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. Find the datasets here.

In [91]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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

In [92]:
```
#read train and test files
df_full_train = pd.read_csv('train.csv')
df_test = pd.read_csv("test.csv")
```

In [93]:
```
df_full_train.head()
```

Out[93]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0	1
1	6	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	0	1
2	7	76.26	az	w	n	c	d	x	j	x	0	0	0	0	0	0	0	1	0
3	9	80.62	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [94]:
```
df_full_train.shape,df_test.shape
```

Out[94]:
```
((4209, 378), (4209, 377))
```

In [95]:
```
df_full_train.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
```

The dataset has 4209 rows in train and 4209 rows in test dataset . The dataset has 378 columns where 8 columns are object datatype.The test datatype as envisaged has 377 columns. The target column is y

Task 1 : Remove columns with zero variance

In [96]:
```
numeric_columns = list(df_full_train.select_dtypes(exclude="object").co
lumns)
```

In [97]:
```
zero_var_columns = []
for col in numeric_columns:
  if df_full_train[col].var() == 0:
    zero_var_columns.append(col)
```

In [98]:
```
zero_var_columns
```

Out[98]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

In [99]:
```
#drop zero var columns from train and test dataset
df_full_train.drop(zero_var_columns,axis=1,inplace=True)
df_test.drop(zero_var_columns,axis=1,inplace=True)
```

In [100]:
```
#check shape of dataframes post dropping
df_full_train.shape,df_test.shape
```

Out[100]:
```
((4209, 366), (4209, 365))
```

Task 2 : Check for null and unique values for test and train sets

In [101]:
```
#null values of train dataset
train_null_cols =[]
for col in df_full_train.columns:
  if df_full_train[col].isnull().sum() != 0:
    train_null_cols.append(col)
```

In [102]:
```
len(train_null_cols) #how many columns have null val in train
```

Out[102]:
```
0
```

In [103]:
```
#null values of train dataset
test_null_cols =[]
for col in df_test.columns:
  if df_test[col].isnull().sum() !=0:
    test_null_cols.append(col)
```

In [104]:
```
len(test_null_cols)
```

Out[104]:
```
0
```

In [105]:
```
df_full_train.isnull().sum().sum(),df_test.isnull().sum().sum() #re che
ck for null values
```

Out[105]:
```
(0, 0)
```

There are no null values in both train and test dataset

In [106]:
```
df_full_train.head()
```

Out[106]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19
0	0	130.81	k	v	at	a	d	u	j	o	0	0	1	0	0	0	0	1	0
1	6	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	1	0
2	7	76.26	az	w	n	c	d	x	j	x	0	0	0	0	0	0	1	0	0

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19
3	9	80.62	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0

5 rows × 366 columns

In [107]:
```
#drop ID column from both train and test
df_full_train.drop('ID',axis=1,inplace=True)
df_test.drop('ID',axis=1,inplace=True)
```

In [108]:
```
# lets check for unique values in numeric columns
numeric_columns = list(df_full_train.select_dtypes(exclude="object").co
lumns)#redefine numeric cols
numeric_columns.remove('y')
unique_val_numeric =[]
for col in numeric_columns:
  unique_val_numeric.append(df_full_train[col].nunique())
```

In [109]:
```
print('Number of unique values in numeric columns',set(unique_val_numer
ic))
```

```
Number of unique values in numeric columns {2}
```

In [110]:
```
df_full_train['X10'].unique()
```

Out[110]:
```
array([0, 1])
```

All numeric columns in the train dataset has only 2 unique values 0 and 1

Lets have a look at object datatype columns

In [111]:
```
cat_columns = list(df_full_train.select_dtypes(include='object').column
s)
```

In [112]:
```
cat_columns
```

Out[112]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [113]:
```
# check for unique values in cat columns
for col in cat_columns:
  print(f'Number of Unique values in {col} are {df_full_train[col].nuni
que()}')
  print('\n\n')
```

```
Number of Unique values in X0 are 47



Number of Unique values in X1 are 27



Number of Unique values in X2 are 44



Number of Unique values in X3 are 7



Number of Unique values in X4 are 4



Number of Unique values in X5 are 29



Number of Unique values in X6 are 12



Number of Unique values in X8 are 25
```

Task Apply label encoder

In [114]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [115]:
```
for col in cat_columns:
  df_full_train[col] = le.fit_transform(df_full_train[col])
  #df_test[col] = le.transform(df_test[col])
```

In [116]:
```
for col in cat_columns:
  df_test[col] = le.fit_transform(df_test[col])
```

I have used fit_transform for test data as the only transform throws an error that unseen value 'av' in test data

In [117]:
```
df_full_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 365 entries, y to X385
dtypes: float64(1), int64(364)
memory usage: 11.7 MB
```

In [118]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 364 entries, X0 to X385
dtypes: int64(364)
memory usage: 11.7 MB
```

There are no cat values in train and test data

Dimensionality Reduction

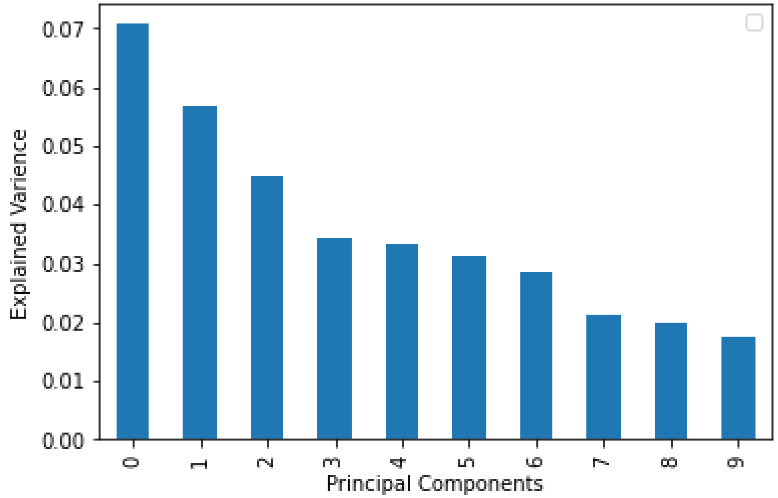
```

In [119]:
```
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
import xgboost as xg
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
```

The most popular technique for dimensionality reduction in machine learning is Principal Component Analysis, or PCA for short. This is a technique that comes from the field of linear algebra and can be used as a data preparation technique to create a projection of a dataset prior to fitting a model

In [121]:
```
train,val = train_test_split(df_full_train,test_size=0.2,random_state=4
2)
X_train = train.drop('y',axis=1)
y_train = train['y']

X_val = val.drop('y',axis=1)
y_val = val['y']
```

In [122]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
X_train_scaled = pd.DataFrame(scalar.fit_transform(X_train), columns=X_
train.columns) #scaling the feature
```

In [124]:
```
X_val_scaled = pd.DataFrame(scalar.transform(X_val), columns=X_train.co
lumns) #scaling the val data
```
```

In [130]:
```
pca =PCA(n_components=10) #applying pca
X_pca = pd.DataFrame(pca.fit_transform(X_train_scaled)) #fit and transf
orm on train data
X_pca.shape
```
Out[130]: (3367, 10)

In [131]:
```
X_val_pca = pd.DataFrame(pca.transform(X_val_scaled)) #applying pca on
 val data
```
In [133]:
```
X_val_pca.shape
```
Out[133]: (842, 10)

In [135]:
```
#applying scaling and val on test data
df_test_scaled = pd.DataFrame(scalar.transform(df_test), columns=df_tes
t.columns)
df_test_pca = pd.DataFrame(pca.transform(df_test_scaled))
```
In [137]:
```
pd.DataFrame(pca.explained_variance_ratio_).plot.bar()
plt.legend('')
plt.xlabel('Principal Components')
plt.ylabel('Explained Varience');
```



In [143]:
```
df_full_train.head()
```

Out[143]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X2
0	130.81	32	23	17	0	3	24	9	14	0	0	1	0	0	0	0	1	0	
1	88.53	32	21	19	4	3	28	11	14	0	0	0	0	0	0	0	1	0	
2	76.26	20	24	34	2	3	27	9	23	0	0	0	0	0	0	1	0	0	
3	80.62	20	21	34	5	3	27	11	4	0	0	0	0	0	0	0	0	0	
4	78.02	20	23	34	5	3	12	3	13	0	0	0	0	0	0	0	0	0	

5 rows × 365 columns

In [148]:
```
#Implementing cross validation

k = 5
kf = KFold(n_splits=k, random_state=None)
xgb_r = xg.XGBRegressor(n_estimators = 10, seed = 123)

score = []
```

```
df_full_train_scaled = pd.DataFrame(scalar.fit_transform(df_full_train
), columns=df_full_train.columns)
X = df_full_train_scaled.drop('y',axis=1)
y = df_full_train_scaled['y']
X = pd.DataFrame(pca.fit_transform(X)) #fit and transform on train data

for train_index , test_index in kf.split(X):
    X_train , X_val = X.iloc[train_index,:],X.iloc[test_index,:]
    y_train , y_val = y[train_index] , y[test_index]

    model.fit(X_train,y_train)
    pred_values = model.predict(X_val)

    mse = MSE(pred_values , y_val)
    score.append(mse)
    print('Metric of each fold - {}'.format(mse))

avg_score = np.mean(score)
print('Avg MSE : {}'.format(avg_score))
```

```
[17:45:59] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Metric of each fold - 0.7496670198095091
[17:46:00] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Metric of each fold - 0.9974039587776969
[17:46:00] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Metric of each fold - 0.7074498549312433
[17:46:00] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Metric of each fold - 0.7437138626935256
[17:46:00] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Metric of each fold - 0.6536280680717771
Avg MSE : 0.7703725528567504
```

In [149]:

```
xgb_r.fit(X,y) #fit on model
```

```
[17:46:53] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
```

Out[149]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=
0,
             max_depth=3, min_child_weight=1, missing=None, n_estimator
s=10,
             n_jobs=1, nthread=None, objective='reg:linear', random_sta
te=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=123,
             silent=None, subsample=1, verbosity=1)
```

In [151]:

```
preds[:5] # first 5 predictions
```

Out[151]:

```
array([-0.943967  , -0.01065439, -0.943967  , -0.943967  ,  0.3770035
],
      dtype=float32)
```

In [ ]:

In [ ]: