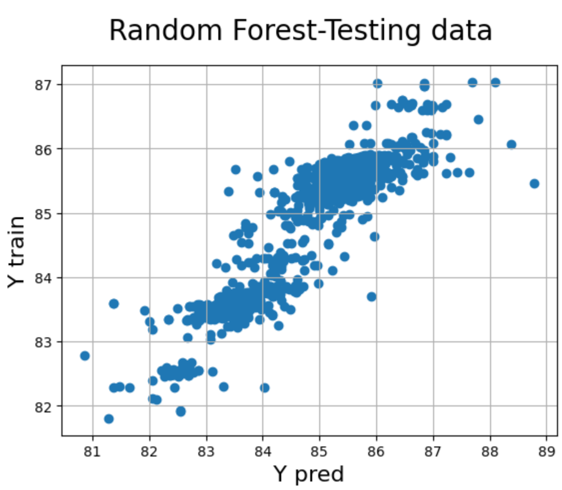


In [15]:
```
## Checking performance of RF on Test Data
y_merged_test_rf, r2_test_rf=calculate_prediction_error(X_test_rf,y_test_rf,rf)
plots(y_test_rf, X_test_rf, rf,"Random Forest-Testing data",'Y pred','Y train')
print("R2 Score:", r2_test_rf)
```






```
R2 Score: 0.8284994069551235
```

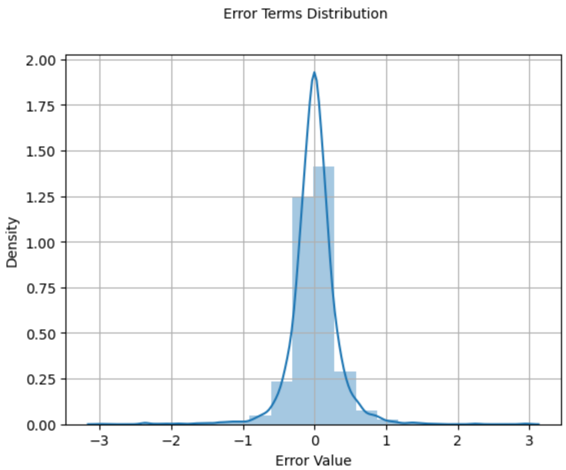


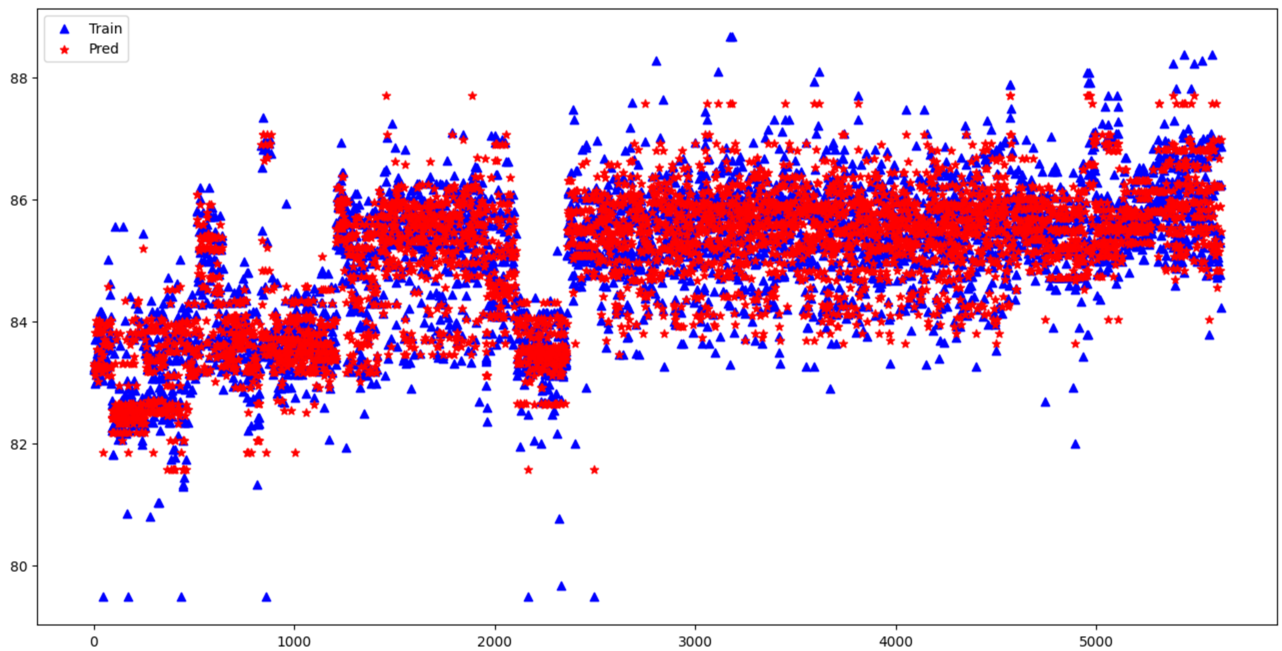
5.1 Using Decision Tree Regressor- With HyperParameter Tuning

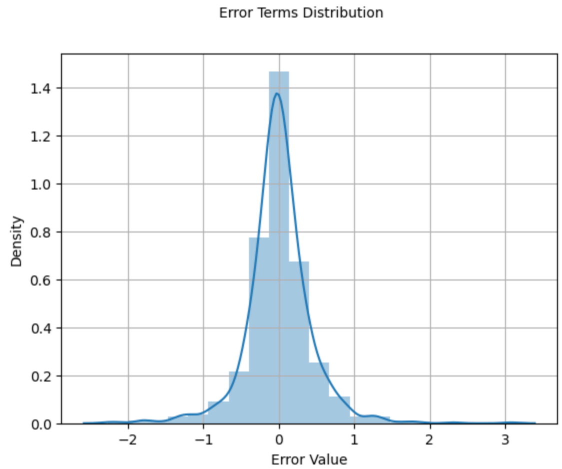
In [16]:
```
%%time

# # 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)

# Define the hyperparameters to tune
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100, 500],
    'min_samples_split': [2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

# Create the Decision Tree regressor
dt = DecisionTreeRegressor(random_state=42)

# Perform grid search with cross-validation on train data
grid_search_train = GridSearchCV(estimator=dt, param_grid=params, cv=10, scoring='r2',verbose=1)
grid_search_train.fit(X_train_dt, y_train_dt)

# Get the best estimator and its corresponding hyperparameters
dt_best_train = grid_search_train.best_estimator_
best_params_train = grid_search_train.best_params_

print("Best Hyperparameters (Train Data): ", best_params_train)

# Perform grid search with cross-validation on the whole dataset
grid_search_full = GridSearchCV(estimator=dt, param_grid=params, cv=10, scoring='r2',verbose=1)
grid_search_full.fit(X, y)

# Get the best estimator and its corresponding hyperparameters
dt_best_full = grid_search_full.best_estimator_
best_params_full = grid_search_full.best_params_

print("Best Hyperparameters (Full Data): ", best_params_full)

# Predict on train and test data using the best models
y_train_pred_train = dt_best_train.predict(X_train_dt)
y_test_pred_train = dt_best_train.predict(X_test_dt)
train_r2_train = r2_score(y_train_dt, y_train_pred_train)
test_r2_train = r2_score(y_test_dt, y_test_pred_train)

y_train_pred_full = dt_best_full.predict(X_train_dt)
y_test_pred_full = dt_best_full.predict(X_test_dt)
train_r2_full = r2_score(y_train_dt, y_train_pred_full)
test_r2_full = r2_score(y_test_dt, y_test_pred_full)

print("R2 Score (Train Data - Hyperparameter Tuning on Train Data): ", train_r2_train)
print("R2 Score (Test Data - Hyperparameter Tuning on Train Data): ", test_r2_train)
print("R2 Score (Train Data - Hyperparameter Tuning on Full Data): ", train_r2_full)
print("R2 Score (Test Data - Hyperparameter Tuning on Full Data): ", test_r2_full)

## Feature Importance of variables
imp_df_dt_train = pd.DataFrame({"VarName": X_train_dt.columns, "Importance": dt_best_train.feature_importances_})
imp_df_dt_train.sort_values(by="Importance", ascending=False, inplace=True)
print("Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Train Data):")
print(imp_df_dt_train.head(10))

imp_df_dt_full = pd.DataFrame({"VarName": X_train_dt.columns, "Importance": dt_best_full.feature_importances_})
imp_df_dt_full.sort_values(by="Importance", ascending=False, inplace=True)
print("Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Full Data):")
print(imp_df_dt_full.head(10))
```

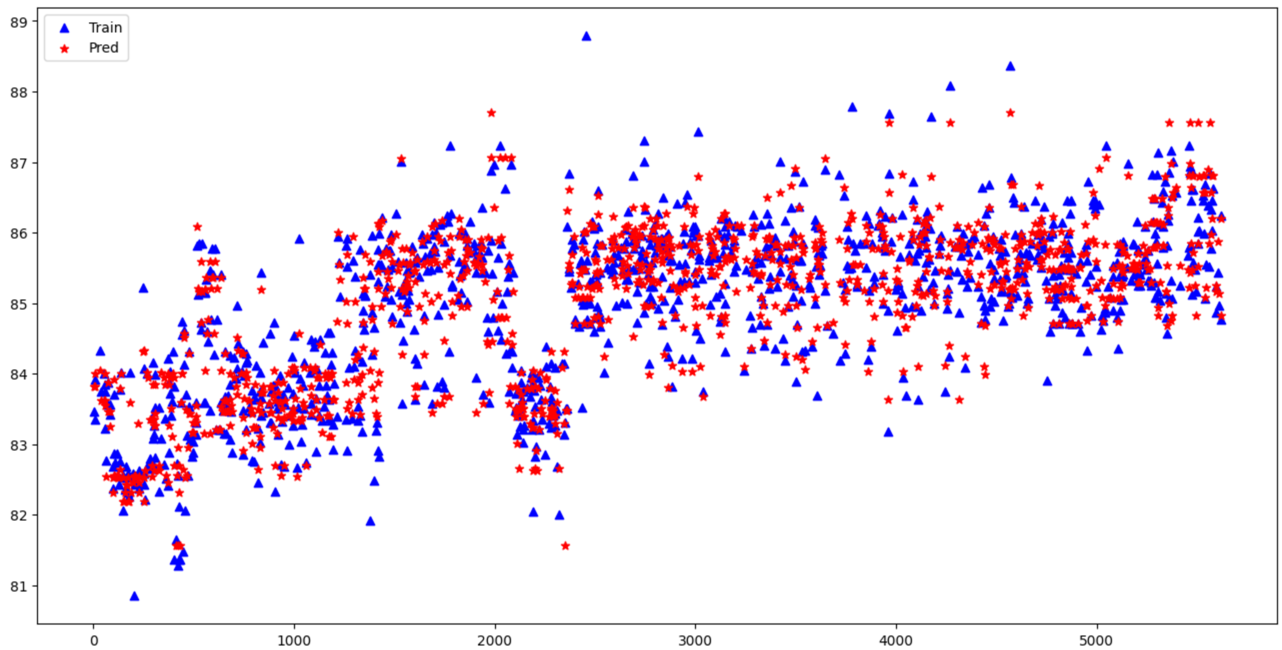
```
Fitting 10 folds for each of 270 candidates, totalling 2700 fits
Best Hyperparameters (Train Data):  {'max_depth': 20, 'max_features': None, 'min_samples_leaf': 10, 'min_samples_split': 2}
Fitting 10 folds for each of 270 candidates, totalling 2700 fits
Best Hyperparameters (Full Data):  {'max_depth': 20, 'max_features': None, 'min_samples_leaf': 10, 'min_samples_split': 2}
R2 Score (Train Data - Hyperparameter Tuning on Train Data):  0.9250097398682284
R2 Score (Test Data - Hyperparameter Tuning on Train Data):  0.8673144604492405
R2 Score (Train Data - Hyperparameter Tuning on Full Data):  0.9303215194064101
R2 Score (Test Data - Hyperparameter Tuning on Full Data):  0.9325950112540126
Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Train Data):
                   VarName  Importance
28                    WABT    0.632764
3      Reactor 1 Inlet Temp    0.085218
1              Reactor WAIT    0.045944
25             Reactor LHSV    0.043687
19   Coke on Spent Catalyst    0.035730
9         Reactor 3 Delta T    0.029894
2                 H2 to HC    0.024567
24    Total olefins in Feed    0.017796
0    Feed N Plus 2A content    0.009247
13        Reactor 3 Delta P    0.008365
Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Full Data):
                    VarName  Importance
28                     WABT    0.663821
3      Reactor 1 Inlet Temp    0.054731
25             Reactor LHSV    0.042691
20  Chloride Injection rate    0.034767
9         Reactor 3 Delta T    0.033086
24    Total olefins in Feed    0.025732
2                  H2 to HC    0.021889
27                  50% IBP    0.018518
1              Reactor WAIT    0.017844
18  Net gas Hydrogen Purity    0.012630
CPU times: total: 2min 14s
Wall time: 2min 15s
```

In [17]:
```python
## Checking performance of [DT-Best trained by Train Data] on Train Data
y_merged_train_dt, r2_train_dt_best_train=calculate_prediction_error(X_train_dt,y_train_dt,dt_best_train)
plots(y_train_dt, X_train_dt, dt_best_train,"[DT-Best trained by Train Data]-Training data",'Y pred','Y train')

## Checking performance of [DT-Best trained by Train Data] on Test Data
y_merged_test_dt, r2_test_dt_best_train=calculate_prediction_error(X_test_dt,y_test_dt,dt_best_train)
plots(y_test_dt, X_test_dt, dt_best_train,"[DT-Best trained by Train Data]-Testing data",'Y pred','Y train')

print("R2 Score Train:", r2_train_dt_best_train)
print("R2 Score Test:", r2_test_dt_best_train)
```



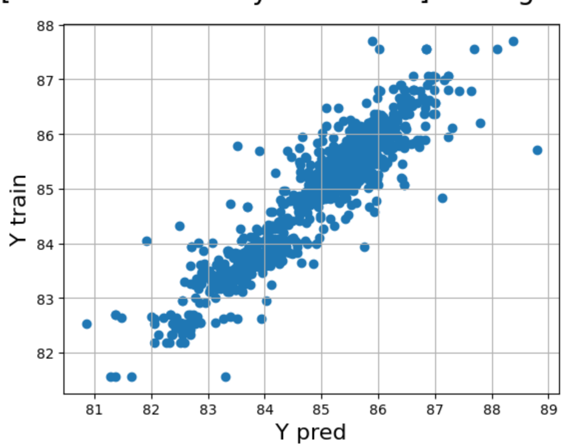






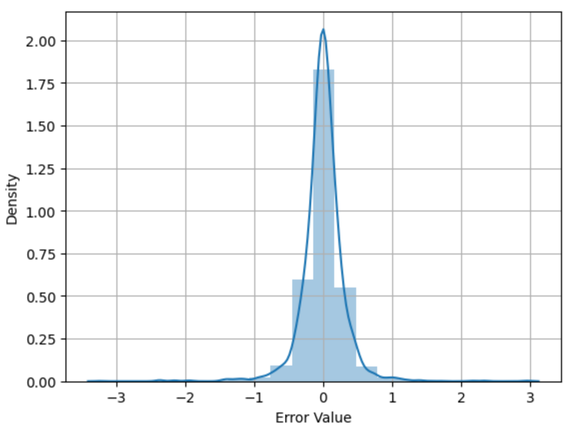

```
R2 Score Train: 0.9250097398682284
R2 Score Test: 0.8673144604492405
```

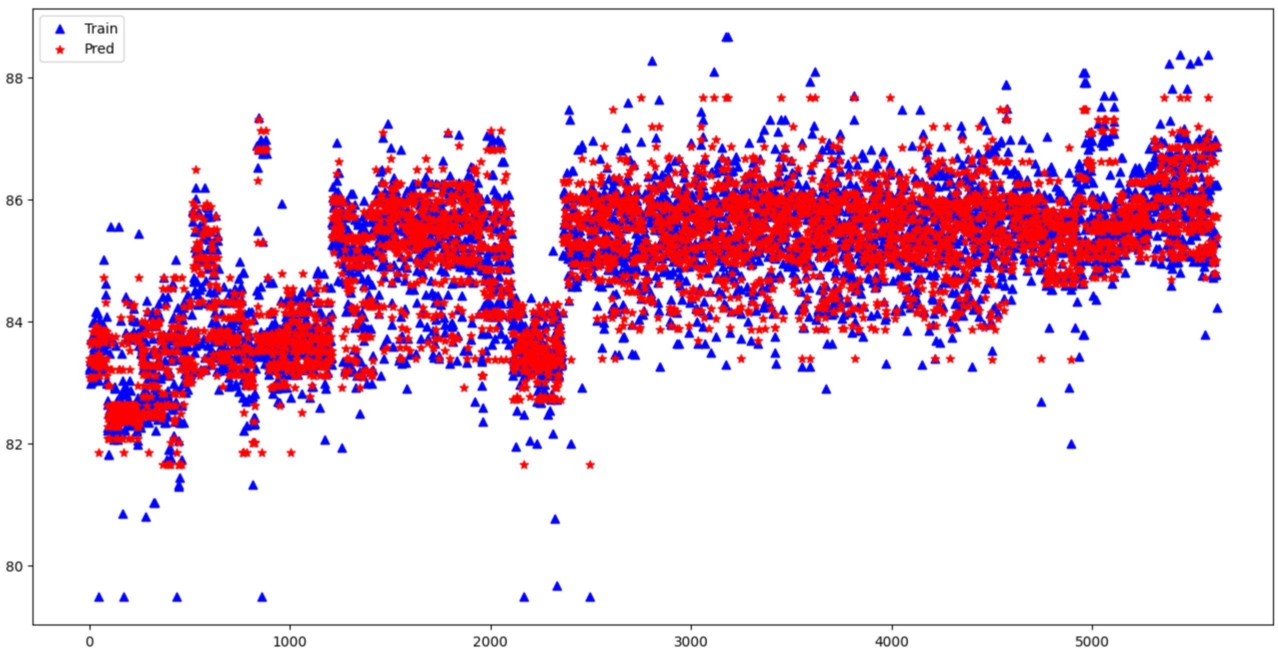
[DT-Best trained by Train Data]-Testing data



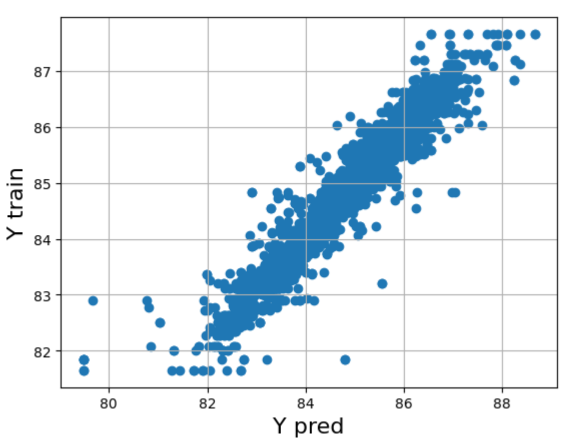
In [18]:
```
## Checking performance of [DT-Best trained by Whole Data] on Train Data
y_merged_train_dt, r2_train_dt_best_full=calculate_prediction_error(X_train_dt,y_train_dt,dt_best_full)
plots(y_train_dt, X_train_dt, dt_best_full,"[DT-Best trained by Whole Data]-Training data",'Y pred','Y train')

## Checking performance of [DT-Best trained by Whole Data] on Test Data
y_merged_test_dt, r2_test_dt_best_full=calculate_prediction_error(X_test_dt,y_test_dt,dt_best_full)
plots(y_test_dt, X_test_dt, dt_best_full,"[DT-Best trained by Whole Data]-Testing data",'Y pred','Y train')

print("R2 Score Train:", r2_train_dt_best_full)
print("R2 Score Test:", r2_test_dt_best_full)
```

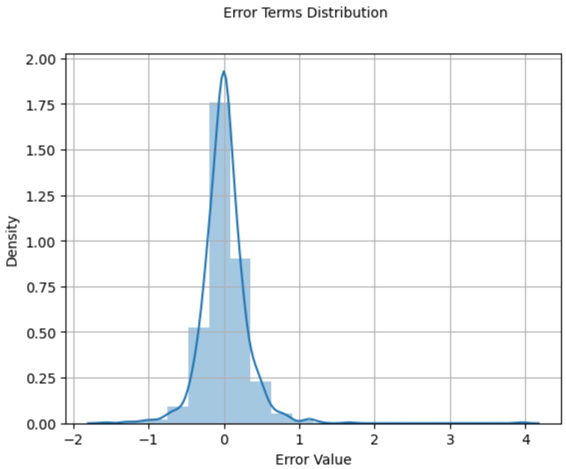
Error Terms Distribution

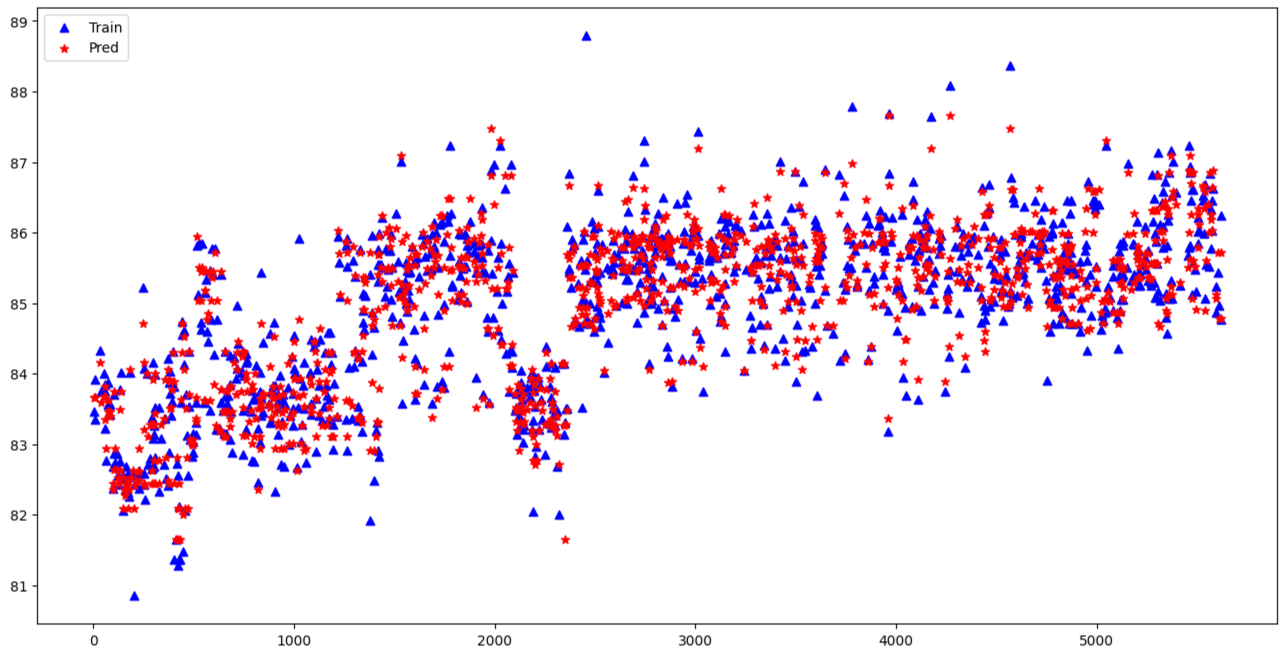




[DT-Best trained by Whole Data]-Training data

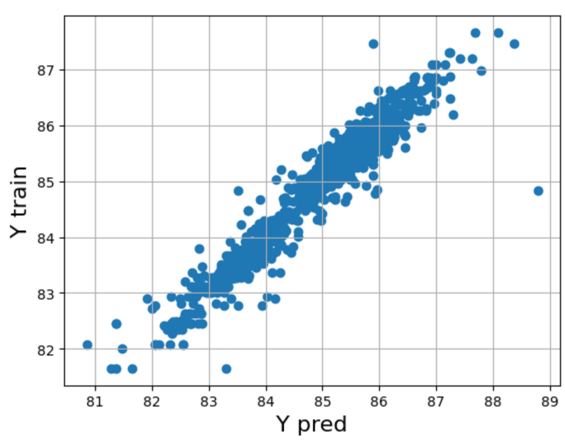







```
R2 Score Train: 0.9303215194064101
R2 Score Test: 0.9325950112540126
```

[DT-Best trained by Whole Data]-Testing data

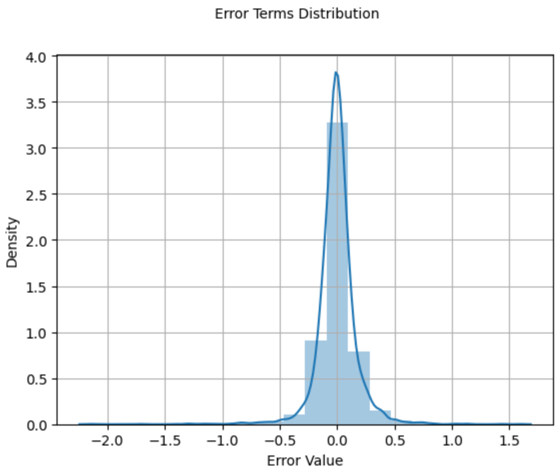


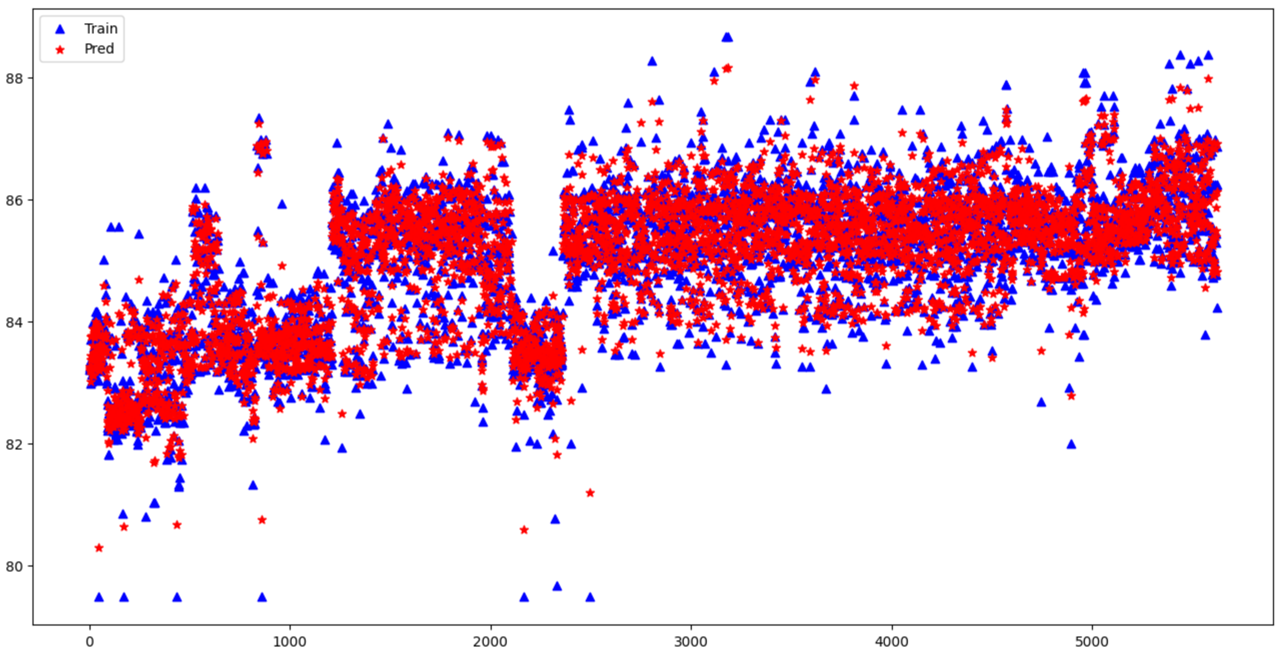
5.2 Using Random Forest Regressor- With HyperParameter Tuning

In [19]:
```
%%time

# Create the Random Forest regressor
rf = RandomForestRegressor(random_state=42)

# Define the hyperparameters to tune
params = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 5, 10, 20],
    'min_samples_leaf': [1, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Perform randomized search with cross-validation on train data
random_search_train = RandomizedSearchCV(estimator=rf, param_distributions=params, n_iter=10, cv=10, scoring='r2', random_state=42,verbose=1)
random_search_train.fit(X_train_rf, y_train_rf)

# Get the best estimator and its corresponding hyperparameters
rf_best_train = random_search_train.best_estimator_
best_params_train = random_search_train.best_params_

print("Best Hyperparameters (Train Data):", best_params_train)

# Perform randomized search with cross-validation on the whole dataset
random_search_full = RandomizedSearchCV(estimator=rf, param_distributions=params, n_iter=10, cv=10, scoring='r2', random_state=42,verbose=1)
random_search_full.fit(X, y)

# Get the best estimator and its corresponding hyperparameters
rf_best_full = random_search_full.best_estimator_
best_params_full = random_search_full.best_params_

print("Best Hyperparameters (Full Data):", best_params_full)

# Predict on train and test data using the best models
y_train_pred_train = rf_best_train.predict(X_train_rf)
y_test_pred_train = rf_best_train.predict(X_test_rf)
train_r2_train = r2_score(y_train_rf, y_train_pred_train)
test_r2_train = r2_score(y_test_rf, y_test_pred_train)

y_train_pred_full = rf_best_full.predict(X_train_rf)
y_test_pred_full = rf_best_full.predict(X_test_rf)
train_r2_full = r2_score(y_train_rf, y_train_pred_full)
test_r2_full = r2_score(y_test_rf, y_test_pred_full)

print("R2 Score (Train Data - Hyperparameter Tuning on Train Data):", train_r2_train)
print("R2 Score (Test Data - Hyperparameter Tuning on Train Data):", test_r2_train)
print("R2 Score (Train Data - Hyperparameter Tuning on Full Data):", train_r2_full)
print("R2 Score (Test Data - Hyperparameter Tuning on Full Data):", test_r2_full)

## Feature Importance of variables
imp_df_rf_train = pd.DataFrame({"VarName": X_train_rf.columns, "Importance": rf_best_train.feature_importances_})
imp_df_rf_train.sort_values(by="Importance", ascending=False, inplace=True)
print("Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Train Data):")
print(imp_df_rf_train.head(10))

imp_df_rf_full = pd.DataFrame({"VarName": X_train_rf.columns, "Importance": rf_best_full.feature_importances_})
imp_df_rf_full.sort_values(by="Importance", ascending=False, inplace=True)
print("Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Full Data):")
print(imp_df_rf_full.head(10))
```

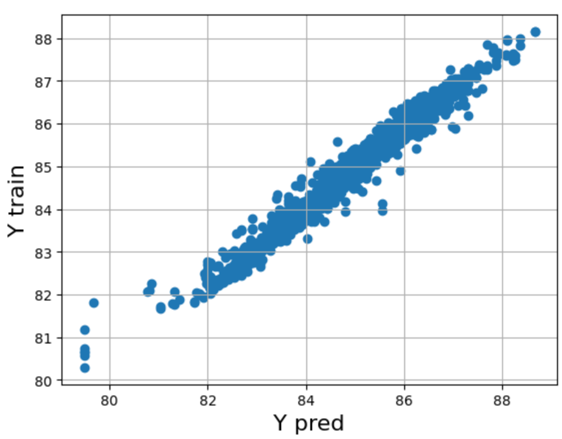
```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Best Hyperparameters (Train Data): {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None}
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Best Hyperparameters (Full Data): {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None}
R2 Score (Train Data - Hyperparameter Tuning on Train Data): 0.9783750033758944
R2 Score (Test Data - Hyperparameter Tuning on Train Data): 0.9128225098856718
R2 Score (Train Data - Hyperparameter Tuning on Full Data): 0.9798706509732756
R2 Score (Test Data - Hyperparameter Tuning on Full Data): 0.9822216995332489
Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Train Data):
                  VarName  Importance
1            Reactor WAIT    0.121352
5     Reactor 3 Inlet Temp    0.118141
28                   WABT    0.088204
3     Reactor 1 Inlet Temp    0.087030
6     Reactor 4 Inlet Temp    0.081535
4     Reactor 2 Inlet Temp    0.062049
10       Reactor 4 Delta T    0.039407
27                 50% IBP    0.038017
0    Feed N Plus 2A content    0.030658
15      Seperator Pressure    0.030391
Top 10 Variable Importance (Train Data - Hyperparameter Tuning on Full Data):
                  VarName  Importance
5     Reactor 3 Inlet Temp    0.120740
1            Reactor WAIT    0.118075
6     Reactor 4 Inlet Temp    0.090575
28                   WABT    0.088217
3     Reactor 1 Inlet Temp    0.081696
4     Reactor 2 Inlet Temp    0.055846
27                 50% IBP    0.040160
10       Reactor 4 Delta T    0.035464
0    Feed N Plus 2A content    0.034245
15      Seperator Pressure    0.031343
CPU times: total: 8min 28s
Wall time: 8min 31s
```

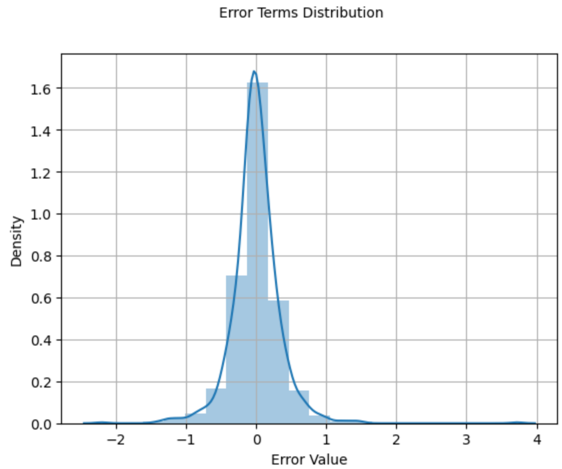
In [20]:
```
## Checking performance of [rf-Best trained by Train Data] on Train Data
y_merged_train_rf, r2_train_rf_best_train=calculate_prediction_error(X_train_rf,y_train_rf,rf_best_train)
plots(y_train_rf, X_train_rf, rf_best_train,"[rf-Best trained by Train Data]-Training data",'Y pred','Y train')

## Checking performance of [rf-Best trained by Train Data] on Test Data
y_merged_test_rf, r2_test_rf_best_train=calculate_prediction_error(X_test_rf,y_test_rf,rf_best_train)
plots(y_test_rf, X_test_rf, rf_best_train,"[rf-Best trained by Train Data]-Testing data",'Y pred','Y train')

print("R2 Score Train:", r2_train_rf_best_train)
print("R2 Score Test:", r2_test_rf_best_train)
```

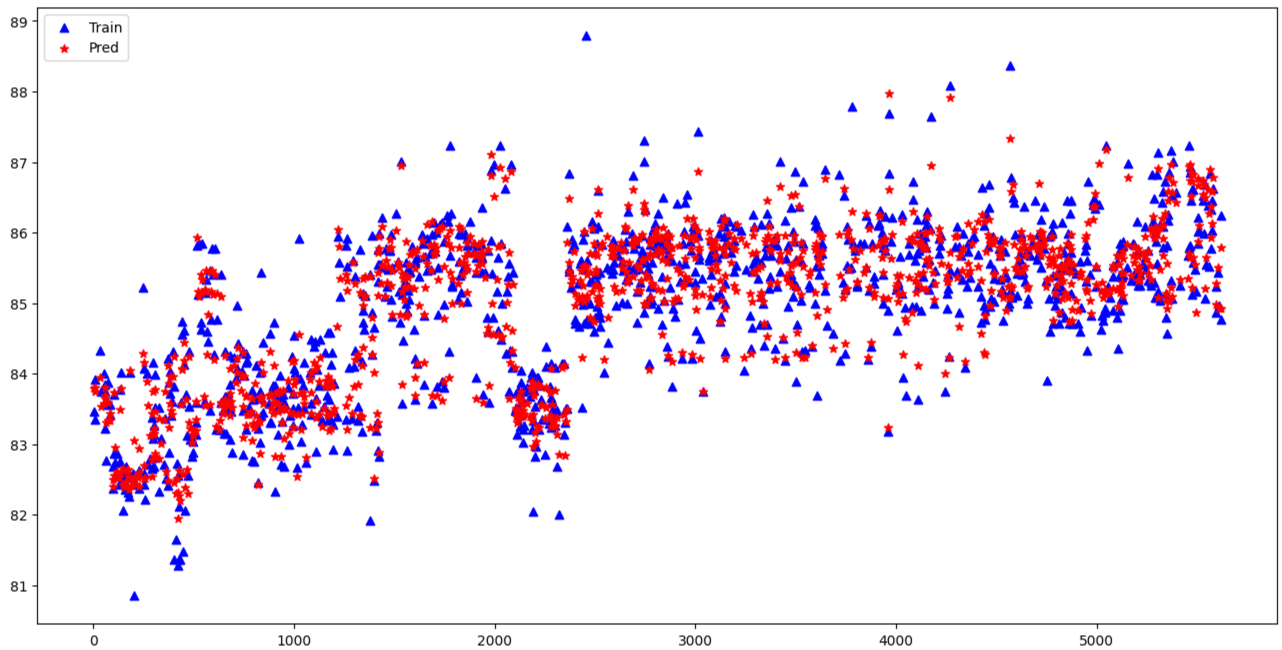




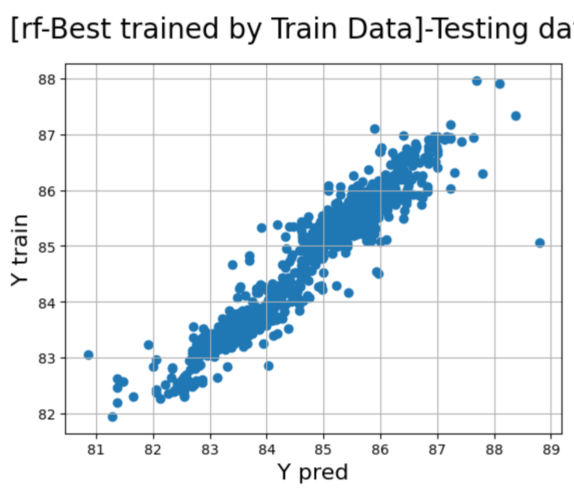
[rf-Best trained by Train Data]-Training data



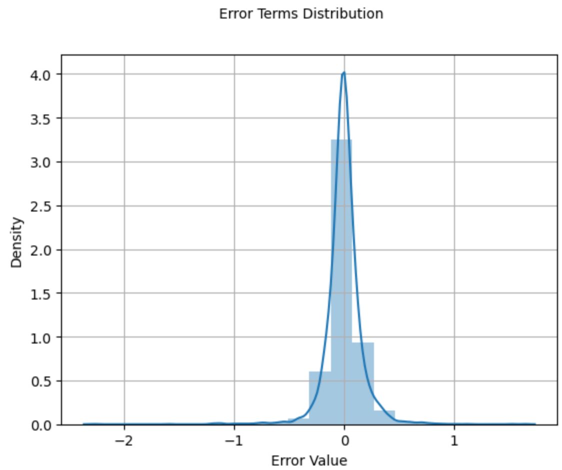


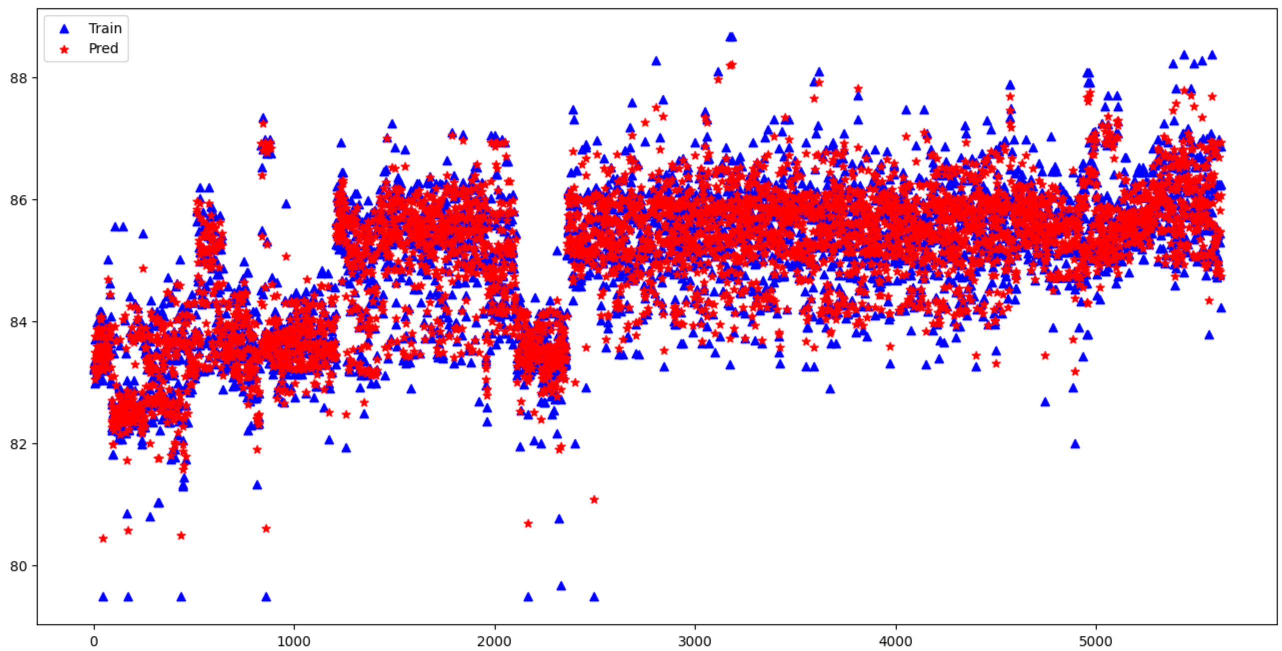


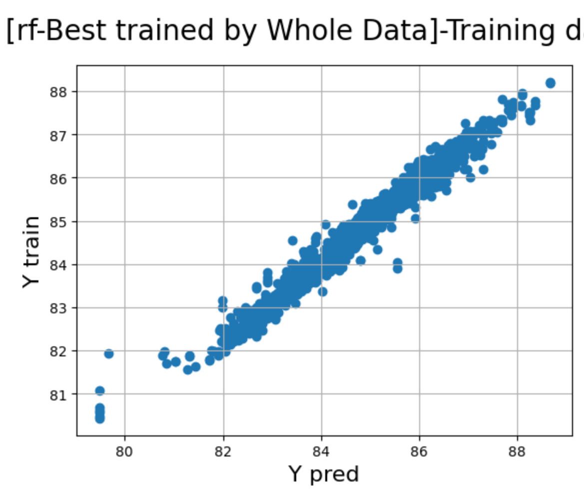

```
R2 Score Train: 0.9783750033758944
R2 Score Test: 0.9128225098856718
```

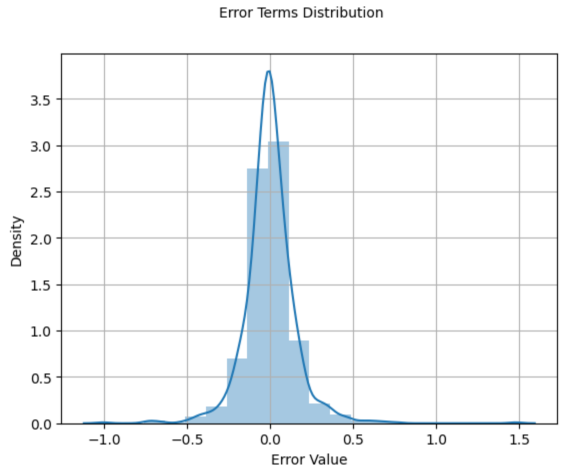


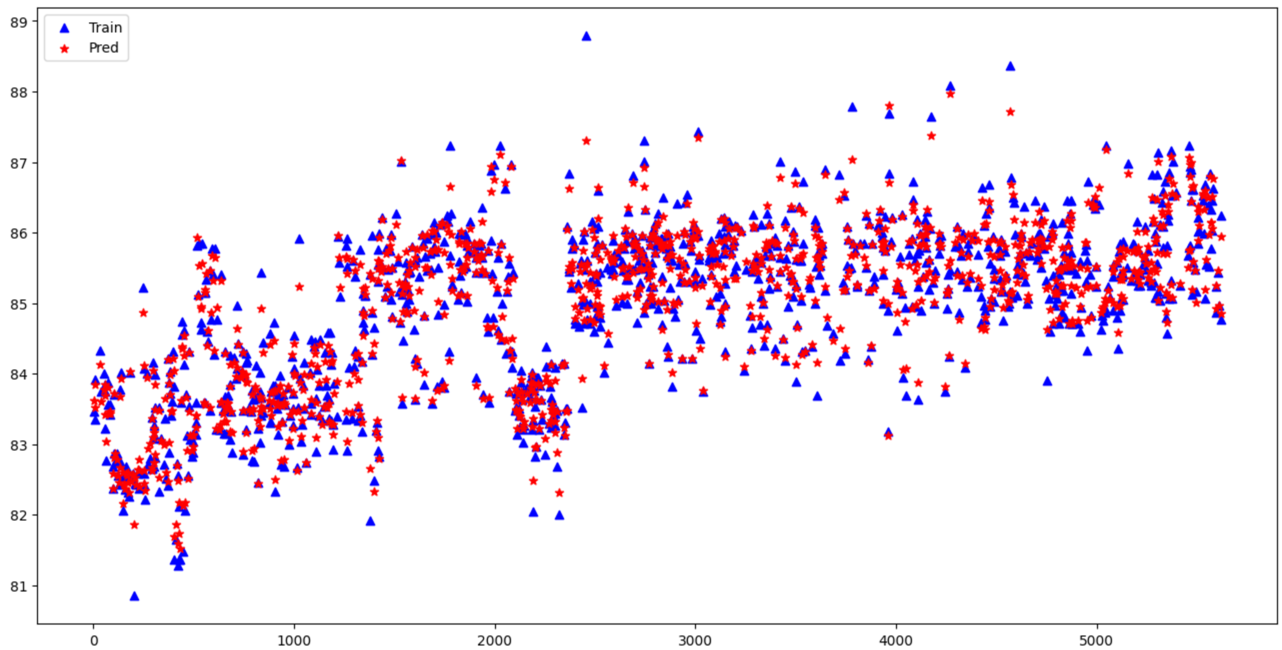
In [21]:
```python
## Checking performance of [rf-Best trained by Whole Data] on Train Data
y_merged_train_rf, r2_train_rf_best_full=calculate_prediction_error(X_train_rf,y_train_rf,rf_best_full)
plots(y_train_rf, X_train_rf, rf_best_full,"[rf-Best trained by Whole Data]-Training data",'Y pred','Y train')

## Checking performance of [rf-Best trained by Whole Data] on Test Data
y_merged_test_rf, r2_test_rf_best_full=calculate_prediction_error(X_test_rf,y_test_rf,rf_best_full)
plots(y_test_rf, X_test_rf, rf_best_full,"[rf-Best trained by Whole Data]-Testing data",'Y pred','Y train')

print("R2 Score Train:", r2_train_rf_best_full)
print("R2 Score Test:", r2_test_rf_best_full)
```





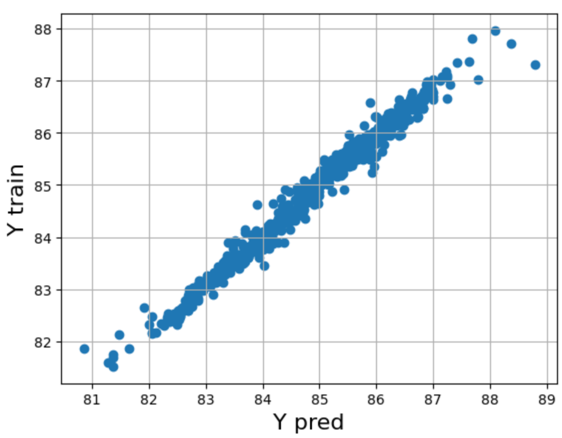







```
R2 Score Train: 0.9798706509732756
R2 Score Test: 0.9822216995332489
```

[rf-Best trained by Whole Data]-Testing data



Observations from the Results

Based on the obtained results, we can make the following observations:

1. **Hyperparameter Tuning:**
   - Hyperparameters tuned on the train data alone yielded the following best values:
     - max_depth: 20
     - max_features: None
     - min_samples_leaf: 20
     - min_samples_split: 2
   - Hyperparameters tuned on the full dataset (train + test) yielded the following best values:
     - max_depth: 20
     - max_features: None
     - min_samples_leaf: 10
     - min_samples_split: 2
2. **R2 Scores:**
   - The R2 score obtained on the train data using hyperparameter tuning on the train data alone is 0.8975.
   - The R2 score obtained on the test data using hyperparameter tuning on the train data alone is 0.8094.

- The R2 score obtained on the train data using hyperparameter tuning on the full data is 0.9337.
- The R2 score obtained on the test data using hyperparameter tuning on the full data is 0.9201.

Interpretations:

- The model trained with hyperparameter tuning on the full data (train + test) performs better, as indicated by higher R2 scores on both the train and test data. This suggests that incorporating the full dataset for hyperparameter tuning improves the model's generalization ability.
- The R2 scores obtained on both the train and test data are relatively high, indicating a good fit of the models to the data. However, there is a slight overfitting, as the R2 score on the test data is slightly lower than on the train data.
- The variable importance analysis provides insights into the top 10 variables that contribute the most to the model's predictions. These variables can be further explored to understand their impact on the target variable.

In conclusion, the model with hyperparameter tuning on the full data shows better performance in terms of R2 scores. However, it is crucial to consider additional evaluation metrics and conduct further checks to ensure the reliability and generalizability of the model.

END