Write up for “Mercedes-Benz Greener Manufacturing” Project

Problem statement: ##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:

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

2. Check for null and unique values for test and train sets.

3. Apply label encoder.

4. Perform dimensionality reduction.

5. Predict your test\_df values using XGBoost.

Created By – Selpa Singh

Solution:

Here for the problem statement we need to perform 5 actions, so Let’s start with first action.

**Action 1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s).**

In order to analyze the dataset and remove any column with 0 variance , first we need to import the dataset and study it.

We have 2 separate dataset I.e train and test dataset. I have renamed it to benz\_train.csv and benz\_test.csv for clear understanding.

Before we start with action1 we need to analyze the dataset , and we will follow below mentioned steps for our analysis

Step 1 – Import dataset for train and test using panda – read\_csv

*#before we start with our analysis and predisction let's import required libraries and load dataset,*

*#we will include more libraries when needed,*

*#for now based on problem statement I can say we would need pandas, numpy, sklearn train test split, xgboost*

*import pandas as pd*

*import numpy as np*

*from sklearn.model\_selection import train\_test\_split*

*import xgboost as xgb*

*import warnings*

*warnings.filterwarnings('ignore')*

*#import dataset for training data*

*#i have renamed the dataset with benz prefix and it is in my jupyter folder names dataset\_files,i can import directly from there*

*train\_data = pd.read\_csv('dataset\_files/benz\_train.csv')*

*Output:*

Step 2 – Lets see the contents of dataset by using head() method, and dimension of the dataset by using shape

train\_data.head()

Output:

|  | **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 | at | a | d | u | j | o | ... | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 6 | 88.53 | k | t | av | e | d | y | l | o | ... | 1 | 0 | 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

train\_data.shape

Output:

(4209, 378)

Step 3 – Lets check range, no of columns, data types in the data by using info() method

Since the Head method does not show all the data, we will set max columns/rows limit set by using set\_option

pd.set\_option('display.max\_columns', 378)

pd.set\_option('display.max\_rows', 4209)

train\_data.info()

Output:

<class 'pandas.core.frame.DataFrame'>

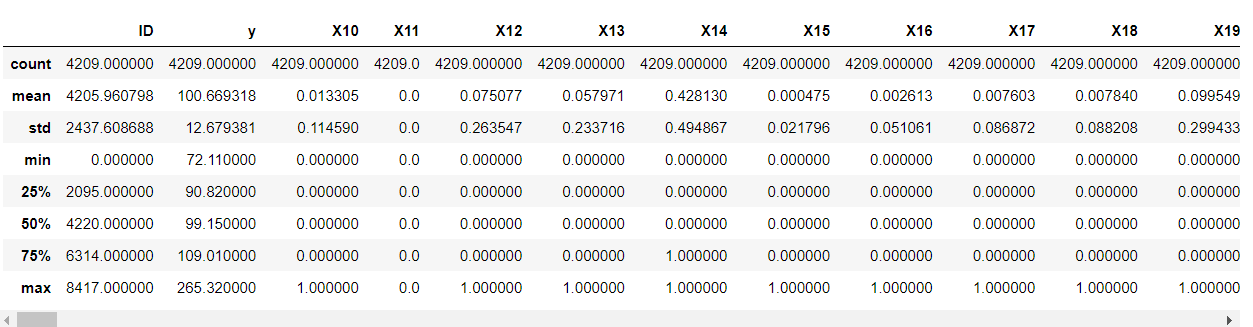
RangeIndex: 4209 entries, 0 to 4208

Columns: 378 entries, ID to X385

dtypes: float64(1), int64(369), object(8)

memory usage: 12.1+ MB

Step 4 – Lets check statistical information of dataset by using describe() method



Step 5 – Lets check the count of each ID type , just to ensure if there are any duplicates

Output:



Step 6 – Lets categorize the variables in object(categorical) and non-object (numerical) sections

*df\_cat = train\_data.select\_dtypes(include = np.object)*

*df\_num = train\_data.select\_dtypes(exclude=np.object)*

*print("Categorical variables:-\n",df\_cat.columns)*

*print("--------------------------------------------")*

*print("Numerical variables:-\n",df\_num.columns)*

*Output:*

Categorical variables:-

Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')

--------------------------------------------

Numerical variables:-

Index(['ID', 'y', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17',

...

'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',

'X385'],

dtype='object', length=370)

Step 7 – Lets plot the categorical columns with respect to output column i.e. “Y”

import matplotlib.pyplot as plt

Y = train\_data['y']

x0 = train\_data['X0']

x1 = train\_data['X1']

x2 = train\_data['X2']

x3 = train\_data['X3']

x4 = train\_data['X4']

x5 = train\_data['X5']

x6 = train\_data['X6']

x8 = train\_data['X8']

plt.subplot(1, 2, 1)

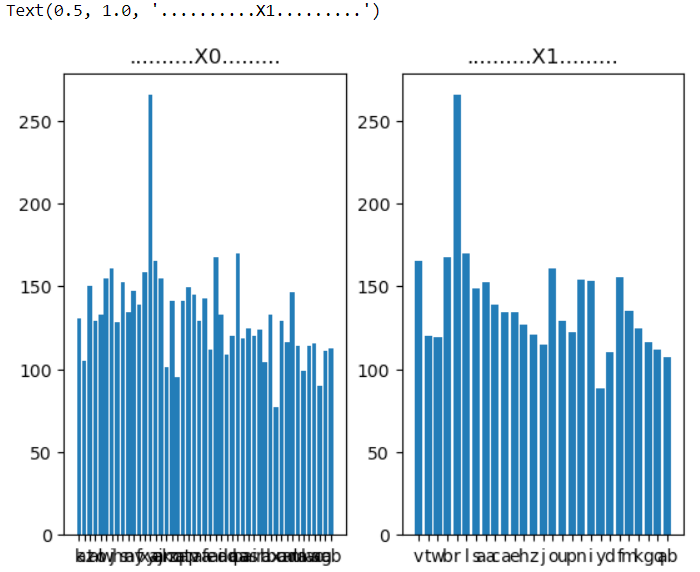
plt.bar(x0,Y)

plt.title("..........X0.........")

plt.subplot(1, 2, 2)

plt.bar(x1,Y)

plt.title("..........X1......…")



plt.subplot(1, 2, 1)

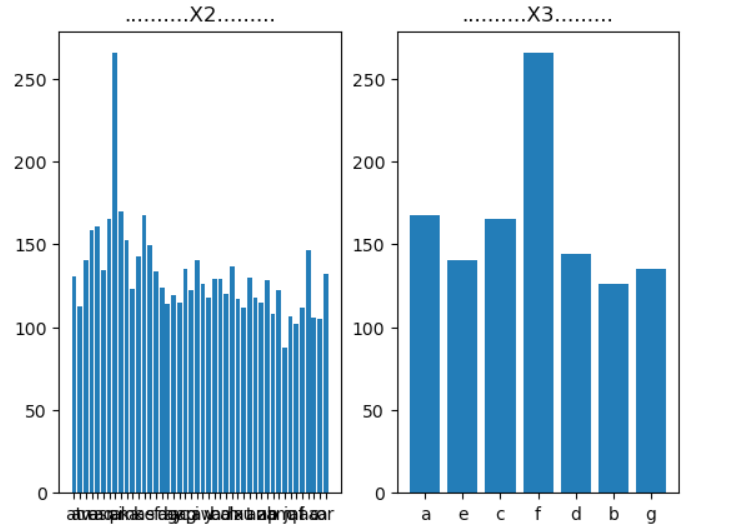
plt.bar(x2,Y)

plt.title("..........X2.........")

plt.subplot(1, 2, 2)

plt.bar(x3,Y)

plt.title("..........X3......…")



plt.subplot(1, 2, 1)

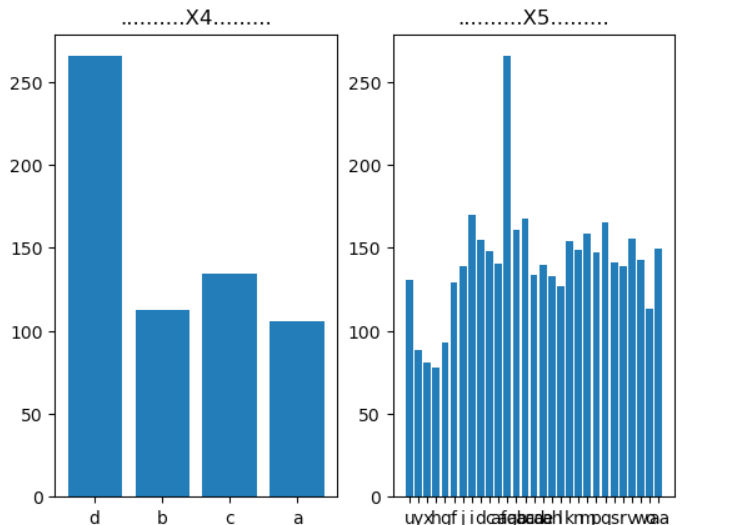
plt.bar(x4,Y)

plt.title("..........X4.........")

plt.subplot(1, 2, 2)

plt.bar(x5,Y)

plt.title("..........X5.........")



plt.subplot(1, 2, 1)

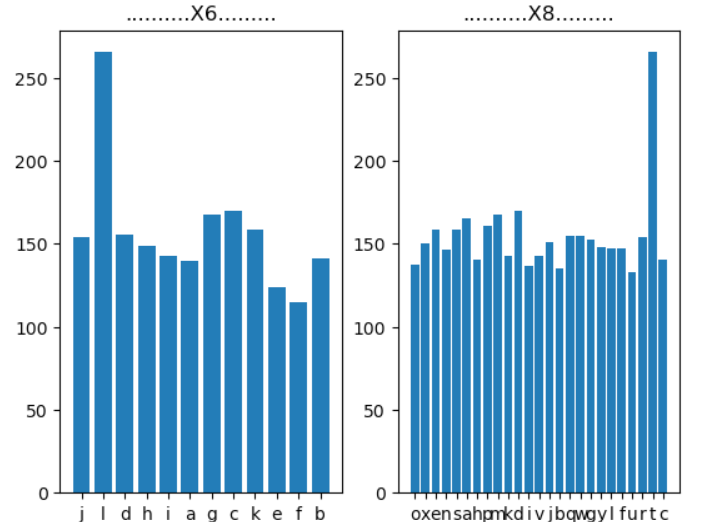
plt.bar(x6,Y)

plt.title("..........X6.........")

plt.subplot(1, 2, 2)

plt.bar(x8,Y)

plt.title("..........X8......…")



Now that we have analyzed the dataset , let us perform action 1 – i.e. find the variance of all columns and drop the columns which has variance equal to 0.

We will try to see variance to 3 decimal points

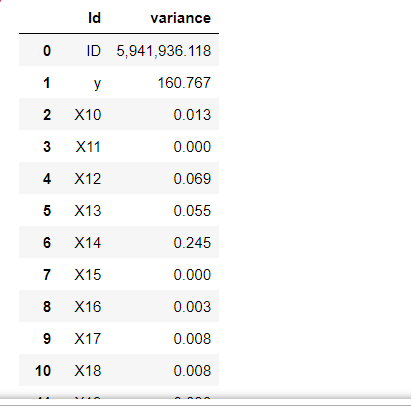
*pd.options.display.float\_format = '{:,.3f}'.format*

*variance\_col = train\_data.var()*

*variance\_col = variance\_col.reset\_index()*

*variance\_col.columns = ['Id', 'variance']*

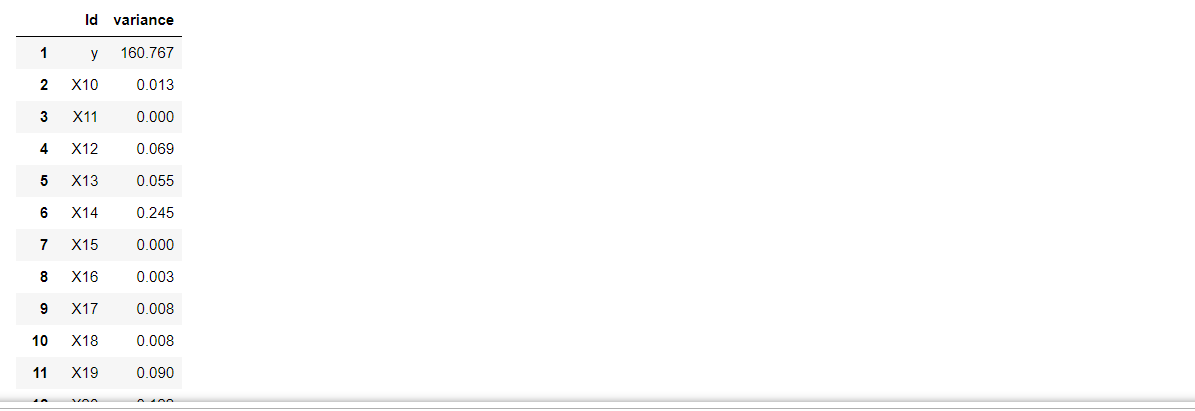
*variance\_col*



First note worthy observation here is - Variance is automatically calculated for numerical columns only, that is what we expected. From Above outcome we can see that first row 'ID' has largest variance, but this is not useful for our observation We see there are several rows e.g. X11, X15 have 0 variance, so these will not be impacted at all by IDV and row "ID" is vastly impacted by IDV, so lets drop them from our analysis

*variance\_col1 = variance\_col.drop(0)*

*variance\_col1*

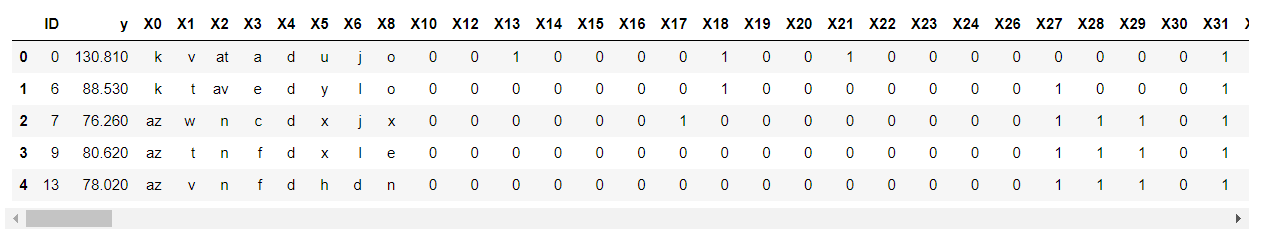


Lets created another Dataset named “train\_data\_updated” which does not have columns with 0 variance

train\_data\_updated = train\_data.drop(drop\_var, axis=1)

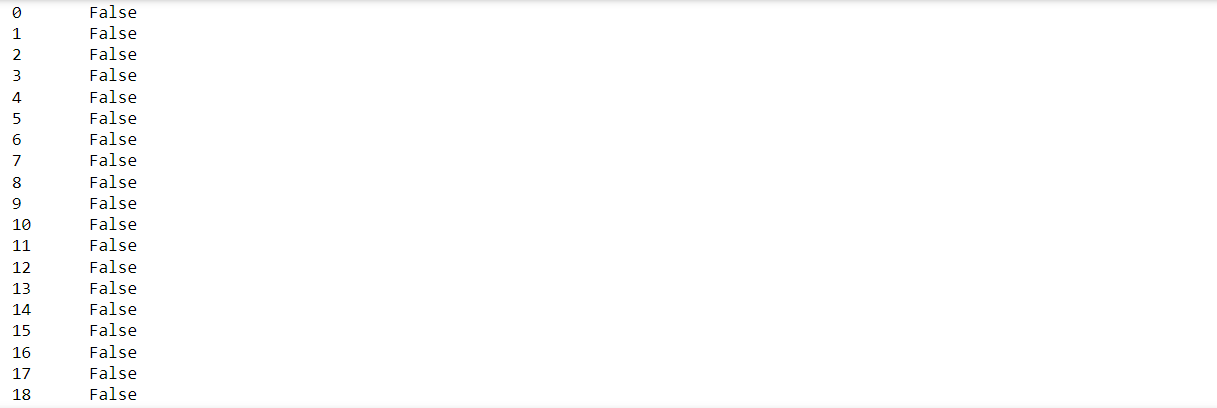
#train\_data\_updated.drop("ID", axis = 1, inplace=True)

train\_data\_updated.head()”

**Action 2. Check for null and unique values for test and train sets.**

Now lets check if this updated dataset has any null value along axis 1, I.e columns

train\_data\_updated.isnull().any(axis=1)

From the Output we can see that there are no null values in our given dataset.

Lets check if there are any duplicates in out train and test data set.

*test\_data['ID'].duplicated().sum()*

*Output: 0*

*train\_data\_updated['ID'].duplicated().sum()*

*Output: 0*

Our test and train data both has 0 duplicates so there is no need to remove any duplicate rows.

**Action 3: Now as next action we want to apply label encoder to the dataset1**

*from sklearn.preprocessing import LabelEncoder # Import LabelEncoder from sklear preprocessing*

*category\_cols = [col for col in train\_data.columns if train\_data[col].dtype == 'object']*

*le = LabelEncoder() #let us use le instead of LabelEncoder for ease of writting code*

*for col in category\_cols:*

*le.fit(train\_data[col].unique().tolist() + test\_data[col].unique().tolist())*

*train\_data[col] = le.transform(train\_data[col])*

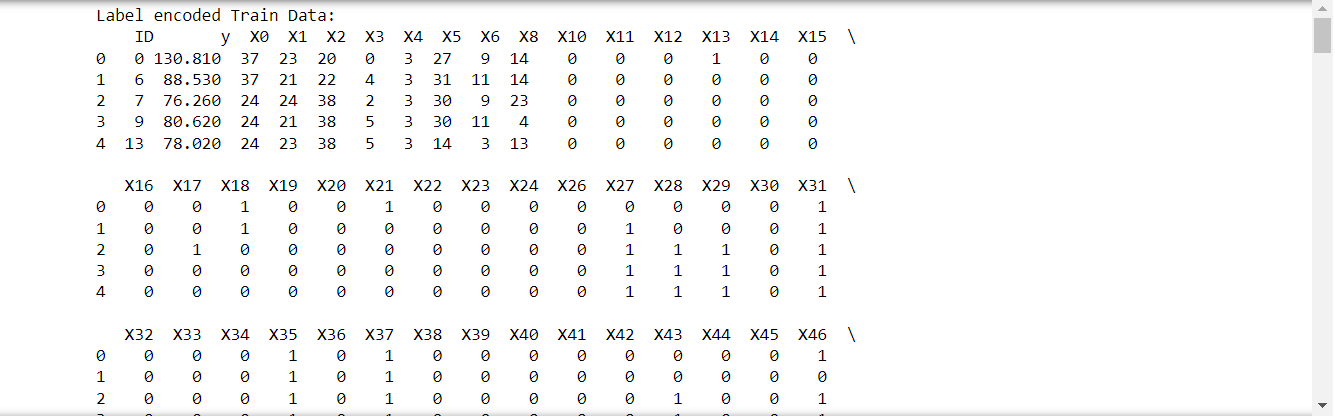
*test\_data[col] = le.transform(test\_data[col])*

*print("Categorical variables array: ", category\_cols)*

*Output:*

*Categorical variables array: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']*

*print("Label encoded Train Data:\n", train\_data.head())*

**Action 4. Perform dimensionality reduction.**

Lets do PCA on the dataset to perform dimensionality reduction. Lets also apply fit and transform on training data , and understand its explained variance ratio

*from sklearn.decomposition import PCA #import PCA from sklearn*

*pca = PCA(n\_components=10)*

*pca.fit(train\_data.drop('y', axis=1))*

*train\_data\_pca = pd.DataFrame(pca.transform(train\_data.drop('y', axis=1)))*

*train\_data\_pca['y'] = train\_data['y']*

*print("PCA variance ratios: ", pca.explained\_variance\_ratio\_)*

*test\_data\_pca = pd.DataFrame(pca.transform(test\_data))*

*print("Training data after principal component analysis :\n", test\_data\_pca.head())*

Lets Plot the correlation matrix after PCA

*import seaborn as sns*

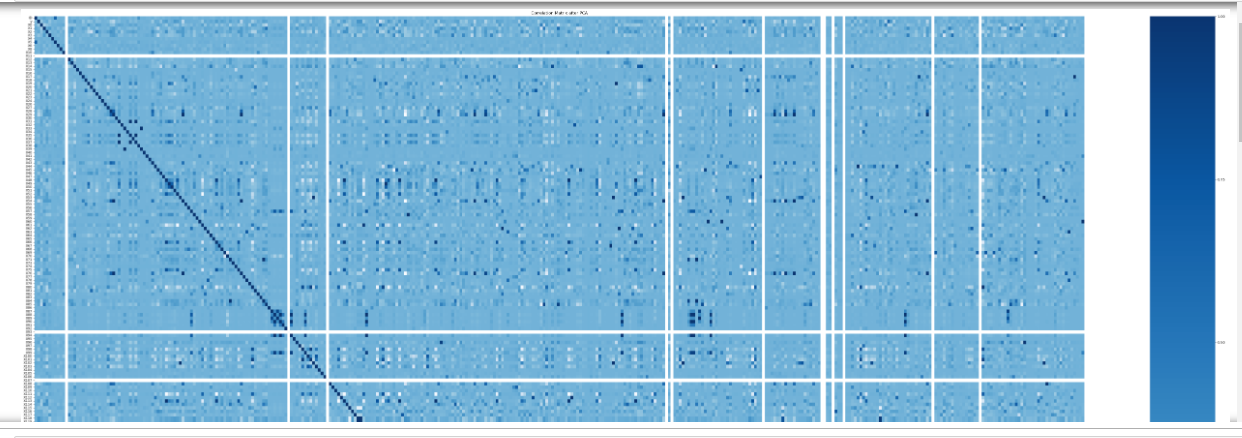
*corr = train\_data.corr()*

*fig, ax = plt.subplots(figsize=(70,70))*

*sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='Blues')*

*plt.title('Correlation Matrix after PCA')*

*plt.show()*



Since our dataset is too huge, heatmap is also very large and not entirely visible in this image, but after enlarging we can see that most important components here are x0 to x8.

Now let us segregate and split our data. So that we can compare our test results with prediction data

*X = train\_data.drop("y",axis=1)*

*y = train\_data.loc[:,"y"]*

*from sklearn.model\_selection import train\_test\_split*

*train\_X, test\_X, train\_y, test\_y = train\_test\_split(X, y,test\_size = 0.3, random\_state = 123)*

**Action 5: Predict your test\_df values using XGBoost.**

# Let us apply xgboots Algorithm now and conclude our predictions.

We need to follow below mentioned steps to apply xgboost algorithm

Step1: Import Xgboost regressor

Step2: define its parameters (learning rate, depth, gamma), no of estimator

Step3: apply or fit the algorithm

Step4: Predict the output

*# Train an XGBoost model and make predictions*

*from xgboost import XGBRegressor*

*param\_grid = {*

*'learning\_rate': [0.05, 0.1, 0.15],*

*'max\_depth': [3, 4, 5],*

*'min\_child\_weight': [1, 3, 5],*

*'gamma': [0, 0.1, 0.2]*

*}*

*xgb = XGBRegressor(n\_estimators=1000, objective='reg:squarederror', seed = 123)*

*xgb.fit(train\_X, train\_y)*

*prediction\_y = xgb.predict(test\_X)*

*print("Predictions:\n", prediction\_y)*

*Output: Predictions:*

*[110.59595 91.98747 98.93885 ... 96.68242 109.02192 106.353455]*

Let’s evaluate the model using MAPE

d = pd.DataFrame()

d["test\_y"] = test\_y

d["prediction\_y"] = prediction\_y

#lets calculate MAPE by using formula ((test-prediction)/test)\*100)

d["mp"] = abs((d["test\_y"]- d["prediction\_y"])/d["test\_y"])

(d.mp.mean())\*100

Output: 6.123251411538857

We are getting a mean absolute percentage error of 6.123251411538857% which is a good prediction. So our model is acceptable

Lets also calculate r2 score for our model.

*from sklearn.metrics import r2\_score*

*# Evaluating the model on the validation set*

*r2 = r2\_score(test\_y, prediction\_y)*

*print("R^2 score:", r2)*

Output: R^2 score: 0.4697389770989081

So we have successfully created a model with 6.123% MAPE and r2 score as 0.469 which indicates that our model is relatively accurate.