

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 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.

Code with Outputs

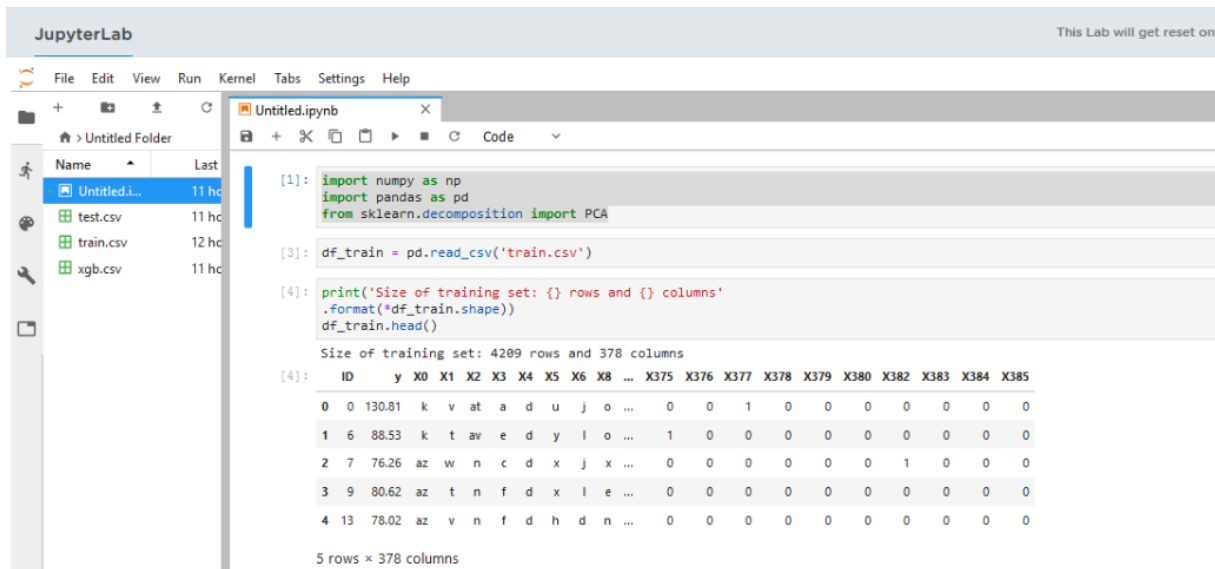
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
```

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

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

```
df_train.head()
```

Output



The screenshot shows a JupyterLab environment with a file explorer on the left containing 'test.csv', 'train.csv', and 'xgb.csv'. The main area displays a code cell with the following code:

```
[1]: import numpy as np
import pandas as pd
from sklearn.decomposition import PCA

[3]: df_train = pd.read_csv('train.csv')

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

The output of the code cell shows the size of the training set and the first five rows of the data:

```
Size of training set: 4209 rows and 378 columns
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	a	t	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0
1	6	88.53	k	t	a	v	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	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	0

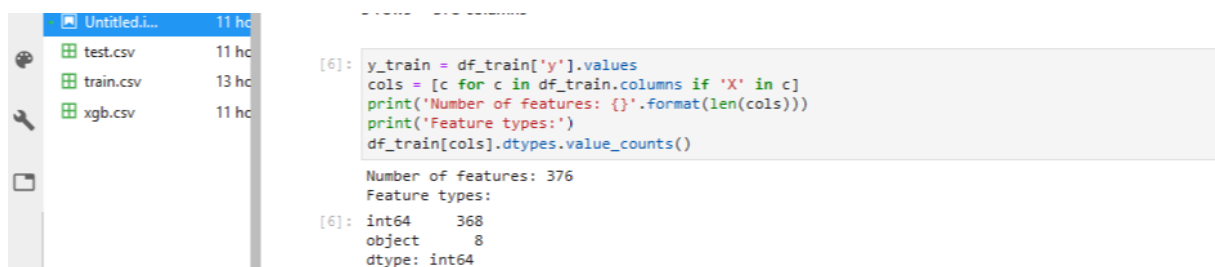
5 rows × 378 columns

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



The screenshot shows a JupyterLab environment with a file explorer on the left containing 'test.csv', 'train.csv', and 'xgb.csv'. The main area displays a code cell with the following code:

```
[6]: 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()
```

The output of the code cell shows the number of features and the feature types:

```
Number of features: 376
Feature types:
```

```
[6]: int64      368
object        8
dtype: int64
```

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: { }
```

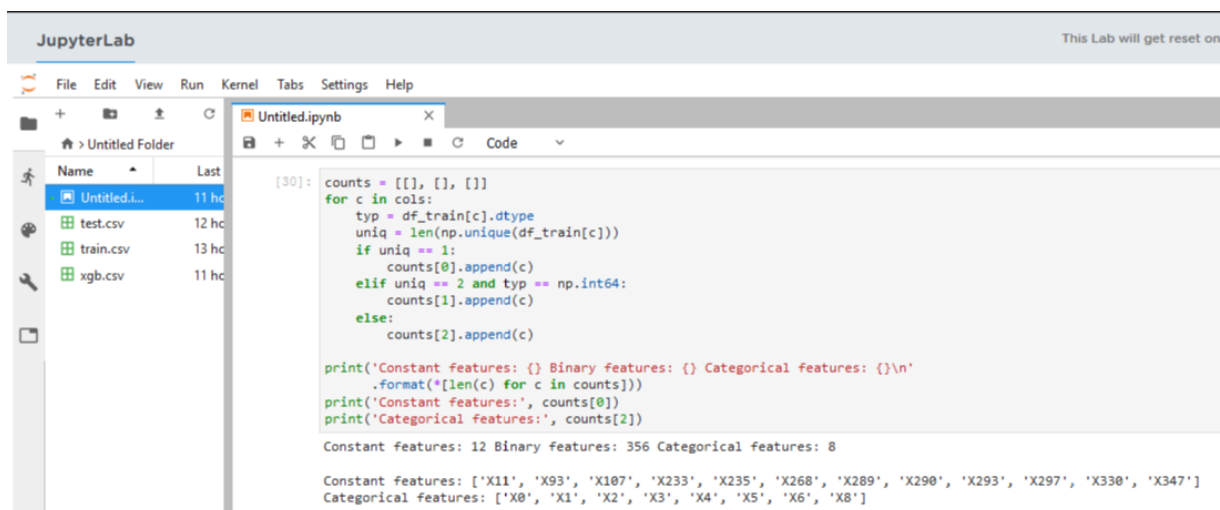
```
Categorical features: { }\n'
```

```
.format(*[len(c) for c in counts]))
```

```
print('Constant features:', counts[0])
```

```
print('Categorical features:', counts[2])
```

Output



The screenshot shows a JupyterLab window with a file explorer on the left containing 'test.csv', 'train.csv', and 'xgb.csv'. The main area displays a code cell with the following Python code:

```
[30]: 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])
```

The output of the code is:

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

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():
```

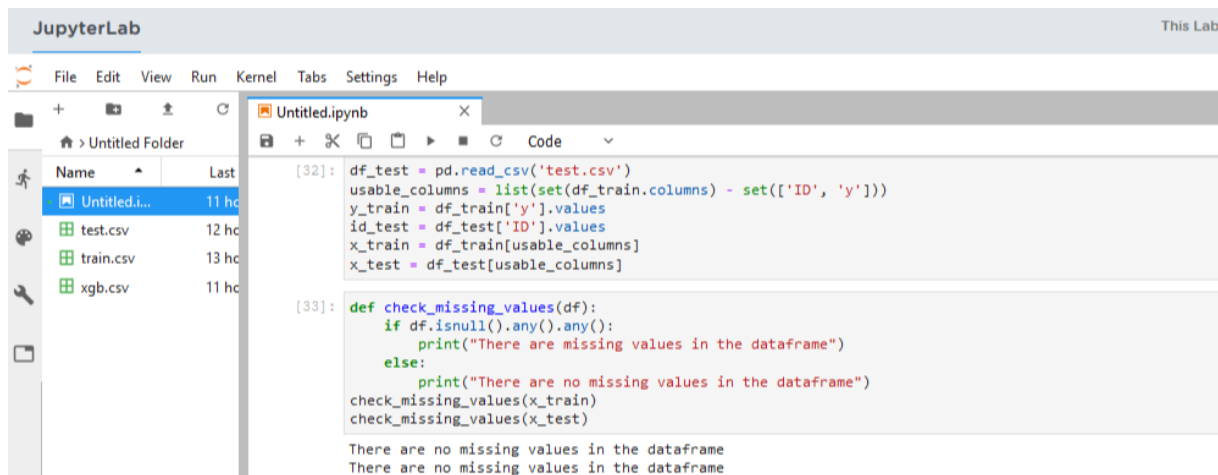
```
print("There are missing values in the dataframe")
```

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

```
check_missing_values(x_train)
```

```
check_missing_values(x_test)
```

Output



The screenshot shows a JupyterLab window with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'Untitled Folder' containing files 'test.csv', 'train.csv', and 'xgb.csv'. The code editor shows two code cells. The first cell (index 32) contains the code to load the data and split it into training and testing sets. The second cell (index 33) contains the code to define and call the 'check_missing_values' function. The output of the second cell shows two lines of text: 'There are no missing values in the dataframe'.

```
JupyterLab This Lab  
File Edit View Run Kernel Tabs Settings Help  
+ + + + + Code  
Untitled.ipynb  
[32]: 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]  
  
[33]: 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)  
  
There are no missing values in the dataframe  
There are no missing values in the dataframe
```

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

Apply Label Encoder

Code

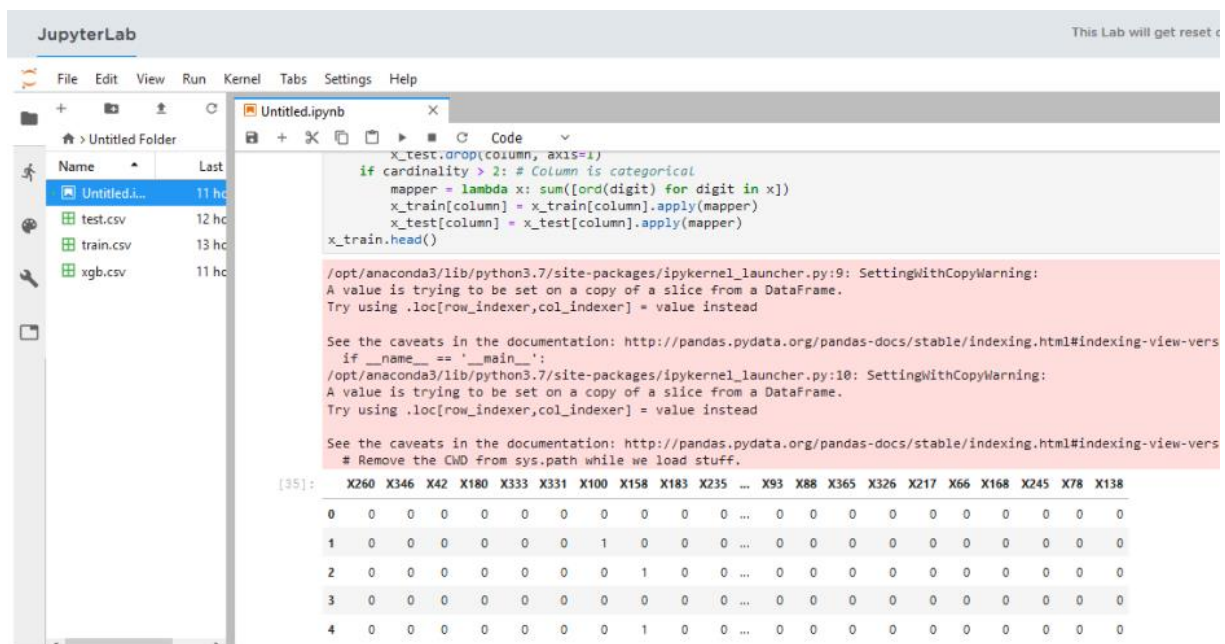
```
for column in usable_columns: cardinality =  
len(np.unique(x_train[column])) if  
cardinality == 1:
```

```

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



The screenshot shows the JupyterLab interface. On the left is a file explorer with a folder named 'Untitled Folder' containing files 'test.csv', 'train.csv', and 'xgb.csv'. The main area displays a code cell with the following Python code:

```

x_test.drop(column, axis=1)
if cardinality > 2: # Column is categorical
    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()

```

Below the code, there are two warning messages from IPyKernel:

```

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-vers
if __name__ == '__main__':
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-vers
# Remove the CWD from sys.path while we load stuff.

```

The output of the code cell is a DataFrame head:

```

[35]:
   X260  X346  X42  X180  X333  X331  X100  X158  X183  X235  ...  X93  X88  X365  X326  X217  X66  X168  X245  X78  X138
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      1      0      0      0  ...  0      0      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0      1      0      0  ...  0      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0      0      0  ...  0      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      1      0      0  ...  0      0      0      0      0      0      0      0      0

```

Code

```

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

```



The screenshot shows the JupyterLab interface. On the left is a file explorer with a folder named 'Untitled Folder' containing files 'test.csv', 'train.csv', and 'xgb.csv'. The main area displays a code cell with the following Python code:

```

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

```

Below the code, there are two warning messages from IPyKernel:

```

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-vers
if __name__ == '__main__':
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-vers
# Remove the CWD from sys.path while we load stuff.

```

The output of the code cell is:

```

[36]:
Feature types:
[36]: int64    376
      dtype: int64

```

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)
```

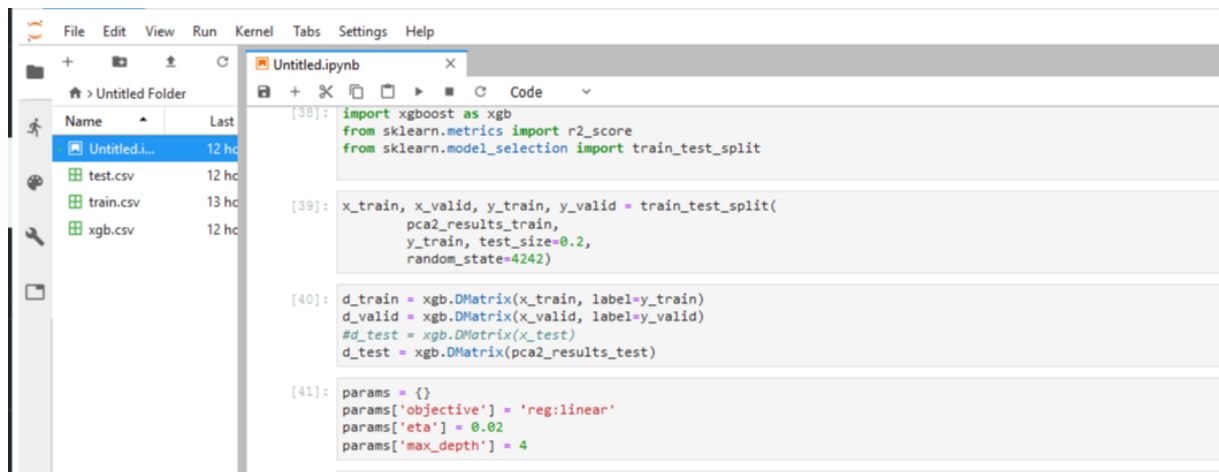
```
d_test = xgb.DMatrix(pca2_results_test)  
params= { }
```

```
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

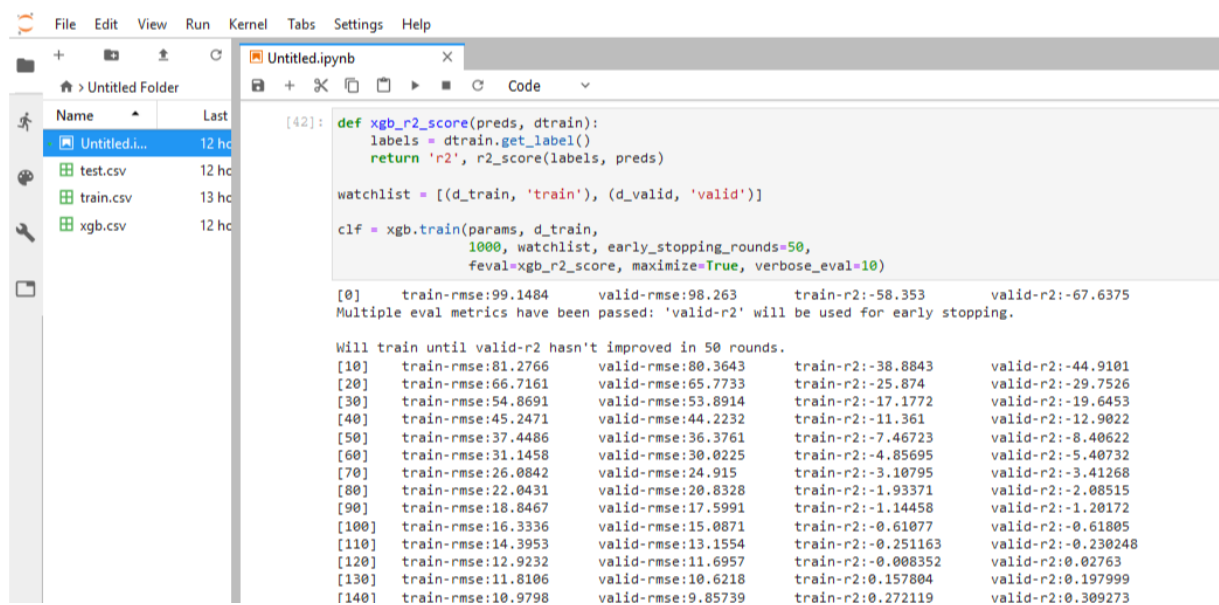


```
[38]: import xgboost as xgb
      from sklearn.metrics import r2_score
      from sklearn.model_selection import train_test_split

[39]: x_train, x_valid, y_train, y_valid = train_test_split(
      pca2_results_train,
      y_train, test_size=0.2,
      random_state=4242)

[40]: d_train = xgb.DMatrix(x_train, label=y_train)
      d_valid = xgb.DMatrix(x_valid, label=y_valid)
      #d_test = xgb.DMatrix(x_test)
      d_test = xgb.DMatrix(pca2_results_test)

[41]: params = {}
      params['objective'] = 'reg:linear'
      params['eta'] = 0.02
      params['max_depth'] = 4
```



```
[42]: 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)

[0]    train-rmse:99.1484    valid-rmse:98.263    train-r2:-58.353    valid-r2:-67.6375
Multiple eval metrics have been passed: 'valid-r2' will be used for early stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]    train-rmse:81.2766    valid-rmse:80.3643    train-r2:-38.8843    valid-r2:-44.9101
[20]    train-rmse:66.7161    valid-rmse:65.7733    train-r2:-25.874    valid-r2:-29.7526
[30]    train-rmse:54.8691    valid-rmse:53.8914    train-r2:-17.1772    valid-r2:-19.6453
[40]    train-rmse:45.2471    valid-rmse:44.2232    train-r2:-11.361    valid-r2:-12.9022
[50]    train-rmse:37.4486    valid-rmse:36.3761    train-r2:-7.46723    valid-r2:-8.40622
[60]    train-rmse:31.1458    valid-rmse:30.0225    train-r2:-4.85695    valid-r2:-5.40732
[70]    train-rmse:26.0842    valid-rmse:24.915    train-r2:-3.10795    valid-r2:-3.41268
[80]    train-rmse:22.0431    valid-rmse:20.8328    train-r2:-1.93371    valid-r2:-2.08515
[90]    train-rmse:18.8467    valid-rmse:17.5991    train-r2:-1.14458    valid-r2:-1.20172
[100]   train-rmse:16.3336    valid-rmse:15.0871    train-r2:-0.61077    valid-r2:-0.61805
[110]   train-rmse:14.3953    valid-rmse:13.1554    train-r2:-0.251163    valid-r2:-0.230248
[120]   train-rmse:12.9232    valid-rmse:11.6957    train-r2:-0.008352    valid-r2:0.02763
[130]   train-rmse:11.8106    valid-rmse:10.6218    train-r2:0.157804    valid-r2:0.197999
[140]   train-rmse:10.9798    valid-rmse:9.85739    train-r2:0.272119    valid-r2:0.309273
```

Code

Predict your test_df values using XGBoost

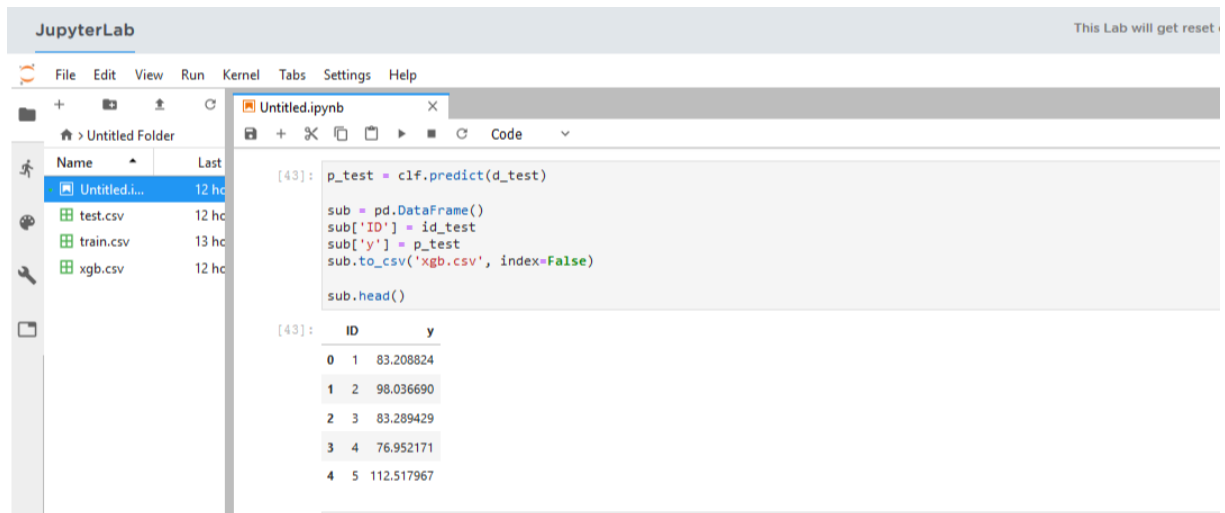
```
p_test = clf.predict(d_test)
```

```
sub = pd.DataFrame() sub['ID'] =
```

```
id_test sub['y'] = p_test
```

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

Output



The image shows a JupyterLab interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'Untitled Folder' containing files 'test.csv', 'train.csv', and 'xgb.csv'. The code editor shows the following code:

```
[43]: 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()
```

The output of the code is a DataFrame with 5 rows and 2 columns, 'ID' and 'y'.

	ID	y
0	1	83.208824
1	2	98.036690
2	3	83.289429
3	4	76.952171
4	5	112.517967