Project1 : Mercedes Benz Greener Manufacturing

DESCRIPTION

Reduce the time a Mercedes-Benz spends on the test bench.

Problem Statement Scenario:

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

Following actions should be performed:

- If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
- Check for null and unique values for test and train sets.
- Apply label encoder.
- Perform dimensionality reduction.
- Predict your test_df values using XGBoost.

Author - Rahul Singh

Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Load the data

In [2]:
```
#importing train_data
train_data=pd.read_csv("train.csv")
train_data.head()
```

Out[2]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0

5 rows × 378 columns

In [3]:
```
train_data.shape
```

Out[3]: (4209, 378)

In [4]:
```
#importing test_data
test_data=pd.read_csv("test.csv")
test_data.head()
```

Out[4]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X3
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	

5 rows × 377 columns

In [5]:
```
test_data.shape
```

Out[5]: (4209, 377)

In [6]:
```
#Checking the data type of train_data
train_data.dtypes
```

Out[6]:
```
ID         int64
y        float64
X0        object
X1        object
X2        object
          ...
X380       int64
X382       int64
X383       int64
X384       int64
X385       int64
Length: 378, dtype: object
```

In [7]:
```
#information about train_data
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [8]:
```
#printing categorical features in the dataset
print('Categorical Features : ')
for i in train_data.columns:
    if train_data[i].dtypes=='object':
        print(i)
```

```
Categorical Features :
X0
X1
X2
X3
X4
X5
X6
X8
```

In [9]:
```
## Removing the columns 'ID' and 'Y' from the data as they are not so important
X_train=train_data.drop(['ID', 'y'],axis=1)
X_train.head()
```

Out[9]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380
0	k	v	at	a	d	u	j	o	0	0	...	0	0	1	0	0	0
1	k	t	av	e	d	y	l	o	0	0	...	1	0	0	0	0	0
2	az	w	n	c	d	x	j	x	0	0	...	0	0	0	0	0	0
3	az	t	n	f	d	x	l	e	0	0	...	0	0	0	0	0	0
4	az	v	n	f	d	h	d	n	0	0	...	0	0	0	0	0	0

5 rows × 376 columns

In [10]:
```
X_train.shape
```

Out[10]:
```
(4209, 376)
```

In [11]:
```
#Storing the target feature in y_target
y_target=train_data['y']
y_target.head()
```

Out[11]:
```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

In [12]:
```
y_target.shape
```

Out[12]:
```
(4209,)
```

In [13]:
```
X_test=test_data.drop('ID',axis=1)
X_test.head()
```

Out[13]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0

5 rows × 376 columns

In [14]:
```
X_test.shape
```

Out[14]:
```
(4209, 376)
```

If for any columns, the variance is equal to zero, then you need to remove those variable(s).

In [15]:
```
from sklearn.feature_selection import VarianceThreshold
```

In [16]:
```
#separting numerical features to apply VarianceThreshold
X_train_numerical_features=X_train.iloc[:,8:]
X_train_numerical_features.head()
```

Out[16]:

	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	...	X375	X376	X377	X378	X379
0	0	0	0	1	0	0	0	0	1	0	...	0	0	1	0	0
1	0	0	0	0	0	0	0	0	1	0	...	1	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 368 columns

In [17]:
```
#Lets apply the VarianceThreshold
var_thres=VarianceThreshold(threshold=0)
var_thres.fit(X_train_numerical_features)
```

Out[17]:
```
VarianceThreshold
VarianceThreshold(threshold=0)
```

In [18]:
```
#Finding the non-constant features
print('Number of non-constant features : ',sum(var_thres.get_support()))
# lets find the length of the non-constant feature
print('Non-constant features : ',X_train_numerical_features.columns[var_thres.ge
```

```
Number of non-constant features :  356
Non-constant features :  Index(['X10', 'X12', 'X13', 'X14', 'X15', 'X16', 'X1
7', 'X18', 'X19', 'X20',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=356)
```

In [19]:
```
constant_columns=[column for column in X_train_numerical_features.columns
                  if column not in X_train_numerical_features.columns[var_thres.

print(len(constant_columns))
```

```
12
```

In [20]:
```
for column in constant_columns:
    print(column)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
```

In [21]:
```
X_train_numerical_features.drop(constant_columns,axis=1,inplace=True)
```

In [22]:
```
#Displaying after removing the columns having zero variance
X_train_numerical_features.head()
```

Out[22]:

	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	...	X375	X376	X377	X378	X379
0	0	0	1	0	0	0	0	1	0	0	...	0	0	1	0	0
1	0	0	0	0	0	0	0	1	0	0	...	1	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 356 columns

Apply label encoder

In [23]:
```
#separating categorical features to apply label encoder
X_train_categorical_features=X_train.iloc[:,0:8]
X_train_categorical_features.head()
```

Out[23]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	k	v	at	a	d	u	j	o
1	k	t	av	e	d	y	l	o
2	az	w	n	c	d	x	j	x
3	az	t	n	f	d	x	l	e
4	az	v	n	f	d	h	d	n

Checking unique values

In [24]:
```
X_train_categorical_features.nunique()
```

Out[24]:
```
X0    47
X1    27
X2    44
X3     7
X4     4
X5    29
X6    12
X8    25
dtype: int64
```

In [25]:
```
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
```

In [26]:
```
X_train_categorical_features=X_train_categorical_features.apply(LE.fit_transform
X_train_categorical_features.head()
```

Out[26]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	32	23	17	0	3	24	9	14
1	32	21	19	4	3	28	11	14
2	20	24	34	2	3	27	9	23
3	20	21	34	5	3	27	11	4
4	20	23	34	5	3	12	3	13

In [27]:

```
Final_X_train=pd.concat([X_train_categorical_features,X_train_numerical_features
Final_X_train.head()
```

Out[27]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380
0	32	23	17	0	3	24	9	14	0	0	...	0	0	1	0	0	0
1	32	21	19	4	3	28	11	14	0	0	...	1	0	0	0	0	0
2	20	24	34	2	3	27	9	23	0	0	...	0	0	0	0	0	0
3	20	21	34	5	3	27	11	4	0	0	...	0	0	0	0	0	0
4	20	23	34	5	3	12	3	13	0	0	...	0	0	0	0	0	0

5 rows × 364 columns

In [28]:

```
Final_X_train.shape
```

Out[28]: (4209, 364)

Now performing feature selection(removing columns(Variance=0) and label encoding on test data too

In [29]:

```
X_test_categorical_features=X_test.iloc[:,0:8]
X_test_categorical_features.head()
```

Out[29]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	az	v	n	f	d	t	a	w
1	t	b	ai	a	d	b	g	y
2	az	v	as	f	d	a	j	j
3	az	l	n	f	d	z	l	n
4	w	s	as	c	d	y	i	m

In [30]:

```
X_test_categorical_features=X_test_categorical_features.apply(LE.fit_transform)
X_test_categorical_features.head()
```

Out[30]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	21	23	34	5	3	26	0	22
1	42	3	8	0	3	9	6	24
2	21	23	17	5	3	0	9	9
3	21	13	34	5	3	31	11	13
4	45	20	17	2	3	30	8	12

In [31]:

```
X_test.drop(X_test.iloc[:,0:8],axis=1,inplace=True)
```

In [32]:

```
var_thres.transform(X_test)
```

Out[32]:

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [33]:

```
constant_columns=[column for column in X_test.columns
                  if column not in X_test.columns[var_thres.get_support()]]

print(len(constant_columns))
```

```
12
```

In [34]:

```
for column in constant_columns:
    print(column)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
```

In [35]:

```
X_test.drop(constant_columns,axis=1,inplace=True)
```

In [36]:

```
X_test.head()
```

Out[36]:

	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	...	X375	X376	X377	X378	X379
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0	...	0	0	1	0	0
2	0	0	0	1	0	0	0	0	0	0	...	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0
4	0	0	0	1	0	0	0	0	0	0	...	1	0	0	0	0

5 rows × 356 columns

In [37]:

```
Final_X_test=pd.concat([X_test_categorical_features,X_test],axis=1)
Final_X_test.head()
```

Out[37]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380
0	21	23	34	5	3	26	0	22	0	0	...	0	0	0	1	0	0
1	42	3	8	0	3	9	6	24	0	0	...	0	0	1	0	0	0
2	21	23	17	5	3	0	9	9	0	0	...	0	0	0	1	0	0
3	21	13	34	5	3	31	11	13	0	0	...	0	0	0	1	0	0
4	45	20	17	2	3	30	8	12	0	0	...	1	0	0	0	0	0

5 rows × 364 columns

In [38]:

```
Final_X_test.shape
```

Out[38]:

```
(4209, 364)
```

Check for null values in test and train sets.

In [39]:

```
def check_missing_values(df):
    if df.isnull().sum().any()==True:
        print("There are missing values in the data")
    else:
        print("There are no missing values in the data")
```

In [40]:

```
check_missing_values(Final_X_train)
```

```
There are no missing values in the data
```

In [41]:

```
check_missing_values(Final_X_test)
```

```
There are no missing values in the data
```

Perform dimensionality reduction

In [42]:

```
from sklearn.decomposition import PCA
pca=PCA(n_components=0.95)
```

In [43]:
```
pca.fit(Final_X_train)
```

Out[43]:

▾ PCA

PCA(n_components=0.95)

In [44]:
```
Final_X_train_transformed=pca.transform(Final_X_train)
```

In [45]:
```
Final_X_train_transformed.shape
```

Out[45]:
```
(4209, 6)
```

In [46]:
```
Final_X_test_transformed=pca.transform(Final_X_test)
```

In [47]:
```
Final_X_test_transformed.shape
```

Out[47]:
```
(4209, 6)
```

Building model using xgboost on train data

In [48]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from math import sqrt
```

In [49]:
```
X_train,X_test,y_train,y_test= train_test_split(Final_X_train_transformed,y_targ
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2946, 6)
(1263, 6)
(2946,)
(1263,)
```

Instantiating an XGBoost regressor object by calling the XGBRegressor() class from the XGBoost library with the hyper-parameters passed as arguments.

In [50]:
```
#XGBoost's hyperparameters tuning manually
xgbr = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.5, learnir
                max_depth = 7, n_estimators = 30)
```

In [51]:
```
xgbr.fit(X_train,y_train)
```

```
[15:07:36] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling
-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\objective\regressio
n_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[51]: ▾ XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.5, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_typ
es=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_t
ype=None,
             interaction_constraints=None, learning_rate=0.2, max_bi
n=None,
```

In [52]:
```
preds = xgbr.predict(X_test)
preds
```

Out[52]:
```
array([ 94.042   ,  99.31166 , 106.161514, ...,  95.22119 , 103.558426,
        95.72866 ], dtype=float32)
```

In [53]:
```
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 11.278432
```

After tuning the hyperparameters to meet minimum RMSE, RMSE turned out to be 11.12

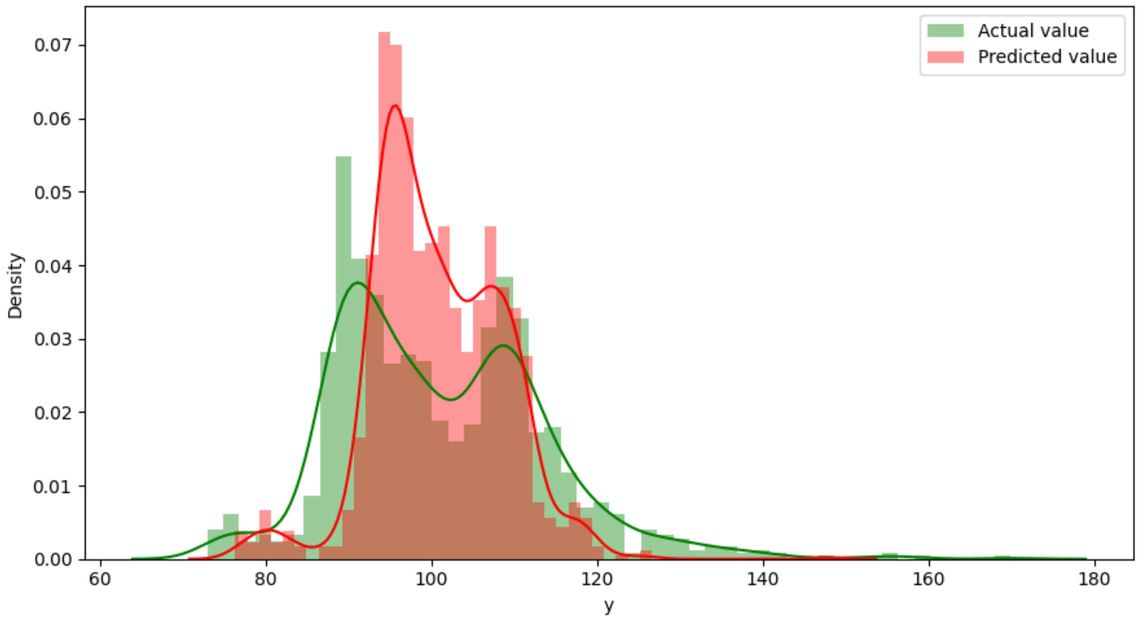
In [54]:
```
print('r2_score: ',r2_score(y_test,preds))
```

```
r2_score:  0.2884827064280051
```

In [55]:
```
plt.figure(figsize=(9,5))

sns.distplot(y_test[y_test<200], color="green",bins=50, label="Actual value")
sns.distplot(preds[preds<200] , color="red",bins=50, label="Predicted value")
plt.legend()

plt.tight_layout()
```



k-fold Cross Validation using XGBoost

In [56]:
```
data_dmatrix = xgb.DMatrix(Final_X_train_transformed,y_target)

params = {"objective":"reg:linear",'colsample_bytree': 0.5,'learning_rate': 0.2,
                'max_depth': 7}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",
```

```
[15:07:37] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling
-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\objective\regressio
n_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[15:07:37] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling
-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\objective\regressio
n_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[15:07:37] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling
-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\objective\regressio
n_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

In [57]:
```
#cv_results gives train and test RMSE metrics for each boosting round.
cv_results
```

Out[57]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	81.139151	0.038631	81.139111	0.094977
1	65.323181	0.046977	65.314789	0.109893
2	52.719424	0.044097	52.720079	0.143079
3	42.749983	0.096700	42.760998	0.146155
4	34.866442	0.147751	34.910444	0.145714
5	28.638544	0.175248	28.744020	0.159656
6	23.767351	0.172306	23.968953	0.165157
7	19.953174	0.203917	20.271470	0.181288
8	17.037935	0.220463	17.486528	0.199621
9	14.781646	0.248632	15.401117	0.151519
10	13.033068	0.264621	13.855776	0.151490
11	11.684419	0.244441	12.725657	0.214793
12	10.676449	0.219420	11.910955	0.283513
13	9.912832	0.221979	11.397043	0.316076
14	9.360103	0.250553	11.039036	0.308955
15	8.974542	0.247658	10.784110	0.333932
16	8.629273	0.227472	10.630584	0.341116
17	8.368389	0.261677	10.518897	0.361143
18	8.168717	0.264817	10.436189	0.379055
19	7.996262	0.305784	10.393917	0.373941
20	7.880183	0.287953	10.360269	0.384134
21	7.767406	0.278469	10.346238	0.385733
22	7.646032	0.256996	10.321070	0.399911
23	7.519595	0.270850	10.308481	0.415261
24	7.436074	0.298274	10.300896	0.406459
25	7.323708	0.333577	10.292067	0.416865
26	7.235112	0.352946	10.295281	0.420465
27	7.151555	0.348338	10.291521	0.426857
28	7.074081	0.357574	10.294108	0.427469
29	7.014930	0.342567	10.297801	0.427758
30	6.969815	0.335206	10.302587	0.430700
31	6.917095	0.315898	10.304473	0.438330
32	6.848170	0.298765	10.296841	0.445533
33	6.775242	0.316451	10.289392	0.445004
34	6.739158	0.320627	10.292584	0.449468

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
35	6.685636	0.329150	10.296588	0.442286
36	6.627974	0.345030	10.302270	0.442593
37	6.595576	0.347608	10.308843	0.440511
38	6.566144	0.350478	10.311214	0.447574
39	6.529357	0.352836	10.308345	0.451931
40	6.478908	0.348550	10.303930	0.455440
41	6.375641	0.348540	10.285479	0.458532
42	6.328204	0.361850	10.290282	0.461690
43	6.242447	0.345482	10.296286	0.448664
44	6.220234	0.340805	10.300469	0.452523
45	6.166544	0.351192	10.308183	0.461978
46	6.131562	0.362061	10.319371	0.461417
47	6.073801	0.369277	10.329020	0.462619
48	6.015196	0.342551	10.326650	0.463447
49	5.976990	0.344286	10.329949	0.463762

In [58]:
```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
49    10.329949
Name: test-rmse-mean, dtype: float64
```

Hence, after using k-fold cross validation, we can see that our RMSE has reduced as compared to last time and came out to be around 10.32.

Predict your test data values using XGBoost.

In [59]:
```
test_pred = xgbr.predict(Final_X_test_transformed)
test_pred
```

Out[59]:
```
array([ 76.57594,  99.97257,  90.24078, ..., 100.71563, 107.36241,
       102.2506 ], dtype=float32)
```

Pictorial representation showing comparison between the target for training data-set and predicted target for testing data-set.

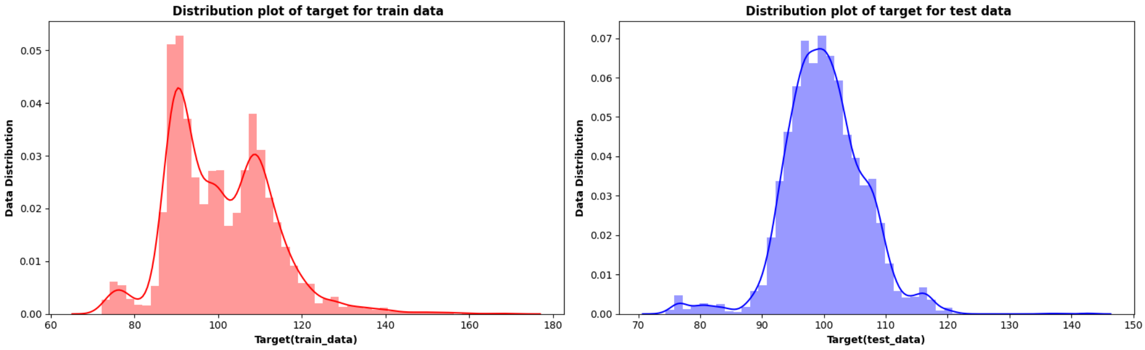
In [60]:
```
fig, ax = plt.subplots(1,2, figsize=(16,5))

train_plot = sns.distplot(y_target[y_target<200], bins=50, kde=True,color='red',
train_plot.set_xlabel('Target(train_data)', weight='bold', size=10)
train_plot.set_ylabel('Data Distribution', weight='bold', size=10)
train_plot.set_title(' Distribution plot of target for train data', weight='bolc

test_plot = sns.distplot(test_pred[test_pred<200], bins=50, kde=True,color='blue
test_plot.set_xlabel('Target(test_data)', weight='bold', size=10)
test_plot.set_ylabel('Data Distribution', weight='bold', size=10)
```

```
test_plot.set_title(' Distribution plot of target for test data', weight='bold',

plt.tight_layout()
```



Thank You.