Machine Learning Project

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 MercedesBenz'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:

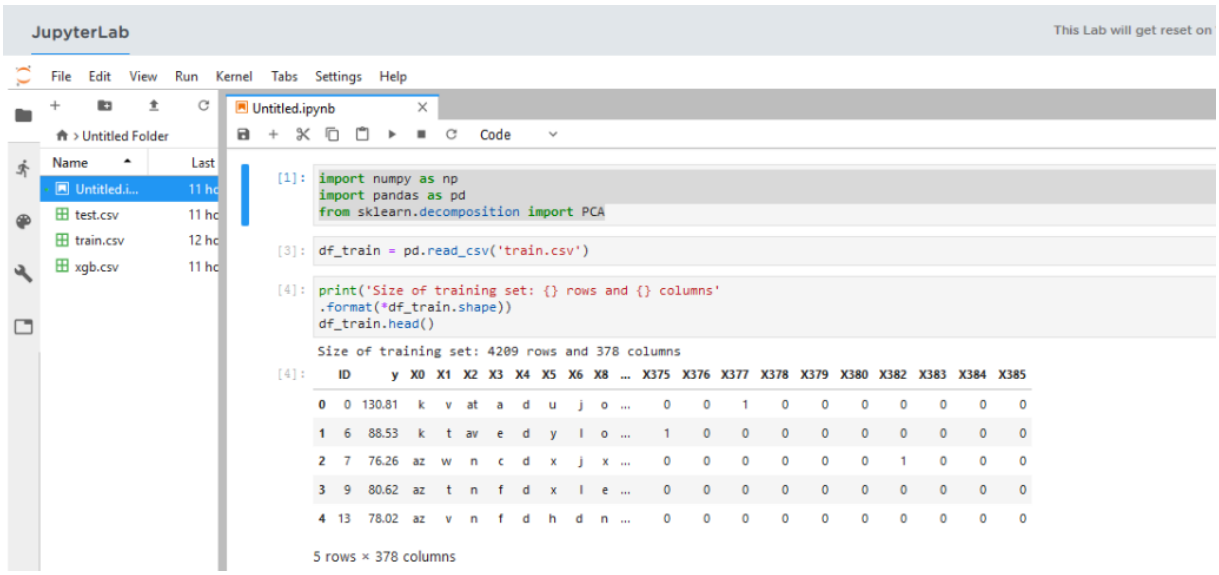
- If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
- Check for null and unique values for test and train sets.
- Apply label encoder.
- Perform dimensionality reduction.
- Predict your test_df values using XGBoost.

Code with Outputs

```
import numpy as np  import
pandas as pd
from sklearn.decomposition import PCA

df_train = pd.read_csv('train.csv')
```

```
print('Size of training set: {} rows and {} columns'
.format(*df_train.shape))

 df_train.head()
```

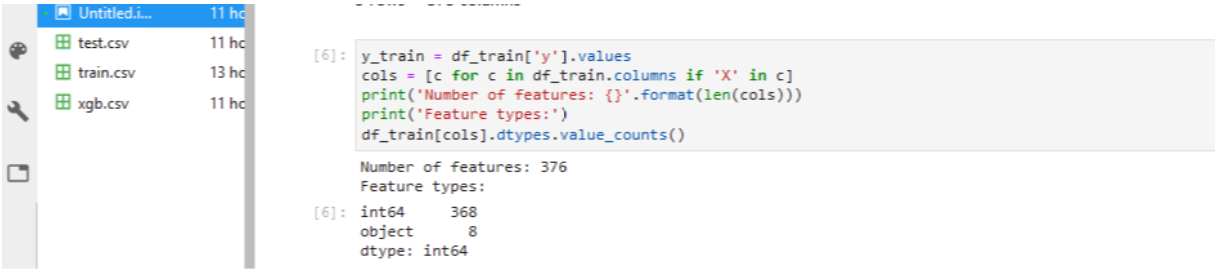
Output



Code

```
 y_train = df_train['y'].values


cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))
print('Feature types:')
df_train[cols].dtypes.value_counts()
```

Output



Code

Count the data in each of the columns
counts = [[], [], []]

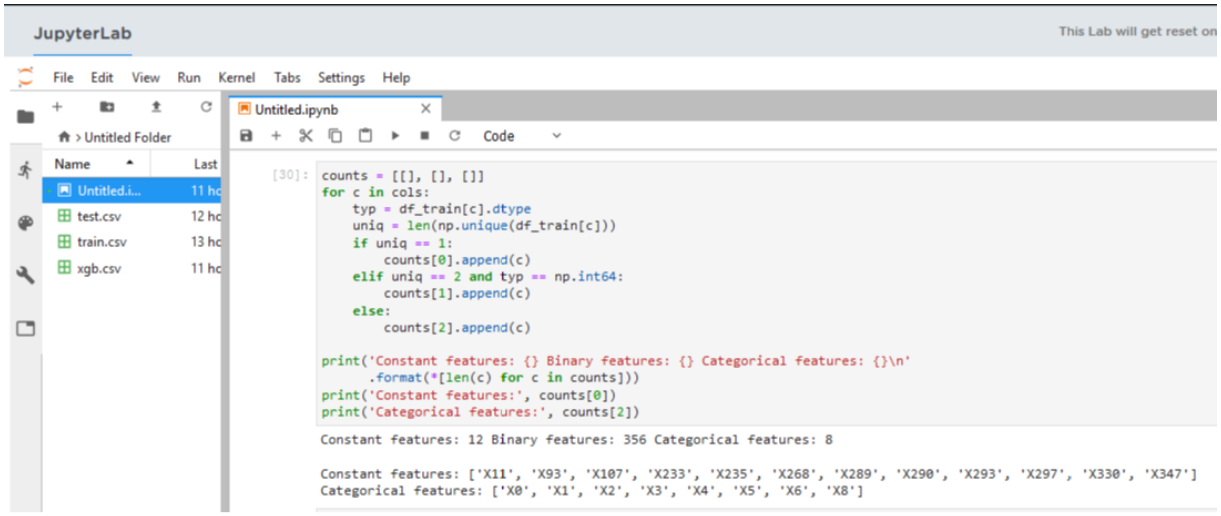
for c in cols: typ = df_train[c].dtype

uniq = len(np.unique(df_train[c]))
if uniq == 1:
counts[0].append(c)
elif uniq == 2 and
typ == np.int64:
counts[1].append(c)
else:
counts[2].append(c) print('Constant features: {} Binary features: {}

Categorical features: {}\n'

.format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])

Output



Code

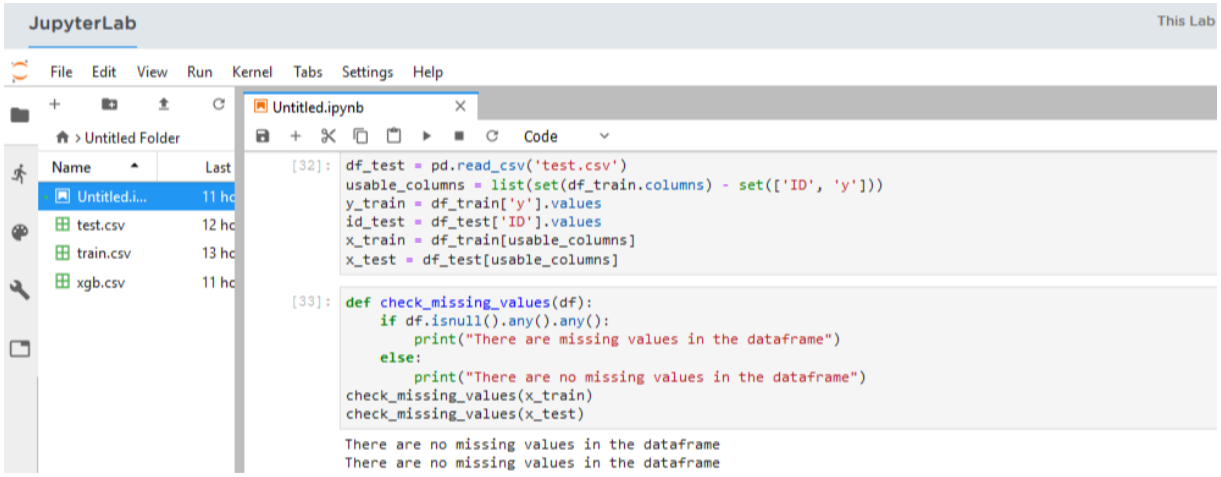
df_test = pd.read_csv('test.csv')

```
usable_columns = list(set(df_train.columns) - set(['ID',
'y']))

y_train = df_train['y'].values  id_test = df_test['ID'].values
x_train = df_train[usable_columns]

x_test = df_test[usable_columns]
```

Check for null and unique values for test and train sets

```
def check_missing_values(df):

if df.isnull().any().any():

print("There are missing values in the dataframe")
else:  print("There are no missing values in the
dataframe")

check_missing_values(x_train)
check_missing_values(x_test)
```

Output



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

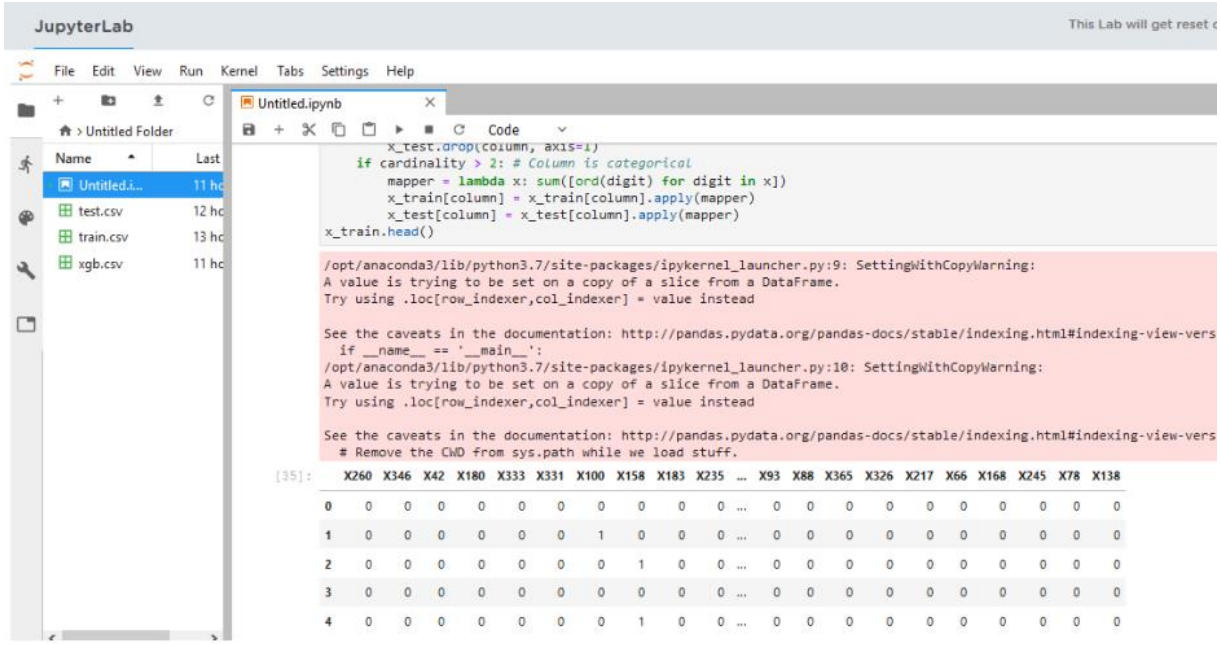
Apply Label Encoder

Code

```
for column in usable_columns:  cardinality =
len(np.unique(x_train[column]))                    if
cardinality == 1:
```

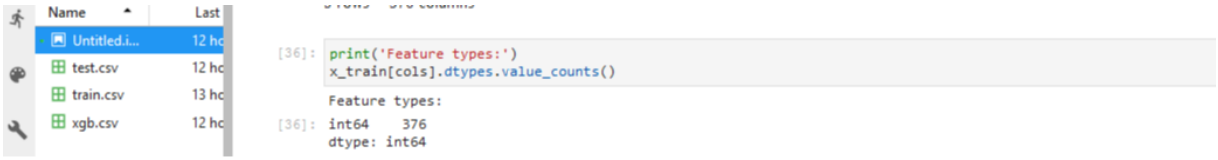
```
x_train.drop(column, axis=1)
x_test.drop(column,   axis=1)
if cardinality > 2:
mapper = lambda x: sum([ord(digit) for digit in x])
x_train[column] = x_train[column].apply(mapper)
x_test[column] = x_test[column].apply(mapper)
x_train.head()
```

Output



Code

```
print('Feature types:')
x_train[cols].dtypes.value_counts()
```



Code

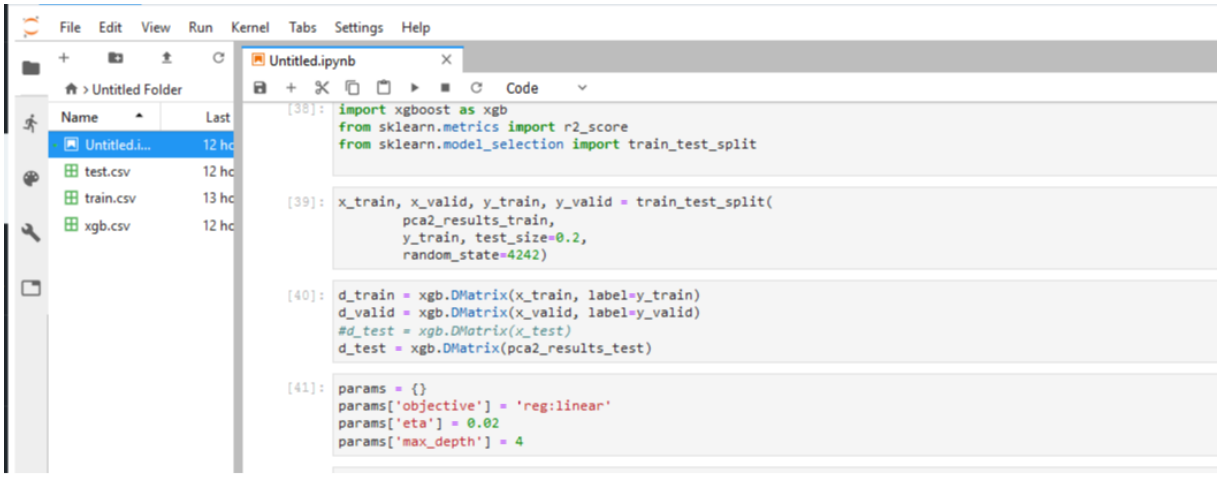
Perform dimensionality reduction

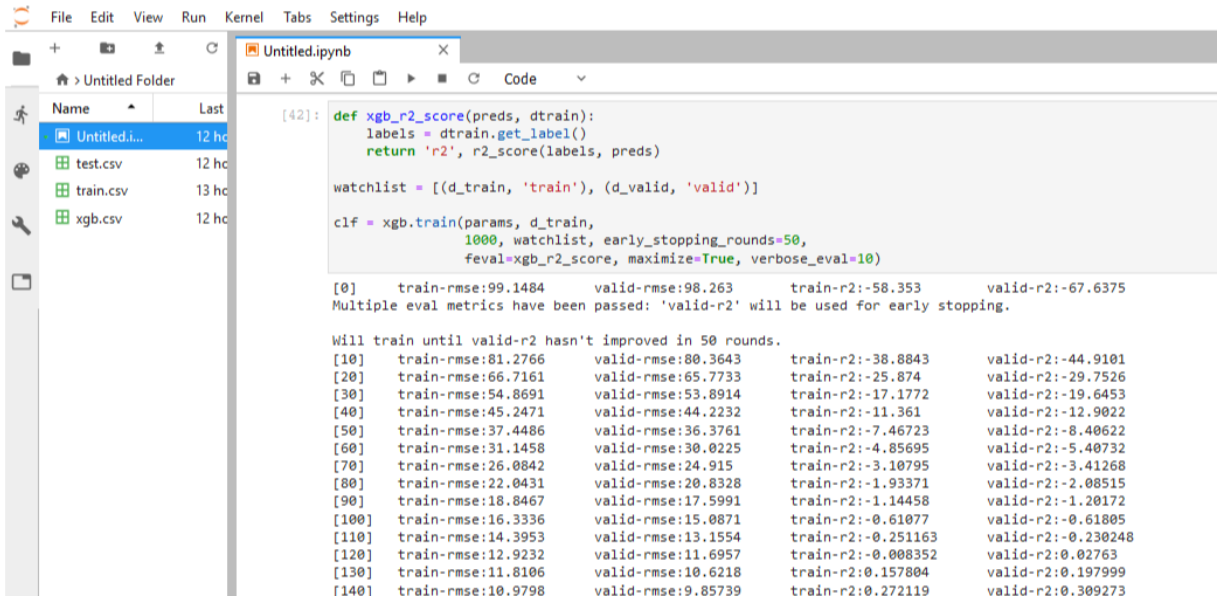
```
n_comp = 12  pca = PCA(n_components=n_comp,
random_state=420)

pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
```

Code

```
import xgboost as xgb  from sklearn.metrics
import r2_score  from sklearn.model_selection
import train_test_split

 x_train, x_valid, y_train, y_valid = train_test_split(
pca2_results_train,
y_train, test_size=0.2,  random_state=4242)


d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)

d_test = xgb.DMatrix(pca2_results_test)
params= {}


params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4  def
xgb_r2_score(preds, dtrain):
labels = dtrain.get_label()  return
'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]
clf = xgb.train(params, d_train,

1000, watchlist, early_stopping_rounds=50,  feval=xgb_r2_score,
maximize=True, verbose_eval=10)
```

Output





Code

Predict your test_df values using XGBoost

p_test = clf.predict(d_test)

sub = pd.DataFrame() sub['ID'] = id_test sub['y'] = p_test

```
sub.to_csv('xgb.csv', index=False)
sub.head()
```

Output