

Project No. 1: Mercedes-Benz Greener Manufacturing

MainCode

```
# Step1: Import the required libraries
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA

# Step2: Read the data from train.csv
df_train = pd.read_csv('train.csv')
print('Size of training set: { } rows and { } columns'
      .format(*df_train.shape))
df_train.head()

# Step3: Collect the Y values into an array
y_train = df_train['y'].values

# Step4: Understand the data types we have
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()

# Step5: 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])

# Step6: Read the test.csv data
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]

# Step7: 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)
```

```
# Step8: If for any column(s), the variance is equal to zero,
```

```
# then you need to remove those variable(s) and Apply label encoder
```

```
for column in usable_columns:
```

```
    cardinality = len(np.unique(x_train[column]))
```

```
    if cardinality == 1:
```

```
        x_train.drop(column, axis=1) # Column with only one
```

```
        # value is useless so we drop it
```

```
        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()
```

```
# Step9: Make sure the data is now changed into numericals
```

```
print('Feature types:')
```

```
x_train[cols].dtypes.value_counts()
```

```
# Step10: 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)
```

```
# Step11: Training using xgboost
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(x_test)
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)

# Step12: 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()

#####

'''
                        End
'''

```

Code Snippet followed a Screenshot of the Output

Q1. Read the data from train.csv and Collect the Y values into an array

```

# Step2: Read the data from train.csv

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

# let us understand the data
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))

# print few rows and see how the data looks like
df_train.head()

```

Step3: Collect the Y values into an array

separete the y from the data as we will use this to learn as

the prediction output

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

The screenshot displays a Jupyter Notebook interface. At the top, the file path is `F:\ML\Simplelearn\Machine Learning\MyProject\P1`. Below this is the 'Variable explorer' panel, which shows two variables: `df_train` (DataFrame, Size: (4209, 378)) and `y_train` (float64, Size: (4209,)). The 'Value' column for `df_train` lists column names: ID, y, X0, X1, X2, X3, X4, X5, X6, X8, X10, X11, X12, X1. The 'Value' column for `y_train` shows a sample of values: [130.81 88.53 76.26 ... 109.22 87.48 110.85]. Below the variable explorer is the 'IPython console' panel, which contains the following code:

```
In [21]: import numpy as np
...: # data processing, CSV file I/O (e.g. pd.read_csv)
...: import pandas as pd
...: # for dimensionality reduction
...: from sklearn.decomposition import PCA

In [22]: df_train = pd.read_csv('train.csv')
...: # let us understand the data
...: print('Size of training set: {} rows and {} columns'
...:       .format(*df_train.shape))
...: # print few rows and see how the data looks like
...: df_train.head()
...:
...: # Step3: Collect the Y values into an array
...:
...: # separete the y from the data as we will use this to learn as
...: # the prediction output
...: y_train = df_train['y'].values
Size of training set: 4209 rows and 378 columns

In [23]:
```

The bottom status bar shows 'Permissions: RW', 'End-of-lines: CRLF', 'Encoding: ASCII', 'Line: 27', 'Column: 31', and 'Memory: 81%'.

Q2. Understand the data types we have and Count the data in each of the columns

Step4: Understand the data types we have

```
# iterate through all the columns which has X in the name of the column
```

```
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()
```

```
# Step5: 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])

```

The screenshot shows a Jupyter Notebook interface with two main panels. The top panel is the 'Variable explorer' which displays the following data:

Name	Type	Size	Value
c	str	1	X385
cols	list	376	['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10', 'X11', ...]
counts	list	3	[['X11', 'X93', 'X107', 'X233', 'X235', ...], ['X10', 'X12', 'X13', 'X ...]]

The bottom panel is the 'IPython console' showing the following code and output:

```

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

```

Output:

```

Number of features: 376
Feature types:
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']

In [24]:

```

The bottom status bar shows: Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 29 | Column: 1 | Memory: 81 %

Q3. Read the test.csv data and Check for null and unique values for test and train sets

Step6: Read the test.csv data

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



```
# remove columns ID and Y from the data as they are not used for learning
```

```
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]
```

```
# Step7: 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)
```

The screenshot shows a Jupyter Notebook interface. At the top, the 'Variable explorer' tab is active, displaying a table of variables:

Name	Type	Size	Value
x_test	DataFrame	(4209, 376)	Column names: X138, X274, X290, X310, X359, X353, X95, X74, X156, X333 ...
x_train	DataFrame	(4209, 376)	Column names: X138, X274, X290, X310, X359, X353, X95, X74, X156, X333 ...
y_train	float64	(4209,)	[130.81 88.53 76.26 ... 109.22 87.48 110.85]

Below the variable explorer is the 'IPython console' tab, which contains the following code and output:

```
In [24]: df_test = pd.read_csv('test.csv')
...:
...: # remove columns ID and Y from the data as they are not used for learning
...: 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]
...:
...: # Step7: 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)
There are no missing values in the dataframe
There are no missing values in the dataframe

In [25]:
```

At the bottom of the console, the status bar shows: Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 76 | Column: 29 | Memory: 81 %

Q4. If for any column(s), the variance is equal to zero, then you need to remove those variable(s) and Apply label encoder

Step8: If for any column(s), the variance is equal to zero,

then you need to remove those variable(s).

Apply label encoder

for column in usable_columns:

 cardinality = len(np.unique(x_train[column]))

 if cardinality == 1:

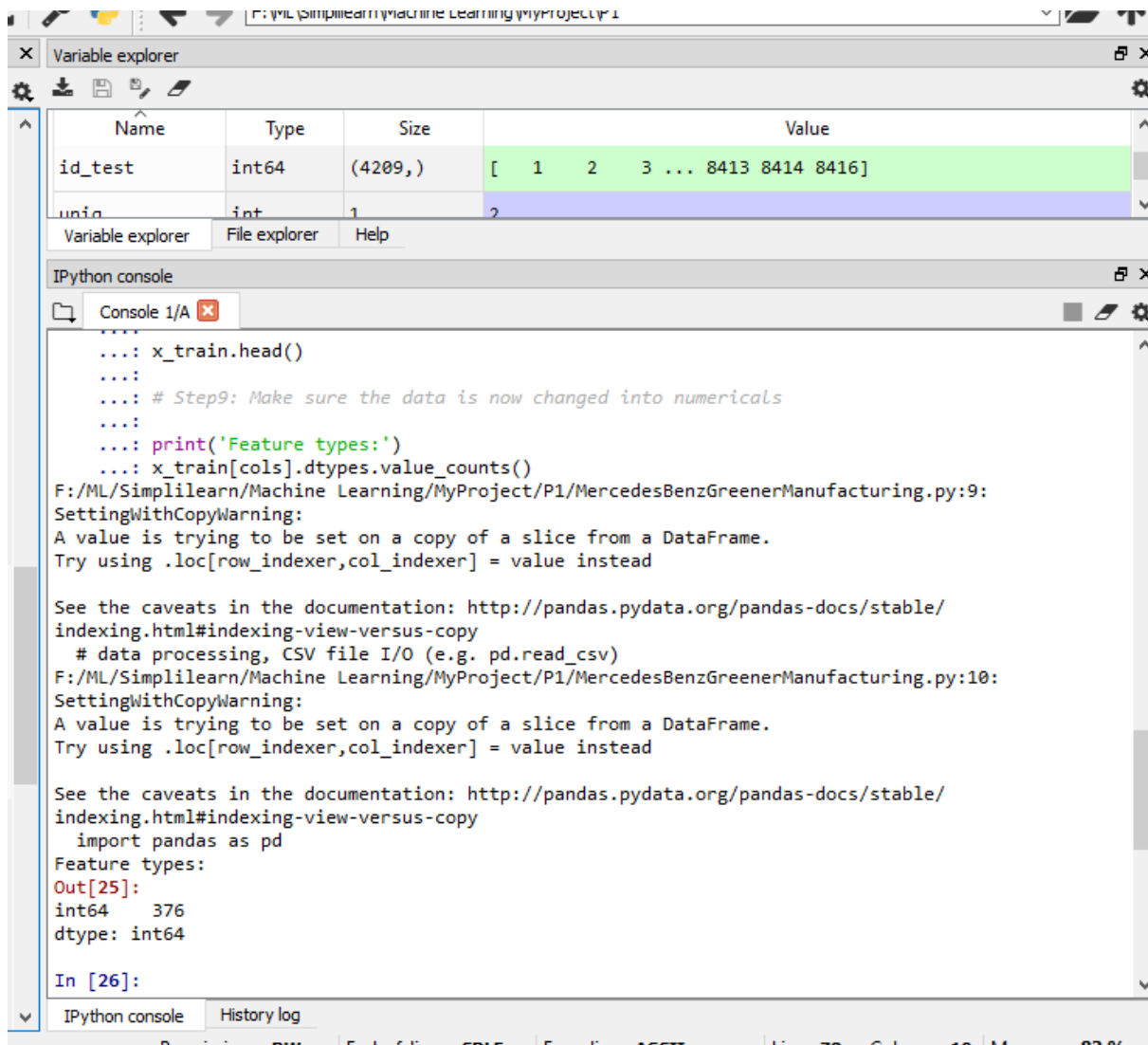
```
x_train.drop(column, axis=1) # Column with only one
# value is useless so we drop it
x_test.drop(column, axis=1)

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()

# Step9: Make sure the data is now changed into numericals

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



Q5. Perform dimensionality reduction

Step10: Perform dimensionality reduction

Linear dimensionality reduction using Singular Value Decomposition of

the data to project it to a lower dimensional space.

```
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)
```

The screenshot shows a Jupyter Notebook interface. The top panel is the 'Variable explorer' showing the following variables:

Variable	Type	Value
n_comp	int	12
pca2_results_test	float64	(4209, 12)
pca2_results_train	float64	(4209, 12)
uniq	int	1
usable_columns	list	376
x_test	DataFrame	(4209, 376)

The bottom panel is the 'IPython console' showing the following code:

```
In [26]: 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)

In [27]:
```

Q6. Training using xgboost

Step11: Training using xgboost

```
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(x_test)

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)
```

The screenshot shows a Jupyter Notebook interface. At the top, the 'Variable explorer' tab is active, displaying a table of variables:

Name	Type	Size	Value
x_test	DataFrame	(4209, 376)	Column names: X138, X274, X290, X310, X359, X353, X95, X74, X156, X333 ...
x_train	float64	(3367, 12)	[[-4.20001451e+01 -1.21246097e+00 -2.89822575e+01 ... 2.93516617e-01 ...
x_valid	float64	(842, 12)	[[9.17571735e+01 3.32703342e+01 -3.19443589e+01 ... -1.60251913e+00 ...
y_train	float64	(3367,)	[88.25 88.51 93.8 ... 113.64 91.99 117.3]
y_valid	float64	(842,)	[75.85 108.08 96.98 ... 88.83 96.62 85.49]

Below the variable explorer, the 'IPython console' tab is active, showing the following code and output:

```

... 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)
[12:22:29] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/
regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[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:81.2765    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:54.8696    valid-rmse:53.8894    train-r2:-17.1775    valid-r2:-19.6438

```

Q7. 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()
```

Variable explorer

sub	DataFrame	(4209, 2)	Column names: ID, y
uniq	int	1	2
usable_columns	list	376	['X138', 'X274', 'X290', 'X310', 'X359', 'X353', 'X95', 'X74', 'X156', ...]

File explorer Help

IPython console

Console 1/A

Stopping. Best iteration:
[495] train-rmse:8.01893 valid-rmse:8.2609 train-r2:0.611757 valid-r2:0.514894

```
In [28]: 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()
```

Out[28]:

ID	y
0 1	82.836075
1 2	97.907310
2 3	82.805481
3 4	76.751198
4 5	113.063164

In [29]:

sub - DataFrame

Index	ID	y
0	1	82.836075
1	2	97.90731
2	3	82.80548
3	4	76.7512
4	5	113.06316
5	8	91.48998
6	10	99.38939
7	11	95.47029

IPython console History log

Permissions: RW End-of-lines: CRLF

End