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.

In [1]:
```
#Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Importing datasets
train_data=pd.read_csv('train1.csv')
test_data=pd.read_csv('test1.csv')
```

In [3]:
```
#Checking shape of the datasets
train_data.shape, test_data.shape
```

Out[3]:
```
((4209, 378), (4209, 377))
```

In [4]:
```
#Checking datatypes
train_data.dtypes
```

Out[4]:
```
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 [5]:
```
# Printing columns of object datatypes
for i in train_data.columns:
    data_type = train_data[i].dtype
    if data_type == 'object':
        print(i)
```

```
X0
X1
X2
X3
X4
X5
X6
X8
```

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

In [6]:
```
variance = pow(train_data.drop(columns={'ID','y'}).std(),2).to_dict()

null_cnt = 0
for key, value in variance.items():
    if(value==0):
        print('Name = ',key)
        null_cnt = null_cnt+1
print('No of columns which has zero variance = ',null_cnt)
```

```
Name =  X11
Name =  X93
Name =  X107
Name =  X233
Name =  X235
Name =  X268
Name =  X289
Name =  X290
Name =  X293
Name =  X297
Name =  X330
Name =  X347
No of columns which has zero variance =  12
```

Since these 12 columns have 0 variance, these all are need to be dropped

In [7]:
```
#Dropping 12 columns with 0 variance from train dataset
train_data.drop(['X11','X93','X107','X233','X235','X268','X289','X290','X293','X330','X2
```

In [8]:
```
train_data.shape
```

Out[8]:
```
(4209, 366)
```

In [9]:
```
#Dropping 12 columns with 0 variance from test dataset
test_data.drop(['X11','X93','X107','X233','X235','X268','X289','X290','X293','X330','X29
```

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

Out[10]:
```
(4209, 365)
```

Check for null and unique values for test and train sets

In [11]:
```
#Null in train dataset
train_data.isnull().sum().any()
```

Out[11]:
```
False
```

In [12]:
```
#checking null in test dataset
test_data.isnull().sum().any()
```

Out[12]:
```
False
```

It seems our datasets dont have any null values

In [13]:
```
#Cheching Unique values in train dataset
train_data['X0'].unique
```

Out[13]:
```
<bound method Series.unique of 0         k
1        k
2       az
3       az
4       az
        ..
4204    ak
4205     j
4206    ak
4207    al
4208     z
Name: X0, Length: 4209, dtype: object>
```

In [14]:
```
#Cheching Unique values in test dataset
test_data['X0'].unique
```

Out[14]:
```
<bound method Series.unique of 0        az
1        t
2       az
3       az
4        w
        ..
4204    aj
4205     t
4206     y
4207    ak
4208     t
Name: X0, Length: 4209, dtype: object>
```

In [15]:
```
train_data['X1'].unique
```

Out[15]:
```
<bound method Series.unique of 0        v
1       t
2       w
3       t
4       v
       ..
4204    s
4205    o
4206    v
4207    r
4208    r
Name: X1, Length: 4209, dtype: object>
```

In [16]:
```
test_data['X1'].unique
```

Out[16]:
```
<bound method Series.unique of 0         v
```

```
1          b
2          v
3          l
4          s
          ..
4204       h
4205      aa
4206       v
4207       v
4208      aa
Name: X1, Length: 4209, dtype: object>
```

Apply label encoder

In [17]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [18]:
```
#Assigning feature and target in train dataset
train_data_feature = train_data.drop(['y','ID'],axis=1)
train_data_target = train_data['y']
print(train_data_feature.shape)
print(train_data_target.shape)
```

```
(4209, 364)
(4209,)
```

In [19]:
```
#Describing object type variables in train dataset
train_data_feature.describe(include='object')
```

Out[19]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

In [20]:
```
train_data_feature['X0'] = le.fit_transform(train_data_feature.X0)
train_data_feature['X1'] = le.fit_transform(train_data_feature.X1)
train_data_feature['X2'] = le.fit_transform(train_data_feature.X2)
train_data_feature['X3'] = le.fit_transform(train_data_feature.X3)
train_data_feature['X4'] = le.fit_transform(train_data_feature.X4)
train_data_feature['X5'] = le.fit_transform(train_data_feature.X5)
train_data_feature['X6'] = le.fit_transform(train_data_feature.X6)
train_data_feature['X8'] = le.fit_transform(train_data_feature.X8)
```

Perform dimensionality reduction

In [21]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=.95)
```

In [22]:
```
#For Train dataset
pca.fit(train_data_feature, train_data_target)
```

Out[22]:
```
PCA(n_components=0.95)
```

In [23]:
```
train_data_feature_tf = pca.fit_transform(train_data_feature)
print(train_data_feature_tf.shape)
```

```
(4209, 6)
```

Predict your test_df values using XGBoost

Building model using the train data set.

In [24]:
```
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 [25]:
```
train_x,test_x,train_y,test_y = train_test_split(train_data_feature_tf,train_data_target
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
```

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

XGBoost's hyperparameters tuning manually

In [26]:
```
xgb_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rat
                           n_estimators = 20)
model = xgb_reg.fit(train_x,train_y)
print('RMSE = ',sqrt(mean_squared_error(model.predict(test_x),test_y)))
```

```
[17:02:32] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-030221e36e1a46bfb-1/xgboost/xgboost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
RMSE =  12.288794806074309
```

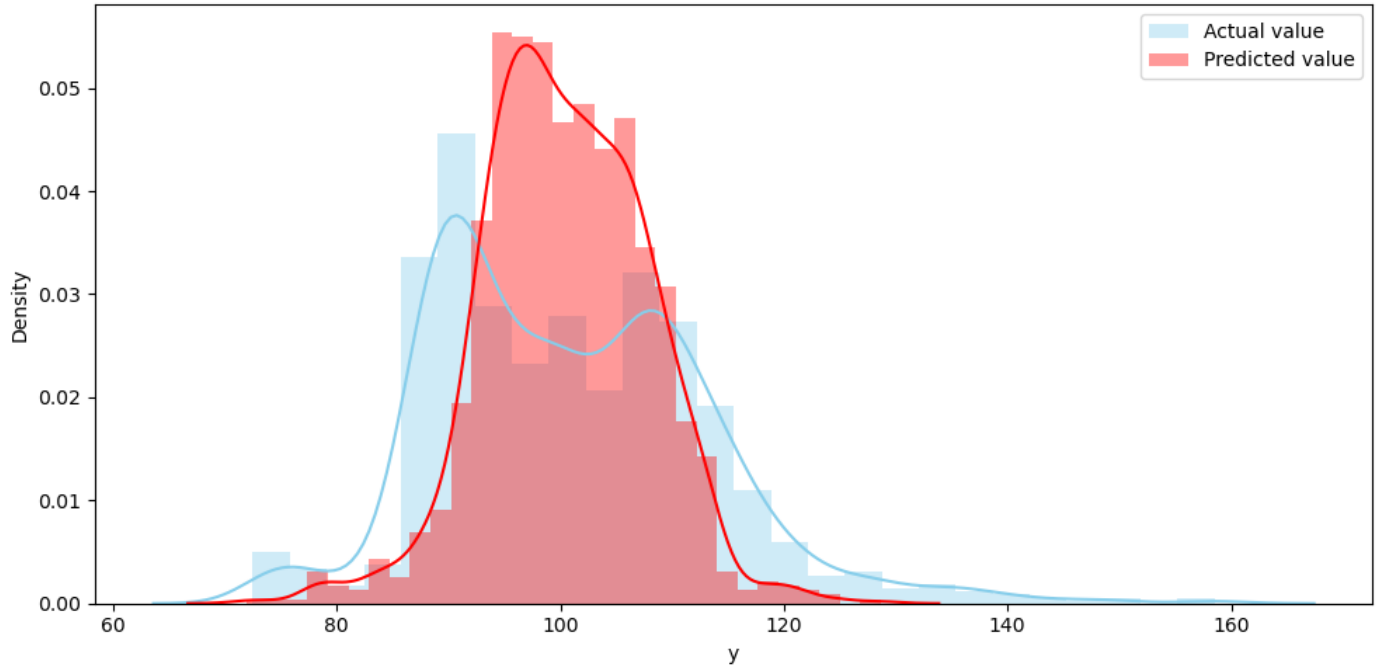
RMSE = 12.288794806074309

In [27]:
```
pred_test_y = model.predict(test_x)

plt.figure(figsize=(10,5))

sns.distplot(test_y[test_y<160], color="skyblue", label="Actual value")
sns.distplot(pred_test_y[pred_test_y<160] , color="red", label="Predicted value")
plt.legend()

plt.tight_layout()
```



k-fold Cross Validation using XGBoost

In [28]:
```
dmatrix_train = xgb.DMatrix(data=train_data_feature_tf,label=train_data_target)

params = {'objective':'reg:linear', 'colsample_bytree': 0.3, 'learning_rate': 0.3, 'max_

model_cv = xgb.cv(dtrain=dmatrix_train, params=params, nfold=3, num_boost_round=50, earl
                  metrics="rmse", as_pandas=True, seed=7)
model_cv.tail(4)
```

```
[17:02:33] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-030221e36e1a46bfb-1/xgboost/xgboost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[17:02:33] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-030221e36e1a46bfb-1/xgboost/xgboost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
[17:02:33] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-030221e36e1a46bfb-1/xgboost/xgboost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[28]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
31	8.935207	0.183408	11.060047	0.736219
32	8.880285	0.174860	11.044372	0.740167
33	8.849045	0.185327	11.049080	0.738351
34	8.792400	0.202135	11.043289	0.728256

Prediction on test data set using XGBoost

In [29]:
```
#Assigning feature to test dataset
test_data_feature = test_data.drop(['ID'],axis=1)
print(test_data_feature.shape)
```

```
(4209, 364)
```

In [30]:
```
#Describing object type variables in test dataset
test_data_feature.describe(include='object')
```

Out[30]:

	X0	X1	X2	X3	X4	X5	X6	X8

count	4209	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	4	32	12	25
top	ak	aa	as	c	d	v	g	e
freq	432	826	1658	1900	4203	246	1073	274

In [31]:
```
test_data_feature['X0'] = le.fit_transform(test_data_feature.X0)
test_data_feature['X1'] = le.fit_transform(test_data_feature.X1)
test_data_feature['X2'] = le.fit_transform(test_data_feature.X2)
test_data_feature['X3'] = le.fit_transform(test_data_feature.X3)
test_data_feature['X4'] = le.fit_transform(test_data_feature.X4)
test_data_feature['X5'] = le.fit_transform(test_data_feature.X5)
test_data_feature['X6'] = le.fit_transform(test_data_feature.X6)
test_data_feature['X8'] = le.fit_transform(test_data_feature.X8)
```

In [32]:
```
pca.fit(test_data_feature)
```

Out[32]:
```
PCA(n_components=0.95)
```

In [33]:
```
test_data_feature_tf = pca.fit_transform(test_data_feature)
print(test_data_feature_tf.shape)
```

```
(4209, 6)
```

In [34]:
```
test_pred = model.predict(test_data_feature_tf)
test_pred
```

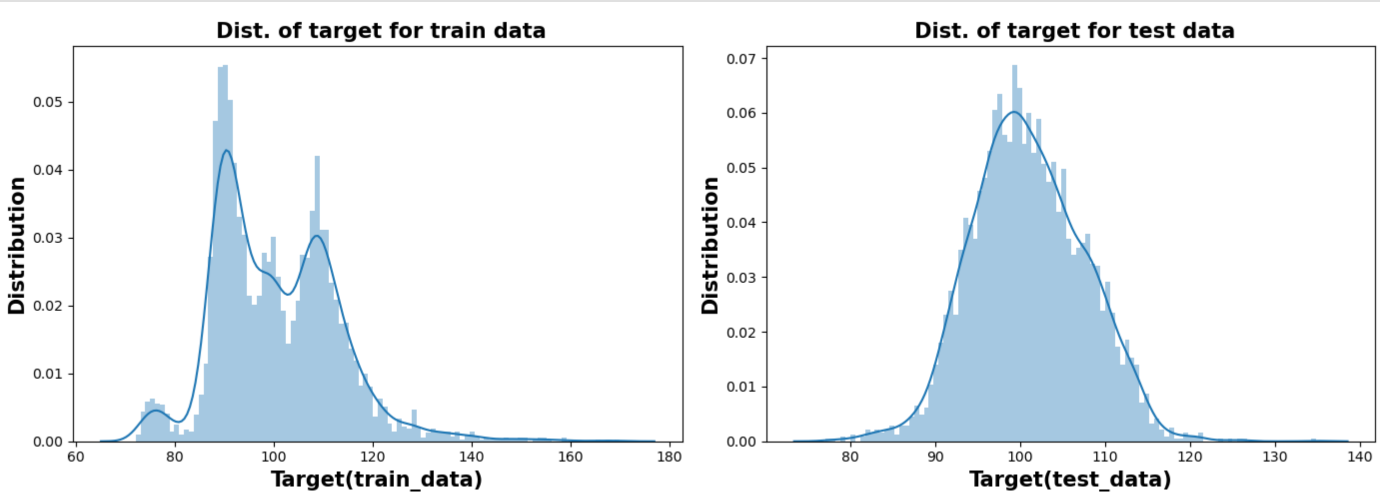
Out[34]:
```
array([ 86.12015 ,  92.929794,  98.74635 , ...,  92.836525, 118.76457 ,
        98.46741 ], dtype=float32)
```

In [35]:
```
fig, ax = plt.subplots(1,2, figsize=(14,5))

train_plot = sns.distplot(train_data_target[train_data_target<200], bins=100, kde=True,
train_plot.set_xlabel('Target(train_data)', weight='bold', size=15)
train_plot.set_ylabel('Distribution', weight='bold', size=15)
train_plot.set_title(' Dist. of target for train data', weight='bold', size=15)

test_plot = sns.distplot(test_pred[test_pred<200], bins=100, kde=True, ax=ax[1])
test_plot.set_xlabel('Target(test_data)', weight='bold', size=15)
test_plot.set_ylabel('Distribution', weight='bold', size=15)
test_plot.set_title(' Dist. of target for test data', weight='bold', size=15)

plt.tight_layout()
```



In [ ]: