ML Project 1) Mercedes-Benz Greener Manufacturing

November 27, 2022

1 Mercedes-Benz Greener Manufacturing

1.0.1 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.

[ ]:

Note: Cells with "###" in first line are operations that we will need to perform on Forecast Data ("test.csv") before making Predictions on them.

[ ]:

[410]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.api import OLS

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA

from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBRegressor
from sklearn.metrics import r2_score

from math import sqrt

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

[ ]:
```
```

[302]:
```
df = pd.read_csv("train.csv")
```

[303]:
```
df.head()
```

[303]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	…	X375	X376	X377	X378	X379	\
0	0	130.81	k	v	at	a	d	u	j	o	…	0	0	1	0	0	
1	6	88.53	k	t	av	e	d	y	l	o	…	1	0	0	0	0	
2	7	76.26	az	w	n	c	d	x	j	x	…	0	0	0	0	0	
3	9	80.62	az	t	n	f	d	x	l	e	…	0	0	0	0	0	
4	13	78.02	az	v	n	f	d	h	d	n	…	0	0	0	0	0	

	X380	X382	X383	X384	X385
0	0	0	0	0	0
1	0	0	0	0	0
2	0	1	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

[5 rows x 378 columns]

[304]:
```
df.shape
```

[304]:
```
(4209, 378)
```

[305]:
```
df.columns
```

[305]:
```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

[306]:
```
df.describe().T
```

[306]:

	count	mean	std	min	25%	50%	75%	max
ID	4209.0	4205.960798	2437.608688	0.00	2095.00	4220.00	6314.00	8417.00
y	4209.0	100.669318	12.679381	72.11	90.82	99.15	109.01	265.32
X10	4209.0	0.013305	0.114590	0.00	0.00	0.00	0.00	1.00
X11	4209.0	0.000000	0.000000	0.00	0.00	0.00	0.00	0.00
X12	4209.0	0.075077	0.263547	0.00	0.00	0.00	0.00	1.00
…	…	…	…	…	…	…	…	…
X380	4209.0	0.008078	0.089524	0.00	0.00	0.00	0.00	1.00
X382	4209.0	0.007603	0.086872	0.00	0.00	0.00	0.00	1.00
X383	4209.0	0.001663	0.040752	0.00	0.00	0.00	0.00	1.00
X384	4209.0	0.000475	0.021796	0.00	0.00	0.00	0.00	1.00
X385	4209.0	0.001426	0.037734	0.00	0.00	0.00	0.00	1.00

[370 rows x 8 columns]

[307]:
```
df.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
```

[ ]:
```

```

1.0.2 Missing Value Treatment:

[308]:
```
df.isna().sum().sum()
```

[308]:
```
0
```

[309]:
```
# There are  no Missing Values in Data.
```

[ ]:
```

```

1.0.3 Checking For Numeric Cintinuous, Numeric Discrete, Numeric Categorical and Categorical Variables:

[310]:
```
df.dtypes
```

[310]:
```
ID        int64
y       float64
X0       object
X1       object
X2       object
         …
X380      int64
X382      int64
X383      int64
X384      int64
X385      int64
Length: 378, dtype: object
```

[311]:
```
# For Object Data Types:

for cols in df.select_dtypes(include= "object"):
    print(cols)
    print(df[cols].nunique())
    print("\n")
```

```
X0
47


X1
27


X2
44


X3
```

```
7


X4
4


X5
29


X6
12


X8
25
```

[312]:
```
# All Object Type Columms are Categorical.
# We will Perform Lable Encoder on Them.
```

[ ]:
```

```

[313]:
```
# Cheking Numeric Data Features:

for cols in df.select_dtypes(exclude= "object"):
    print(cols)
    print(df[cols].nunique())
```

```
ID
4209
y
2545
X10
2
X11
1
X12
2
X13
2
X14
2
X15
2
X16
```

2
X17
2
X18
2
X19
2
X20
2
X21
2
X22
2
X23
2
X24
2
X26
2
X27
2
X28
2
X29
2
X30
2
X31
2
X32
2
X33
2
X34
2
X35
2
X36
2
X37
2
X38
2
X39
2
X40
2
X41

2
X42
2
X43
2
X44
2
X45
2
X46
2
X47
2
X48
2
X49
2
X50
2
X51
2
X52
2
X53
2
X54
2
X55
2
X56
2
X57
2
X58
2
X59
2
X60
2
X61
2
X62
2
X63
2
X64
2
X65

2
X66
2
X67
2
X68
2
X69
2
X70
2
X71
2
X73
2
X74
2
X75
2
X76
2
X77
2
X78
2
X79
2
X80
2
X81
2
X82
2
X83
2
X84
2
X85
2
X86
2
X87
2
X88
2
X89
2
X90

2
X91
2
X92
2
X93
1
X94
2
X95
2
X96
2
X97
2
X98
2
X99
2
X100
2
X101
2
X102
2
X103
2
X104
2
X105
2
X106
2
X107
1
X108
2
X109
2
X110
2
X111
2
X112
2
X113
2
X114

2
X115
2
X116
2
X117
2
X118
2
X119
2
X120
2
X122
2
X123
2
X124
2
X125
2
X126
2
X127
2
X128
2
X129
2
X130
2
X131
2
X132
2
X133
2
X134
2
X135
2
X136
2
X137
2
X138
2
X139

2
X140
2
X141
2
X142
2
X143
2
X144
2
X145
2
X146
2
X147
2
X148
2
X150
2
X151
2
X152
2
X153
2
X154
2
X155
2
X156
2
X157
2
X158
2
X159
2
X160
2
X161
2
X162
2
X163
2
X164

2
X165
2
X166
2
X167
2
X168
2
X169
2
X170
2
X171
2
X172
2
X173
2
X174
2
X175
2
X176
2
X177
2
X178
2
X179
2
X180
2
X181
2
X182
2
X183
2
X184
2
X185
2
X186
2
X187
2
X189

2
X190
2
X191
2
X192
2
X194
2
X195
2
X196
2
X197
2
X198
2
X199
2
X200
2
X201
2
X202
2
X203
2
X204
2
X205
2
X206
2
X207
2
X208
2
X209
2
X210
2
X211
2
X212
2
X213
2
X214

2
X215
2
X216
2
X217
2
X218
2
X219
2
X220
2
X221
2
X222
2
X223
2
X224
2
X225
2
X226
2
X227
2
X228
2
X229
2
X230
2
X231
2
X232
2
X233
1
X234
2
X235
1
X236
2
X237
2
X238

2
X239
2
X240
2
X241
2
X242
2
X243
2
X244
2
X245
2
X246
2
X247
2
X248
2
X249
2
X250
2
X251
2
X252
2
X253
2
X254
2
X255
2
X256
2
X257
2
X258
2
X259
2
X260
2
X261
2
X262

2
X263
2
X264
2
X265
2
X266
2
X267
2
X268
1
X269
2
X270
2
X271
2
X272
2
X273
2
X274
2
X275
2
X276
2
X277
2
X278
2
X279
2
X280
2
X281
2
X282
2
X283
2
X284
2
X285
2
X286

2
X287
2
X288
2
X289
1
X290
1
X291
2
X292
2
X293
1
X294
2
X295
2
X296
2
X297
1
X298
2
X299
2
X300
2
X301
2
X302
2
X304
2
X305
2
X306
2
X307
2
X308
2
X309
2
X310
2
X311

2
X312
2
X313
2
X314
2
X315
2
X316
2
X317
2
X318
2
X319
2
X320
2
X321
2
X322
2
X323
2
X324
2
X325
2
X326
2
X327
2
X328
2
X329
2
X330
1
X331
2
X332
2
X333
2
X334
2
X335

2
X336
2
X337
2
X338
2
X339
2
X340
2
X341
2
X342
2
X343
2
X344
2
X345
2
X346
2
X347
1
X348
2
X349
2
X350
2
X351
2
X352
2
X353
2
X354
2
X355
2
X356
2
X357
2
X358
2
X359

2
X360
2
X361
2
X362
2
X363
2
X364
2
X365
2
X366
2
X367
2
X368
2
X369
2
X370
2
X371
2
X372
2
X373
2
X374
2
X375
2
X376
2
X377
2
X378
2
X379
2
X380
2
X382
2
X383
2
X384

```
2
X385
2
```

[314]:
```
# Except "ID" and "y", all other Numeric Columns are Binary Categorical.

# "ID" has unique values for all observations. We can either drop it or make it␣
↪an Index Column.

# "y" is our Target Variable (Numeric Continuous).
```

[ ]:
```

```

[315]:
```
###

df = df.set_index("ID")
```

[316]:
```
df.head()
```

[316]:

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	…	X375	X376	X377	X378	X379	\
											…						
0	130.81	k	v	at	a	d	u	j	o	0	…	0	0	1	0	0	
6	88.53	k	t	av	e	d	y	l	o	0	…	1	0	0	0	0	
7	76.26	az	w	n	c	d	x	j	x	0	…	0	0	0	0	0	
9	80.62	az	t	n	f	d	x	l	e	0	…	0	0	0	0	0	
13	78.02	az	v	n	f	d	h	d	n	0	…	0	0	0	0	0	

ID	X380	X382	X383	X384	X385
0	0	0	0	0	0
6	0	0	0	0	0
7	0	1	0	0	0
9	0	0	0	0	0
13	0	0	0	0	0

[5 rows x 377 columns]

[ ]:
```

```

1.0.4 Applying Label Encoder on Object Data Type Columns:

[317]:
```
###

le = LabelEncoder()
```

[318]:
```
cat_col = list(df.select_dtypes(include="object").columns)
```

[319]:
```
cat_col
```

[319]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[320]:
```
for col in cat_col:
    df[col] = le.fit_transform(df[col])
```

[321]:
```
df.head()
```

[321]:

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	…	X375	X376	X377	X378	\
0	130.81	32	23	17	0	3	24	9	14	0	…	0	0	1	0	
6	88.53	32	21	19	4	3	28	11	14	0	…	1	0	0	0	
7	76.26	20	24	34	2	3	27	9	23	0	…	0	0	0	0	
9	80.62	20	21	34	5	3	27	11	4	0	…	0	0	0	0	
13	78.02	20	23	34	5	3	12	3	13	0	…	0	0	0	0	

ID	X379	X380	X382	X383	X384	X385
0	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	1	0	0	0
9	0	0	0	0	0	0
13	0	0	0	0	0	0

[5 rows x 377 columns]

[322]:
```
df.select_dtypes(include= "object")
```

[322]:
```
Empty DataFrame
Columns: []
Index: [0, 6, 7, 9, 13, 18, 24, 25, 27, 30, 31, 32, 34, 36, 37, 38, 39, 40, 44,
47, 48, 49, 50, 52, 54, 60, 61, 62, 66, 67, 68, 70, 74, 75, 79, 80, 81, 86, 90,
92, 100, 102, 106, 107, 108, 109, 112, 116, 118, 119, 124, 125, 127, 128, 129,
130, 131, 133, 134, 136, 139, 141, 143, 145, 146, 147, 150, 151, 152, 153, 154,
158, 159, 160, 161, 164, 165, 166, 168, 169, 170, 171, 173, 174, 175, 176, 177,
178, 184, 186, 190, 197, 200, 202, 203, 207, 208, 209, 212, 213, …]

[4209 rows x 0 columns]
```

[323]:
```
# All Object Type Categorical Columns have been Label Encoded.
```

[ ]:
```

```

1.0.5 Checking Variance of All Features:

[324]:
```
cols_to_drop = []

for cols in df.columns:

```

```
    if df[cols].var() == 0:
        cols_to_drop.append(cols)
```

[325]:
```
cols_to_drop
```

[325]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

[326]:
```
# This Columns have no Variance in it:

df[cols_to_drop].nunique()
```

[326]:
```
X11     1
X93     1
X107    1
X233    1
X235    1
X268    1
X289    1
X290    1
X293    1
X297    1
X330    1
X347    1
dtype: int64
```

[327]:
```
# We can Drop These Columns.
```

[328]:
```
###

df = df.drop(cols_to_drop, axis= 1)
```

[329]:
```
df.shape
```

[329]:
```
(4209, 365)
```

[ ]:
```

```

1.0.6 Using Ordinary Least Square Method to Find Out Features Significant to Target Variable:

[330]:
```
x = df.drop("y", axis =1)
y = df["y"]
```

[331]:
```
x.head()
```

[331]:
```
    X0  X1  X2  X3  X4  X5  X6  X8  X10  X12  …  X375  X376  X377  X378  \
ID                                                …
0   32  23  17   0   3  24   9  14    0    0  …     0     0     1     0
6   32  21  19   4   3  28  11  14    0    0  …     1     0     0     0
7   20  24  34   2   3  27   9  23    0    0  …     0     0     0     0
9   20  21  34   5   3  27  11   4    0    0  …     0     0     0     0
13  20  23  34   5   3  12   3  13    0    0  …     0     0     0     0

    X379  X380  X382  X383  X384  X385
ID
0      0     0     0     0     0     0
6      0     0     0     0     0     0
7      0     0     1     0     0     0
9      0     0     0     0     0     0
13     0     0     0     0     0     0

[5 rows x 364 columns]
```

[332]:
```
x.shape
```

[332]:
```
(4209, 364)
```

[333]:
```
y.shape
```

[333]:
```
(4209,)
```

[334]:
```
y.head()
```

[334]:
```
ID
0     130.81
6      88.53
7      76.26
9      80.62
13     78.02
Name: y, dtype: float64
```

[ ]:
```

```

[335]:
```
ols_model = OLS(y, x)
```

[336]:
```
results = ols_model.fit()
```

[337]:
```
results.summary()
```

[337]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.592
Model:                            OLS   Adj. R-squared:                  0.566
Method:                 Least Squares   F-statistic:                     22.15
Date:                Fri, 25 Nov 2022   Prob (F-statistic):               0.00
Time:                        12:02:00   Log-Likelihood:                -14775.
No. Observations:                4209   AIC:                         3.007e+04
Df Residuals:                    3949   BIC:                         3.172e+04
Df Model:                         259
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
X0             0.0665      0.015      4.351      0.000       0.037       0.097
X1            -0.0418      0.038     -1.090      0.276      -0.117       0.033
X2             0.0082      0.052      0.156      0.876      -0.094       0.111
X3            -0.1232      0.126     -0.978      0.328      -0.370       0.124
X4            -0.1255      1.771     -0.071      0.944      -3.598       3.347
X5            -0.0532      0.017     -3.089      0.002      -0.087      -0.019
X6             0.0392      0.050      0.786      0.432      -0.059       0.137
X8             0.0013      0.020      0.067      0.946      -0.038       0.041
X10            4.7385      4.469      1.060      0.289      -4.024      13.501
X12            3.4574      3.144      1.100      0.272      -2.707       9.622
X13            1.7291      1.614      1.071      0.284      -1.435       4.893
X14            2.6897      3.317      0.811      0.417      -3.813       9.192
X15            9.7476      9.592      1.016      0.310      -9.058      28.554
X16            7.7675      4.661      1.667      0.096      -1.370      16.905
X17           -0.1173      1.629     -0.072      0.943      -3.311       3.077
X18            6.0716      4.946      1.228      0.220      -3.624      15.768
X19            2.8901      5.444      0.531      0.596      -7.784      13.564
X20            2.3847      3.227      0.739      0.460      -3.942       8.711
X21           -1.8490      4.642     -0.398      0.690     -10.949       7.251
X22            1.3970      3.261      0.428      0.668      -4.997       7.791
X23           -0.1387      3.449     -0.040      0.968      -6.901       6.624
X24           -2.8952      3.210     -0.902      0.367      -9.188       3.398
X26            4.5853      4.812      0.953      0.341      -4.850      14.020
X27           -0.0250      0.567     -0.044      0.965      -1.137       1.087
X28            4.5681      5.313      0.860      0.390      -5.849      14.985
X29           -0.1245      1.830     -0.068      0.946      -3.712       3.463
X30          -19.3617     14.885     -1.301      0.193     -48.544       9.821
```

X31	0.2870	0.191	1.501	0.133	-0.088	0.662
X32	6.2324	14.995	0.416	0.678	-23.167	35.632
X33	-4.1785	4.461	-0.937	0.349	-12.926	4.569
X34	7.5138	7.788	0.965	0.335	-7.754	22.782
X35	0.2870	0.191	1.501	0.133	-0.088	0.662
X36	0.3436	4.166	0.082	0.934	-7.824	8.511
X37	0.2870	0.191	1.501	0.133	-0.088	0.662
X38	-1.4369	0.803	-1.789	0.074	-3.012	0.138
X39	-4.1785	4.461	-0.937	0.349	-12.926	4.569
X40	-0.3397	4.572	-0.074	0.941	-9.303	8.624
X41	0.5267	1.623	0.325	0.746	-2.655	3.709
X42	3.7311	8.727	0.428	0.669	-13.378	20.840
X43	-0.6690	2.293	-0.292	0.770	-5.164	3.826
X44	-0.8078	3.507	-0.230	0.818	-7.683	6.068
X45	-5.3797	1.836	-2.931	0.003	-8.979	-1.781
X46	0.3032	0.408	0.743	0.458	-0.497	1.103
X47	5.6377	2.333	2.416	0.016	1.063	10.212
X48	3.4414	1.582	2.175	0.030	0.339	6.544
X49	4.0971	6.866	0.597	0.551	-9.364	17.558
X50	-0.0345	0.586	-0.059	0.953	-1.184	1.115
X51	0.2035	0.475	0.428	0.669	-0.728	1.135
X52	9.4499	2.964	3.188	0.001	3.638	15.262
X53	2.2585	1.800	1.254	0.210	-1.271	5.788
X54	8.9090	2.076	4.292	0.000	4.839	12.979
X55	-3.8646	2.941	-1.314	0.189	-9.631	1.902
X56	0.3284	1.319	0.249	0.803	-2.258	2.914
X57	-2.2168	4.334	-0.512	0.609	-10.714	6.280
X58	-0.3913	0.612	-0.639	0.523	-1.592	0.809
X59	-2.5107	3.920	-0.640	0.522	-10.196	5.175
X60	-1.3332	1.051	-1.269	0.205	-3.394	0.727
X61	-0.7186	5.686	-0.126	0.899	-11.866	10.429
X62	1.1134	2.432	0.458	0.647	-3.656	5.882
X63	-0.5934	1.814	-0.327	0.744	-4.150	2.963
X64	-0.3013	0.431	-0.699	0.484	-1.146	0.543
X65	-2.6270	8.893	-0.295	0.768	-20.062	14.808
X66	1.1568	2.839	0.407	0.684	-4.410	6.723
X67	-3.0562	2.792	-1.095	0.274	-8.530	2.418
X68	-0.4810	0.940	-0.512	0.609	-2.324	1.362
X69	-0.5852	1.575	-0.372	0.710	-3.673	2.503
X70	-0.6355	0.933	-0.681	0.496	-2.465	1.194
X71	0.5827	0.510	1.144	0.253	-0.416	1.582
X73	1.1820	1.070	1.104	0.270	-0.917	3.281
X74	11.0538	5.274	2.096	0.036	0.713	21.395
X75	-3.4338	1.463	-2.347	0.019	-6.303	-0.565
X76	8.9090	2.076	4.292	0.000	4.839	12.979
X77	1.1064	1.508	0.734	0.463	-1.851	4.063
X78	0.6168	3.616	0.171	0.865	-6.472	7.706

X79	-3.4251	1.005	-3.410	0.001	-5.395	-1.456
X80	3.1330	4.009	0.782	0.435	-4.726	10.992
X81	-0.8783	0.616	-1.425	0.154	-2.087	0.330
X82	-0.1402	1.416	-0.099	0.921	-2.916	2.635
X83	-15.4402	15.396	-1.003	0.316	-45.625	14.745
X84	0.5827	0.510	1.144	0.253	-0.416	1.582
X85	-0.9674	1.652	-0.586	0.558	-4.205	2.271
X86	7.8595	5.037	1.560	0.119	-2.017	17.736
X87	-1.0734	11.891	-0.090	0.928	-24.386	22.239
X88	-0.9138	1.878	-0.487	0.627	-4.596	2.768
X89	1.5438	4.119	0.375	0.708	-6.531	9.619
X90	-2.9601	2.613	-1.133	0.257	-8.083	2.163
X91	-4.6875	3.862	-1.214	0.225	-12.259	2.884
X92	-13.4201	9.785	-1.371	0.170	-32.605	5.765
X94	-2.9601	2.613	-1.133	0.257	-8.083	2.163
X95	10.6001	3.649	2.905	0.004	3.446	17.754
X96	3.3984	2.204	1.542	0.123	-0.922	7.719
X97	-10.8255	12.722	-0.851	0.395	-35.767	14.116
X98	-8.3557	7.840	-1.066	0.287	-23.726	7.014
X99	-2.2470	1.711	-1.313	0.189	-5.601	1.107
X100	-0.8166	0.732	-1.116	0.264	-2.251	0.618
X101	-10.6141	8.876	-1.196	0.232	-28.015	6.787
X102	2.2585	1.800	1.254	0.210	-1.271	5.788
X103	-0.5690	3.201	-0.178	0.859	-6.845	5.707
X104	7.0358	3.120	2.255	0.024	0.919	13.153
X105	5.1557	4.174	1.235	0.217	-3.028	13.340
X106	-0.8702	1.751	-0.497	0.619	-4.303	2.562
X108	0.5942	1.968	0.302	0.763	-3.264	4.453
X109	0.5332	1.247	0.428	0.669	-1.911	2.978
X110	2.0554	4.833	0.425	0.671	-7.420	11.531
X111	14.4421	2.436	5.928	0.000	9.665	19.219
X112	0.7716	2.848	0.271	0.786	-4.812	6.355
X113	3.4414	1.582	2.175	0.030	0.339	6.544
X114	-0.7451	0.867	-0.859	0.390	-2.446	0.955
X115	8.6956	1.211	7.180	0.000	6.321	11.070
X116	-0.1983	0.677	-0.293	0.770	-1.525	1.129
X117	4.5080	1.259	3.579	0.000	2.039	6.977
X118	5.5760	0.784	7.108	0.000	4.038	7.114
X119	5.5760	0.784	7.108	0.000	4.038	7.114
X120	9.2052	3.148	2.925	0.003	3.034	15.376
X122	-0.9138	1.878	-0.487	0.627	-4.596	2.768
X123	7.3980	2.584	2.863	0.004	2.332	12.464
X124	-0.4529	6.669	-0.068	0.946	-13.527	12.621
X125	-1.4995	3.487	-0.430	0.667	-8.336	5.337
X126	-1.3308	7.062	-0.188	0.851	-15.176	12.514
X127	-3.4765	1.798	-1.934	0.053	-7.001	0.048
X128	5.1062	2.581	1.978	0.048	0.046	10.167

X129	-4.2733	6.814	-0.627	0.531	-17.632	9.085
X130	13.5489	2.496	5.428	0.000	8.655	18.443
X131	0.2727	1.616	0.169	0.866	-2.895	3.441
X132	-0.0982	1.421	-0.069	0.945	-2.885	2.688
X133	5.6352	1.318	4.276	0.000	3.051	8.219
X134	3.4414	1.582	2.175	0.030	0.339	6.544
X135	2.1595	1.871	1.154	0.248	-1.508	5.827
X136	9.7461	2.424	4.021	0.000	4.994	14.499
X137	1.8672	1.211	1.542	0.123	-0.507	4.241
X138	2.6985	1.813	1.488	0.137	-0.857	6.254
X139	2.2939	1.613	1.422	0.155	-0.869	5.457
X140	1.2054	2.101	0.574	0.566	-2.913	5.324
X141	0.4603	1.272	0.362	0.718	-2.034	2.955
X142	9.2060	1.048	8.784	0.000	7.151	11.261
X143	6.8039	1.988	3.422	0.001	2.905	10.702
X144	3.4173	1.785	1.915	0.056	-0.082	6.916
X145	1.9805	3.465	0.572	0.568	-4.813	8.774
X146	2.6985	1.813	1.488	0.137	-0.857	6.254
X147	3.4414	1.582	2.175	0.030	0.339	6.544
X148	-2.2114	1.362	-1.624	0.104	-4.881	0.458
X150	-0.3893	1.488	-0.262	0.794	-3.306	2.527
X151	0.6235	0.545	1.144	0.253	-0.445	1.692
X152	1.0700	0.445	2.402	0.016	0.197	1.943
X153	1.9543	6.595	0.296	0.767	-10.976	14.884
X154	0.1523	1.699	0.090	0.929	-3.179	3.483
X155	1.5364	1.158	1.327	0.184	-0.733	3.806
X156	9.6463	1.042	9.260	0.000	7.604	11.689
X157	9.0089	1.042	8.648	0.000	6.966	11.051
X158	9.4491	1.030	9.173	0.000	7.430	11.469
X159	-1.6289	3.609	-0.451	0.652	-8.705	5.448
X160	-3.8335	3.596	-1.066	0.286	-10.883	3.216
X161	0.5496	3.430	0.160	0.873	-6.174	7.274
X162	2.6207	1.921	1.364	0.173	-1.145	6.387
X163	-1.7856	0.550	-3.245	0.001	-2.864	-0.707
X164	-0.8887	1.167	-0.761	0.447	-3.178	1.400
X165	0.3218	4.523	0.071	0.943	-8.545	9.189
X166	0.5592	2.978	0.188	0.851	-5.280	6.398
X167	1.0881	5.450	0.200	0.842	-9.598	11.774
X168	1.0439	4.193	0.249	0.803	-7.177	9.265
X169	0.9733	2.795	0.348	0.728	-4.507	6.454
X170	0.6731	5.209	0.129	0.897	-9.540	10.887
X171	1.9114	4.188	0.456	0.648	-6.299	10.122
X172	1.1134	2.432	0.458	0.647	-3.656	5.882
X173	1.1416	2.425	0.471	0.638	-3.613	5.896
X174	4.4777	1.746	2.564	0.010	1.054	7.901
X175	-0.4330	1.233	-0.351	0.726	-2.851	1.985
X176	0.0869	1.358	0.064	0.949	-2.575	2.749

X177	-0.0385	1.172	-0.033	0.974	-2.336	2.259
X178	-21.4583	8.757	-2.450	0.014	-38.626	-4.290
X179	-25.1639	9.334	-2.696	0.007	-43.464	-6.863
X180	5.4474	1.136	4.794	0.000	3.219	7.675
X181	0.4671	1.138	0.410	0.682	-1.764	2.698
X182	-0.7200	1.125	-0.640	0.522	-2.925	1.485
X183	-1.8056	3.756	-0.481	0.631	-9.169	5.558
X184	0.7282	2.056	0.354	0.723	-3.303	4.760
X185	2.4788	2.129	1.164	0.244	-1.696	6.653
X186	11.4641	2.560	4.478	0.000	6.445	16.483
X187	-1.7180	2.384	-0.721	0.471	-6.393	2.957
X189	5.2306	2.137	2.448	0.014	1.041	9.420
X190	-4.5516	8.582	-0.530	0.596	-21.377	12.274
X191	0.6675	1.417	0.471	0.638	-2.110	3.445
X192	-0.8335	3.037	-0.274	0.784	-6.789	5.122
X194	7.1910	2.704	2.659	0.008	1.889	12.493
X195	1.0222	1.484	0.689	0.491	-1.888	3.932
X196	2.1665	1.470	1.474	0.141	-0.715	5.048
X197	-0.4801	2.410	-0.199	0.842	-5.205	4.245
X198	0.9308	4.167	0.223	0.823	-7.240	9.101
X199	0.7716	2.848	0.271	0.786	-4.812	6.355
X200	-2.2581	3.708	-0.609	0.543	-9.529	5.013
X201	-5.1378	1.732	-2.967	0.003	-8.533	-1.742
X202	-0.3471	2.159	-0.161	0.872	-4.580	3.885
X203	-2.0492	1.304	-1.571	0.116	-4.606	0.508
X204	22.9028	4.700	4.873	0.000	13.688	32.118
X205	-4.2477	3.970	-1.070	0.285	-12.031	3.536
X206	-2.6192	1.137	-2.304	0.021	-4.848	-0.390
X207	9.3218	5.364	1.738	0.082	-1.195	19.839
X208	-3.1819	3.915	-0.813	0.416	-10.858	4.494
X209	5.7113	1.085	5.266	0.000	3.585	7.838
X210	-30.1599	13.004	-2.319	0.020	-55.655	-4.664
X211	0.7379	2.257	0.327	0.744	-3.687	5.163
X212	-0.8362	2.186	-0.382	0.702	-5.123	3.450
X213	-3.0562	2.792	-1.095	0.274	-8.530	2.418
X214	2.2585	1.800	1.254	0.210	-1.271	5.788
X215	-1.2482	4.963	-0.251	0.801	-10.979	8.482
X216	1.1134	2.432	0.458	0.647	-3.656	5.882
X217	9.6863	3.866	2.505	0.012	2.106	17.267
X218	0.4692	0.537	0.874	0.382	-0.584	1.522
X219	-2.7310	2.133	-1.281	0.200	-6.912	1.450
X220	-0.0587	0.488	-0.120	0.904	-1.015	0.898
X221	-0.4841	3.120	-0.155	0.877	-6.600	5.632
X222	3.4414	1.582	2.175	0.030	0.339	6.544
X223	-1.0272	0.754	-1.362	0.173	-2.506	0.451
X224	0.2534	0.753	0.336	0.737	-1.223	1.730
X225	0.1502	0.584	0.257	0.797	-0.994	1.294

X226	1.0700	0.445	2.402	0.016	0.197	1.943
X227	-1.4995	3.487	-0.430	0.667	-8.336	5.337
X228	-3.0174	2.335	-1.292	0.196	-7.595	1.560
X229	-2.7741	2.255	-1.230	0.219	-7.195	1.647
X230	-0.1411	1.022	-0.138	0.890	-2.145	1.863
X231	0.8863	1.244	0.712	0.476	-1.554	3.326
X232	-0.1245	1.830	-0.068	0.946	-3.712	3.463
X234	1.2105	1.095	1.105	0.269	-0.937	3.358
X236	9.0335	3.227	2.799	0.005	2.707	15.360
X237	-2.8981	5.792	-0.500	0.617	-14.253	8.457
X238	-2.5900	8.094	-0.320	0.749	-18.459	13.279
X239	2.2585	1.800	1.254	0.210	-1.271	5.788
X240	3.1495	1.244	2.532	0.011	0.711	5.588
X241	-1.1619	1.526	-0.762	0.446	-4.153	1.829
X242	-2.9601	2.613	-1.133	0.257	-8.083	2.163
X243	-0.9138	1.878	-0.487	0.627	-4.596	2.768
X244	0.5827	0.510	1.144	0.253	-0.416	1.582
X245	1.5438	4.119	0.375	0.708	-6.531	9.619
X246	-0.8616	1.650	-0.522	0.602	-4.096	2.373
X247	-0.3471	2.159	-0.161	0.872	-4.580	3.885
X248	-1.3332	1.051	-1.269	0.205	-3.394	0.727
X249	7.6400	3.478	2.197	0.028	0.821	14.459
X250	12.1091	4.116	2.942	0.003	4.040	20.178
X251	-8.9760	5.641	-1.591	0.112	-20.036	2.084
X252	7.1027	6.589	1.078	0.281	-5.816	20.022
X253	-1.3332	1.051	-1.269	0.205	-3.394	0.727
X254	-0.1411	1.022	-0.138	0.890	-2.145	1.863
X255	-3.0915	4.188	-0.738	0.460	-11.303	5.120
X256	-0.7624	2.751	-0.277	0.782	-6.155	4.630
X257	-8.8947	6.344	-1.402	0.161	-21.333	3.543
X258	1.7268	4.438	0.389	0.697	-6.974	10.427
X259	-5.8790	6.476	-0.908	0.364	-18.577	6.819
X260	11.5840	6.605	1.754	0.080	-1.366	24.534
X261	6.6708	4.203	1.587	0.113	-1.569	14.910
X262	0.7282	2.056	0.354	0.723	-3.303	4.760
X263	18.7796	2.319	8.097	0.000	14.232	23.327
X264	9.4496	6.312	1.497	0.134	-2.925	21.824
X265	-1.1560	4.601	-0.251	0.802	-10.177	7.865
X266	0.7282	2.056	0.354	0.723	-3.303	4.760
X267	2.1121	2.729	0.774	0.439	-3.238	7.462
X269	0.0564	6.328	0.009	0.993	-12.350	12.463
X270	-2.3394	9.093	-0.257	0.797	-20.167	15.488
X271	4.9377	3.666	1.347	0.178	-2.249	12.125
X272	6.2260	3.063	2.033	0.042	0.221	12.231
X273	-0.4736	0.575	-0.823	0.410	-1.601	0.654
X274	2.8510	2.612	1.092	0.275	-2.270	7.972
X275	-0.0835	0.670	-0.125	0.901	-1.396	1.229

X276	0.1806	2.883	0.063	0.950	-5.473	5.834
X277	-4.7078	6.178	-0.762	0.446	-16.819	7.404
X278	-0.5913	5.847	-0.101	0.919	-12.054	10.871
X279	-0.1245	1.830	-0.068	0.946	-3.712	3.463
X280	4.7537	4.967	0.957	0.339	-4.985	14.493
X281	-3.5112	3.729	-0.942	0.346	-10.823	3.800
X282	1.0159	3.561	0.285	0.775	-5.967	7.998
X283	0.2414	1.259	0.192	0.848	-2.227	2.709
X284	2.5557	2.592	0.986	0.324	-2.525	7.637
X285	1.5100	1.175	1.285	0.199	-0.793	3.813
X286	-0.3049	1.825	-0.167	0.867	-3.882	3.272
X287	0.6406	2.264	0.283	0.777	-3.797	5.078
X288	0.1277	8.754	0.015	0.988	-17.035	17.290
X291	-1.0271	1.359	-0.756	0.450	-3.691	1.637
X292	-2.7188	1.517	-1.792	0.073	-5.694	0.256
X294	0.3835	0.874	0.439	0.661	-1.330	2.097
X295	-3.6850	4.470	-0.824	0.410	-12.449	5.079
X296	-3.6850	4.470	-0.824	0.410	-12.449	5.079
X298	-0.4029	1.021	-0.394	0.693	-2.405	1.599
X299	-0.4029	1.021	-0.394	0.693	-2.405	1.599
X300	-0.7539	0.681	-1.107	0.268	-2.089	0.581
X301	2.1241	0.763	2.785	0.005	0.629	3.619
X302	-0.8078	3.507	-0.230	0.818	-7.683	6.068
X304	11.2278	8.260	1.359	0.174	-4.965	27.421
X305	3.8220	6.019	0.635	0.525	-7.979	15.623
X306	2.7574	5.639	0.489	0.625	-8.298	13.813
X307	-2.1305	5.121	-0.416	0.677	-12.170	7.909
X308	-1.8938	4.787	-0.396	0.692	-11.279	7.492
X309	4.8721	5.944	0.820	0.412	-6.782	16.527
X310	6.2883	2.455	2.561	0.010	1.475	11.102
X311	1.4064	1.852	0.759	0.448	-2.225	5.038
X312	-4.0347	9.017	-0.447	0.655	-21.714	13.644
X313	-0.9993	0.995	-1.005	0.315	-2.950	0.951
X314	5.3030	2.373	2.235	0.025	0.651	9.955
X315	16.2249	2.282	7.110	0.000	11.751	20.699
X316	-1.7490	0.987	-1.771	0.077	-3.685	0.187
X317	6.4272	4.559	1.410	0.159	-2.511	15.366
X318	-1.7510	6.858	-0.255	0.798	-15.196	11.694
X319	-2.4816	6.284	-0.395	0.693	-14.802	9.839
X320	-0.9138	1.878	-0.487	0.627	-4.596	2.768
X321	-0.2229	1.779	-0.125	0.900	-3.711	3.266
X322	1.1138	1.556	0.716	0.474	-1.936	4.164
X323	-1.0065	1.680	-0.599	0.549	-4.300	2.287
X324	-0.3913	0.612	-0.639	0.523	-1.592	0.809
X325	-0.2709	4.877	-0.056	0.956	-9.833	9.291
X326	1.0700	0.445	2.402	0.016	0.197	1.943
X327	1.2946	0.830	1.561	0.119	-0.332	2.921

X328	-1.4440	4.753	-0.304	0.761	-10.762	7.874
X329	-0.7409	2.195	-0.337	0.736	-5.045	3.563
X331	1.7530	9.829	0.178	0.858	-17.518	21.024
X332	-0.9684	4.919	-0.197	0.844	-10.613	8.676
X333	0.1599	2.622	0.061	0.951	-4.980	5.300
X334	3.4940	1.831	1.909	0.056	-0.095	7.083
X335	4.1694	2.778	1.501	0.134	-1.277	9.616
X336	-3.5224	1.689	-2.085	0.037	-6.834	-0.211
X337	6.5140	1.214	5.366	0.000	4.134	8.894
X338	-5.9605	2.167	-2.751	0.006	-10.208	-1.713
X339	41.5984	12.930	3.217	0.001	16.249	66.948
X340	-1.0842	1.364	-0.795	0.427	-3.758	1.590
X341	-2.6205	2.076	-1.262	0.207	-6.691	1.450
X342	-3.0373	1.874	-1.621	0.105	-6.711	0.636
X343	1.4415	2.439	0.591	0.555	-3.341	6.224
X344	-4.2527	2.980	-1.427	0.154	-10.096	1.590
X345	0.4356	1.857	0.235	0.815	-3.206	4.077
X346	-0.1273	2.195	-0.058	0.954	-4.430	4.176
X348	-5.5039	10.629	-0.518	0.605	-26.343	15.335
X349	2.6485	1.975	1.341	0.180	-1.223	6.521
X350	0.4586	0.580	0.791	0.429	-0.679	1.596
X351	-0.0783	0.556	-0.141	0.888	-1.169	1.012
X352	-6.3360	9.473	-0.669	0.504	-24.908	12.236
X353	2.4766	3.363	0.736	0.461	-4.116	9.069
X354	0.0613	0.557	0.110	0.912	-1.031	1.153
X355	-0.1849	1.260	-0.147	0.883	-2.655	2.285
X356	0.4021	2.217	0.181	0.856	-3.944	4.748
X357	-3.0145	8.722	-0.346	0.730	-20.114	14.085
X358	0.7208	1.367	0.527	0.598	-1.960	3.401
X359	-1.0940	0.992	-1.103	0.270	-3.038	0.850
X360	1.5364	1.158	1.327	0.184	-0.733	3.806
X361	0.6911	1.572	0.440	0.660	-2.391	3.773
X362	-5.9384	6.812	-0.872	0.383	-19.295	7.418
X363	-2.3713	2.163	-1.097	0.273	-6.611	1.868
X364	3.1495	1.244	2.532	0.011	0.711	5.588
X365	3.1495	1.244	2.532	0.011	0.711	5.588
X366	-5.4202	6.261	-0.866	0.387	-17.695	6.855
X367	1.9964	3.555	0.562	0.574	-4.973	8.966
X368	4.6468	2.118	2.194	0.028	0.495	8.799
X369	1.4932	3.258	0.458	0.647	-4.894	7.880
X370	-2.4953	4.135	-0.603	0.546	-10.603	5.612
X371	1.0789	2.906	0.371	0.710	-4.618	6.776
X372	-0.4847	4.413	-0.110	0.913	-9.136	8.167
X373	2.2620	2.510	0.901	0.368	-2.659	7.183
X374	2.8080	1.674	1.677	0.094	-0.474	6.090
X375	5.2794	1.216	4.342	0.000	2.895	7.663
X376	6.3490	2.400	2.646	0.008	1.644	11.054

```
X377          1.9473      1.896       1.027      0.305      -1.771       5.665
X378         -0.6014      1.927      -0.312      0.755      -4.380       3.177
X379          2.4868      2.634       0.944      0.345      -2.677       7.650
X380          0.5422      2.487       0.218      0.827      -4.333       5.418
X382         -0.1173      1.629      -0.072      0.943      -3.311       3.077
X383          9.0168      3.710       2.431      0.015       1.744      16.290
X384          0.1948      7.893       0.025      0.980     -15.279      15.669
X385         -1.3332      1.051      -1.269      0.205      -3.394       0.727
==============================================================================
Omnibus:                     4021.263   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           484285.739
Skew:                           4.226   Prob(JB):                         0.00
Kurtosis:                      54.865   Cond. No.                     1.31e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 4.51e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

[ ]:

[338]:
```
# Lot of Columns have P-Value > 0.05.

# These columns are not required to Predict Target.

# We can try Building the Model Without these Columns.
```

[ ]:

[339]:
```
results.pvalues
```

[339]:
```
X0      0.000014
X1      0.275892
X2      0.876181
X3      0.327985
X4      0.943520
          …
X380    0.827400
X382    0.942622
X383    0.015117
X384    0.980310
X385    0.204673
Length: 364, dtype: float64
```

[340]:
```
results.pvalues[results.pvalues < 0.05]
```

[340]:
```
X0      0.000014
X5      0.002022
X45     0.003403
X47     0.015723
X48     0.029690
          …
X365    0.011386
X368    0.028281
X375    0.000014
X376    0.008187
X383    0.015117
Length: 70, dtype: float64
```

[341]:
```
(results.pvalues[results.pvalues < 0.05]).index
```

[341]:
```
Index(['X0', 'X5', 'X45', 'X47', 'X48', 'X52', 'X54', 'X74', 'X75', 'X76',
       'X79', 'X95', 'X104', 'X111', 'X113', 'X115', 'X117', 'X118', 'X119',
       'X120', 'X123', 'X128', 'X130', 'X133', 'X134', 'X136', 'X142', 'X143',
       'X147', 'X152', 'X156', 'X157', 'X158', 'X163', 'X174', 'X178', 'X179',
       'X180', 'X186', 'X189', 'X194', 'X201', 'X204', 'X206', 'X209', 'X210',
       'X217', 'X222', 'X226', 'X236', 'X240', 'X249', 'X250', 'X263', 'X272',
       'X301', 'X310', 'X314', 'X315', 'X326', 'X336', 'X337', 'X338', 'X339',
       'X364', 'X365', 'X368', 'X375', 'X376', 'X383'],
      dtype='object')
```

[342]:
```
significant_cols = list((results.pvalues[results.pvalues < 0.05]).index)
```

[343]:
```
len(significant_cols)
```

[343]:
```
70
```

[344]:
```
# we will Build our Model Using these 70 Features Only.
```

[372]:
```
new_x = x[significant_cols].copy()
```

[373]:
```
new_x.shape
```

[373]:
```
(4209, 70)
```

[374]:
```
new_x.head()
```

[374]:

ID	X0	X5	X45	X47	X48	X52	X54	X74	X75	X76	…	X336	X337	X338	\
											…				
0	32	24	0	0	0	0	0	1	0	0	…	0	0	0	
6	32	28	0	0	0	0	0	1	0	0	…	1	1	0	
7	20	27	0	0	0	0	1	1	1	1	…	0	0	0	

```
9   20  27    0    0    0    0    1    1    0    1  …     0     0     0
13  20  12    0    0    0    0    1    1    0    1  …     0     0     0

    X339  X364  X365  X368  X375  X376  X383
ID
0      0     0     0     0     0     0     0
6      0     0     0     0     1     0     0
7      0     0     0     0     0     0     0
9      0     0     0     0     0     0     0
13     0     0     0     0     0     0     0

[5 rows x 70 columns]
```

[ ]:

[ ]:

1.0.7 Scalling:

- While Mostly all Categorical Variables have Values of 0 and 1, Variables that we did Label Encoder on have Multiple Values. So, it is Possible that Machine Learning Models will give More Importance to them as They have Values Greater Than 1. So, we will Scale the dataset.

[376]:
```
sc= StandardScaler()
```

[378]:
```
temp = sc.fit_transform(new_x)
scaled_x = pd.DataFrame(temp, index=new_x.index, columns= new_x.columns)
scaled_x.head()
```

[378]:
```
           X0        X5       X45       X47       X48       X52       X54  \
ID
0    0.163012  1.292117 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
6    0.163012  1.776974 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
7   -0.710560  1.655760 -0.58238 -0.114002 -0.15114 -0.210138  4.690416
9   -0.710560  1.655760 -0.58238 -0.114002 -0.15114 -0.210138  4.690416
13 -0.710560 -0.162454 -0.58238 -0.114002 -0.15114 -0.210138  4.690416

          X74       X75       X76  …      X336      X337      X338      X339  \
ID                                  …
0    0.026707 -0.193562 -0.213201  … -0.382008 -1.033588 -0.083293 -0.015416
6    0.026707 -0.193562 -0.213201  …  2.617749  0.967503 -0.083293 -0.015416
7    0.026707  5.166313  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416
9    0.026707 -0.193562  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416
13   0.026707 -0.193562  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416

         X364      X365      X368      X375      X376      X383
ID
0   -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
```

```
6  -0.053471 -0.053471 -0.258689  1.461630 -0.246447 -0.040815
7  -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
9  -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
13 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815

[5 rows x 70 columns]
```

[ ]:

1.0.8 Dimensionality Reduction Using PCA:

[379]:
```
pca = PCA(n_components= 0.99)
```

[380]:
```
pca_result = pca.fit_transform(scaled_x)
```

[381]:
```
pca_result
```

[381]:
```
array([[ 7.38566730e-01, -6.06310781e-01,  8.42687902e-01, …,
        -4.62081770e-01,  2.71190481e-02,  9.29520826e-01],
       [-8.77990014e-01, -5.24783033e-01,  1.46480126e+00, …,
        -1.43627754e-01, -6.13814733e-02, -4.77159197e-02],
       [ 2.53123660e+00,  1.19378448e+01, -6.82449209e-02, …,
         2.16114663e+00,  1.09025426e-01, -1.47043665e+00],
       …,
       [-3.26179928e-01, -2.04293626e-01,  1.55053118e-01, …,
         4.63004956e-01, -1.12608503e-01, -6.64108647e-02],
       [-9.78535047e-01, -2.41528421e-02,  3.96064695e-01, …,
        -5.87925831e-01,  1.57913682e-01, -1.72809030e-02],
       [-1.18629764e+00, -1.15946417e+00, -5.89086902e-01, …,
        -5.31059588e-01,  6.75673743e-03, -1.82552487e-01]])
```

[383]:
```
pca_result.shape
```

[383]:
```
(4209, 43)
```

[384]:
```
pca.explained_variance_ratio_
```

[384]:
```
array([0.15670056, 0.09494875, 0.05538113, 0.05306349, 0.04308408,
       0.04106288, 0.03696266, 0.0345388 , 0.0312952 , 0.0299603 ,
       0.02534326, 0.02456276, 0.02115288, 0.02014233, 0.0180675 ,
       0.01737717, 0.01612341, 0.01576077, 0.01525681, 0.0150391 ,
       0.01435188, 0.01425306, 0.01414013, 0.01403319, 0.01383682,
       0.01347849, 0.01306786, 0.0128261 , 0.01236534, 0.011939  ,
       0.01156939, 0.01028064, 0.0097577 , 0.00909089, 0.00900305,
       0.0082925 , 0.00732237, 0.00633129, 0.00478431, 0.00459924,
       0.00432563, 0.00414165, 0.00326093])
```

[385]:
```
np.sum(pca.explained_variance_ratio_)
```

[385]:
```
0.9928753048841577
```

[ ]:

1.0.9 Train Test Split:

[386]:
```
pca_result.shape
```

[386]:
```
(4209, 43)
```

[387]:
```
y.shape
```

[387]:
```
(4209,)
```

[ ]:

[388]:
```
x_train, x_test, y_train, y_test = train_test_split(pca_result, y, test_size= 0.
↪2, random_state= 42)
```

[390]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 43)
(842, 43)
(3367,)
(842,)
```

[393]:
```
x_train
```

[393]:
```
array([[ 0.01935025,  0.46122708, -0.07545592, …, -0.08958546,
         0.20525837, -0.31718347],
       [ 0.50260866, -0.11270403,  0.42246925, …, -0.53121479,
        -0.34511288, -0.69113605],
       [-0.45737264,  0.640639  , -0.80102688, …,  0.20973414,
        -0.00425784, -0.10340277],
       …,
       [-1.6878898 , -0.8662463 , -0.24779434, …,  0.27311174,
        -0.09071486, -0.20574805],
       [-1.46444445, -0.142273  , -1.02934594, …,  0.30099518,
        -0.09149417, -0.02939956],
       [-1.01640424, -1.32260175, -0.05691751, …, -0.42583111,
         0.08116195, -0.12648478]])
```

[394]:
```
y_train
```

[394]:
```
ID
2011     88.96
3690     89.90
7597     92.59
322     108.84
3103    111.15
         …
6879    109.42
898      78.25
6214     92.18
7558     91.92
1712     87.71
Name: y, Length: 3367, dtype: float64
```

[395]:
```
x_test
```

[395]:
```
array([[-1.4129803 ,  0.3257658 ,  0.95520315, …, -1.71545701,
         0.28846419, -0.02113956],
       [-0.72302668, -0.28694555,  0.2555645 , …, -0.15784824,
         0.02061572,  0.35074088],
       [ 1.52635451, -0.96242301, -0.19938882, …,  0.41793905,
        -0.00647589, -0.22711589],
       …,
       [ 1.31727029, -0.89018872, -0.66790561, …,  0.18478042,
         0.01967501, -0.32233412],
       [-0.45551154,  0.64007108, -0.80133001, …,  0.20805844,
        -0.0041023 , -0.10448369],
       [-0.51235256, -1.54242536,  0.35736107, …, -0.85119515,
         0.12623405,  0.33733054]])
```

[396]:
```
y_test
```

[396]:
```
ID
2140     97.94
310      96.41
4779    105.83
385      79.09
5180    108.69
         …
1280    113.68
7972     88.85
1810     89.60
7206     89.23
3922    109.49
Name: y, Length: 842, dtype: float64
```

[ ]:
```

```

1.0.10 XGBRegressor Model on PCA Data:

[397]:
```
XGBR_Model_1 = XGBRegressor(n_estimators=300, max_depth=3, learning_rate=0.02,␣
↪random_state=100, min_child_weight=1,
                            colsample_bytree=0.4, alpha=10)
```

[398]:
```
XGBR_Model_1.fit(x_train, y_train)
```

[398]:
```
XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=300, n_jobs=4, num_parallel_tree=1, random_state=100,
             reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[399]:
```
pred = XGBR_Model_1.predict(x_test)
```

[400]:
```
r2_score(y_test, pred)
```

[400]:
```
0.5309834096629831
```

[ ]:
```

```

Grid Search Below Takes around 3 Hours to run as I have Checked Many Combinations of Parameters and Data

If you are Following this Notebook, it's better if you just take Best Parameters Found from Grid Search below an Use Those Parameters to Build Model rather than Running full Grid Search.

[ ]:
```

```

[404]:
```
xgbr = XGBRegressor()
```

[403]:
```
param_dict = {
    'n_estimators': [100,200,300,500,1000],
    'max_depth': range(3, 6),
    'learning_rate': [0.01, 0.02, 0.1, 0.5, 0.75, 1],
    'colsample_bytree': [0.4,0.5,0.6, 0.7, 0.8, 0.9, 1]}
```

[405]:
```
grid_xgbr = GridSearchCV(estimator= xgbr, param_grid= param_dict, cv= 5)
```

[406]:
```
grid_xgbr.fit(x_train, y_train)
```

[406]:
```
GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
```

```
                                   gpu_id=None, importance_type='gain',
                                   interaction_constraints=None,
                                   learning_rate=None, max_delta_step=None,
                                   max_depth=None, min_child_weight=None,
                                   missing=nan, monotone_constraints=None,
                                   n_estimators=100, n_jobs=None,
                                   num_parallel_tree=None, random_state=None,
                                   reg_alpha=None, reg_lambda=None,
                                   scale_pos_weight=None, subsample=None,
                                   tree_method=None, validate_parameters=None,
                                   verbosity=None),
             param_grid={'colsample_bytree': [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
                         'learning_rate': [0.01, 0.02, 0.1, 0.5, 0.75, 1],
                         'max_depth': range(3, 6),
                         'n_estimators': [100, 200, 300, 500, 1000]})
```

[407]:
```
grid_xgbr.best_estimator_
```

[407]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[408]:
```
grid_xgbr.best_score_
```

[408]:
```
0.49330473622053717
```

[409]:
```
r2_score(y_test, grid_xgbr.best_estimator_.predict(x_test))
```

[409]:
```
0.5505314980737926
```

[ ]:
```

```

[411]:
```
# XGB Regressor on PCA Data Doesn't Give Propre Accuracy.

# We will try to build XGB Regressor on 70 Features selected by OLS.
```

[ ]:
```

```

1.0.11 XGBRegressor Model on Features selected using OLS:

[546]:
```
# We have 70 Featues Selected Using OLS scaled above in scaled_x data frame (on
 ↪Which we performed PCA.)
# We can Use scaled_x data frame here.
```

[412]:
```
scaled_x.head()
```

[412]:
```
           X0        X5       X45       X47       X48       X52       X54  \
ID
0    0.163012  1.292117 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
6    0.163012  1.776974 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
7   -0.710560  1.655760 -0.58238 -0.114002 -0.15114 -0.210138  4.690416
9   -0.710560  1.655760 -0.58238 -0.114002 -0.15114 -0.210138  4.690416
13 -0.710560 -0.162454 -0.58238 -0.114002 -0.15114 -0.210138  4.690416

          X74       X75       X76  …      X336      X337      X338      X339  \
ID                                  …
0    0.026707 -0.193562 -0.213201  … -0.382008 -1.033588 -0.083293 -0.015416
6    0.026707 -0.193562 -0.213201  …  2.617749  0.967503 -0.083293 -0.015416
7    0.026707  5.166313  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416
9    0.026707 -0.193562  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416
13   0.026707 -0.193562  4.690416  … -0.382008 -1.033588 -0.083293 -0.015416

         X364      X365      X368      X375      X376      X383
ID
0   -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
6   -0.053471 -0.053471 -0.258689  1.461630 -0.246447 -0.040815
7   -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
9   -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
13 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815

[5 rows x 70 columns]
```

[413]:
```
scaled_x.shape
```

[413]:
```
(4209, 70)
```

[414]:
```
y.head()
```

[414]:
```
ID
0     130.81
6      88.53
7      76.26
9      80.62
13     78.02
Name: y, dtype: float64
```

[415]:
```
y.shape
```

[415]:
```
(4209,)
```

[ ]:
```

```

[421]:
```
# Train Test Split:
```

[422]:
```
x_train, x_test, y_train, y_test = train_test_split(scaled_x, y, test_size= 0.
↪2, random_state= 42)
```

[423]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 70)
(842, 70)
(3367,)
(842,)
```

[427]:
```
x_train.head()
```

[427]:

ID	X0	X5	X45	X47	X48	X52	X54	\
2011	0.308607	-0.889740	-0.582380	-0.114002	-0.15114	-0.210138	-0.213201	
3690	0.381405	-1.132169	-0.582380	-0.114002	-0.15114	-0.210138	-0.213201	
7597	-0.200976	1.413332	-0.582380	-0.114002	-0.15114	-0.210138	-0.213201	
322	0.090214	-0.041240	-0.582380	-0.114002	-0.15114	-0.210138	-0.213201	
3103	-0.783357	-1.253383	1.717092	-0.114002	-0.15114	-0.210138	-0.213201	

ID	X74	X75	X76	…	X336	X337	X338	\
				…				
2011	0.026707	-0.193562	-0.213201	…	-0.382008	-1.033588	-0.083293	
3690	0.026707	-0.193562	-0.213201	…	2.617749	0.967503	-0.083293	
7597	0.026707	-0.193562	-0.213201	…	-0.382008	-1.033588	-0.083293	
322	0.026707	-0.193562	-0.213201	…	-0.382008	-1.033588	-0.083293	
3103	0.026707	-0.193562	-0.213201	…	-0.382008	-1.033588	-0.083293	

ID	X339	X364	X365	X368	X375	X376	X383
2011	-0.015416	-0.053471	-0.053471	-0.258689	-0.684167	-0.246447	-0.040815
3690	-0.015416	-0.053471	-0.053471	3.865641	1.461630	-0.246447	-0.040815
7597	-0.015416	-0.053471	-0.053471	-0.258689	-0.684167	-0.246447	-0.040815
322	-0.015416	-0.053471	-0.053471	-0.258689	-0.684167	4.057675	-0.040815
3103	-0.015416	-0.053471	-0.053471	-0.258689	-0.684167	-0.246447	-0.040815

[5 rows x 70 columns]

[428]:
```
y_train.head()
```

[428]:
```
ID
2011     88.96
3690     89.90
7597     92.59
322     108.84
3103    111.15
Name: y, dtype: float64
```

[429]:
```
x_test.head()
```

[429]:
```
            X0        X5      X45       X47      X48       X52       X54  \
ID
2140 -1.511334 -0.889740 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
310  -0.200976 -0.041240 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201
4779  0.090214  0.564832 -0.58238 -0.114002 -0.15114  4.758788 -0.213201
385  -0.710560 -0.041240 -0.58238 -0.114002 -0.15114 -0.210138  4.690416
5180 -1.584132  0.443617 -0.58238 -0.114002 -0.15114 -0.210138 -0.213201

           X74       X75       X76  …      X336      X337      X338  \
ID                                   …
2140  0.026707 -0.193562 -0.213201  … -0.382008  0.967503 -0.083293
310   0.026707 -0.193562 -0.213201  … -0.382008  0.967503 -0.083293
4779  0.026707 -0.193562 -0.213201  … -0.382008  0.967503 -0.083293
385   0.026707 -0.193562  4.690416  … -0.382008 -1.033588 -0.083293
5180  0.026707 -0.193562 -0.213201  … -0.382008 -1.033588 -0.083293

          X339      X364      X365      X368      X375      X376      X383
ID
2140 -0.015416 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
310  -0.015416 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
4779 -0.015416 -0.053471 -0.053471 -0.258689  1.461630 -0.246447 -0.040815
385  -0.015416 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815
5180 -0.015416 -0.053471 -0.053471 -0.258689 -0.684167 -0.246447 -0.040815

[5 rows x 70 columns]
```

[430]:
```
y_test.head()
```

[430]:
```
ID
2140     97.94
310      96.41
4779    105.83
385      79.09
5180    108.69
Name: y, dtype: float64
```

[ ]:

[431]:
```
XGBR_Model_2= XGBRegressor(base_score=0.5, booster='gbtree',␣
↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[432]:
```
XGBR_Model_2.fit(x_train, y_train)
```

[432]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[433]:
```
pred = XGBR_Model_2.predict(x_test)
```

[434]:
```
r2_score(y_test, pred)
```

[434]:
```
0.6007262412395912
```

[ ]:

[435]:
```
# Still, Not decent Accuracy.

# We will Try to Build Model on all the features that were in data just before␣
↪performing OLS.
```

[ ]:

1.0.12 XGBRegressor Model on all Features:

[547]:
```
# We are going to take all the features available after dropping features with␣
↪Zero Variance.
# Those Features are stored in df dataframe, so we will use df here to create x.
```

[438]:
```
x = df.drop("y", axis= 1)
y = df["y"]
```

[439]:
```
x.head()
```

[439]:
```
    X0  X1  X2  X3  X4  X5  X6  X8  X10  X12  …  X375  X376  X377  X378  \
ID                                             …
0   32  23  17   0   3  24   9  14    0    0  …     0     0     1     0
6   32  21  19   4   3  28  11  14    0    0  …     1     0     0     0
7   20  24  34   2   3  27   9  23    0    0  …     0     0     0     0
9   20  21  34   5   3  27  11   4    0    0  …     0     0     0     0
13  20  23  34   5   3  12   3  13    0    0  …     0     0     0     0

    X379  X380  X382  X383  X384  X385
ID
0      0     0     0     0     0     0
6      0     0     0     0     0     0
7      0     0     1     0     0     0
9      0     0     0     0     0     0
13     0     0     0     0     0     0

[5 rows x 364 columns]
```

[440]:
```
y.head()
```

[440]:
```
ID
0     130.81
6      88.53
7      76.26
9      80.62
13     78.02
Name: y, dtype: float64
```

[ ]:
```

```

[451]:
```
# Scalling:
```

[452]:
```
sc = StandardScaler()
```

[453]:
```
temp = sc.fit_transform(x)
scaled_x_all_features = pd.DataFrame(temp, index=x.index, columns= x.columns)
scaled_x_all_features.head()
```

[453]:
```
          X0        X1        X2        X3        X4        X5        X6  \
ID
0   0.163012  1.393488 -0.028122 -1.678270  0.028938  1.292117  0.751787
6   0.163012  1.159021  0.155388  0.620969  0.028938  1.776974  1.437511
7  -0.710560  1.510721  1.531709 -0.528650  0.028938  1.655760  0.751787
9  -0.710560  1.159021  1.531709  1.195779  0.028938  1.655760  1.437511
13 -0.710560  1.393488  1.531709  1.195779  0.028938 -0.162454 -1.305384
```

```
          X8       X10       X12  …      X375      X376      X377     X378  \
ID                                  …
0   0.339445 -0.116122 -0.284906  … -0.684167 -0.246447  1.475332 -0.14528
6   0.339445 -0.116122 -0.284906  …  1.461630 -0.246447 -0.677814 -0.14528
7   1.618389 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814 -0.14528
9  -1.081605 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814 -0.14528
13  0.197340 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814 -0.14528

        X379      X380       X382      X383      X384      X385
ID
0  -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
6  -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
7  -0.097952 -0.090243  11.425027 -0.040815 -0.021804 -0.037783
9  -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
13 -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783

[5 rows x 364 columns]
```

[ ]:

[454]:
```
# Train Test Split:

x_train, x_test, y_train, y_test = train_test_split(scaled_x_all_features, y,␣
 ↪test_size= 0.2, random_state= 42)
```

[455]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 364)
(842, 364)
(3367,)
(842,)
```

[456]:
```
x_train.head()
```

[456]:
```
            X0        X1        X2       X3        X4        X5        X6  \
ID
2011  0.308607  1.393488 -0.119876 -0.52865  0.028938 -0.889740  1.094649
3690  0.381405  1.041787 -0.119876  0.04616  0.028938 -1.132169 -0.276798
7597 -0.200976 -0.833948  1.439954 -0.52865  0.028938  1.413332  0.408925
322   0.090214 -1.185648  0.889426  0.04616  0.028938 -0.041240  0.408925
3103 -0.783357 -0.130547 -0.119876 -0.52865  0.028938 -1.253383  1.437511

            X8       X10       X12  …      X375      X376      X377  \
ID                                  …
2011  1.618389 -0.116122 -0.284906  … -0.684167 -0.246447  1.475332
```

```
3690  0.907865 -0.116122 -0.284906  …  1.461630 -0.246447 -0.677814
7597 -1.081605 -0.116122 -0.284906  … -0.684167 -0.246447  1.475332
322  -1.081605 -0.116122 -0.284906  … -0.684167  4.057675 -0.677814
3103 -0.228975 -0.116122 -0.284906  … -0.684167 -0.246447  1.475332

         X378      X379      X380      X382      X383      X384      X385
ID
2011 -0.14528 -0.097952 -0.090243 -0.087527 -0.040815 -0.021804 -0.037783
3690 -0.14528 -0.097952 -0.090243 -0.087527 -0.040815 -0.021804 -0.037783
7597 -0.14528 -0.097952 -0.090243 -0.087527 -0.040815 -0.021804 -0.037783
322  -0.14528 -0.097952 -0.090243 -0.087527 -0.040815 -0.021804 -0.037783
3103 -0.14528 -0.097952 -0.090243 -0.087527 -0.040815 -0.021804 -0.037783

[5 rows x 364 columns]
```

[457]:
```
y_train.head()
```

[457]:
```
ID
2011     88.96
3690     89.90
7597     92.59
322     108.84
3103    111.15
Name: y, dtype: float64
```

[458]:
```
x_test.head()
```

[458]:
```
            X0        X1        X2        X3        X4        X5        X6  \
ID
2140 -1.511334  0.572854 -0.945669  1.195779  0.028938 -0.889740  0.751787
310  -0.200976  0.221153 -1.312688  1.195779  0.028938 -0.041240  0.408925
4779  0.090214 -1.185648  0.338897 -0.528650  0.028938  0.564832  1.437511
385  -0.710560  1.627955  0.430652 -0.528650  0.028938 -0.041240  0.751787
5180 -1.584132  1.393488 -0.853914  0.046160  0.028938  0.443617  0.408925

            X8       X10       X12  …      X375      X376      X377  \
ID                                   …
2140 -0.086870 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814
310   1.476285 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814
4779  0.339445  8.611662 -0.284906  …  1.461630 -0.246447 -0.677814
385  -0.086870 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814
5180  0.765760 -0.116122 -0.284906  … -0.684167 -0.246447 -0.677814

         X378      X379      X380       X382      X383      X384      X385
ID
2140 -0.14528 -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
310  -0.14528 -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
```

```
4779 -0.14528 -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783
385  -0.14528 -0.097952 -0.090243  11.425027 -0.040815 -0.021804 -0.037783
5180 -0.14528 -0.097952 -0.090243  -0.087527 -0.040815 -0.021804 -0.037783

[5 rows x 364 columns]
```

[459]:
```
y_test.head()
```

[459]:
```
ID
2140     97.94
310      96.41
4779    105.83
385      79.09
5180    108.69
Name: y, dtype: float64
```

[ ]:

[460]:
```
XGBR_Model_3= XGBRegressor(base_score=0.5, booster='gbtree',␣
↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[461]:
```
XGBR_Model_3.fit(x_train, y_train)
```

[461]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[462]:
```
pred = XGBR_Model_3.predict(x_test)
```

[463]:
```
r2_score(y_test, pred)
```

[463]:
```
0.5971265803872543
```

[ ]:

[464]:
```
# Again, Not very Decent.

# We will try to Build XGB Regressor on all features after Converting␣
↪Categorical Variable into Dummy Variables.
```

[ ]:
```

```

1.0.13 XGBRegressor Model on all Features after Getting Dummy Variables for Categorical Features:

[548]:
```
# In Above Model, Our x had all the features which we then scaled to get␣
↪scaled_x_all_features data frame.
# We will use x with all unscaled features from above to create dummy variables␣
↪from categorical ones.
```

[504]:
```
x.head()
```

[504]:

ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	…	X375	X376	X377	X378	\
0	32	23	17	0	3	24	9	14	0	0	…	0	0	1	0	
6	32	21	19	4	3	28	11	14	0	0	…	1	0	0	0	
7	20	24	34	2	3	27	9	23	0	0	…	0	0	0	0	
9	20	21	34	5	3	27	11	4	0	0	…	0	0	0	0	
13	20	23	34	5	3	12	3	13	0	0	…	0	0	0	0	

ID	X379	X380	X382	X383	X384	X385
0	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	1	0	0	0
9	0	0	0	0	0	0
13	0	0	0	0	0	0

[5 rows x 364 columns]

[505]:
```
columns_to_dummy = []
for col in x.columns:
    if df[col].nunique() > 2:
        print(col)
        columns_to_dummy.append(col)
```

```
X0
X1
X2
X3
X4
X5
```

```
X6
X8
```

[506]:
```
# Only This columns have Multiple Values in them, other features already have Binary Data.
```

[507]:
```
columns_to_dummy
```

[507]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[508]:
```
x_dummy = pd.get_dummies(x, columns= columns_to_dummy, drop_first= True)
```

[509]:
```
x_dummy.head()
```

[509]:

ID	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	…	X8_15	X8_16	\
0	0	0	1	0	0	0	0	1	0	0	…	0	0	
6	0	0	0	0	0	0	0	1	0	0	…	0	0	
7	0	0	0	0	0	0	1	0	0	0	…	0	0	
9	0	0	0	0	0	0	0	0	0	0	…	0	0	
13	0	0	0	0	0	0	0	0	0	0	…	0	0	

ID	X8_17	X8_18	X8_19	X8_20	X8_21	X8_22	X8_23	X8_24
0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0

[5 rows x 543 columns]

[ ]:
```

```

[510]:
```
# We don't need to Scale this Data as all columns have binary data now.
```

[511]:
```
# Train Test Split:

x_train, x_test, y_train, y_test = train_test_split(x_dummy, y, test_size= 0.2, random_state= 42)
```

[512]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 543)
(842, 543)
```

```
(3367,)
(842,)
```

[513]:
```
x_train.head()
```

[513]:
```
      X10  X12  X13  X14  X15  X16  X17  X18  X19  X20  …  X8_15  X8_16  \
ID                                                      …
2011    0    0    1    1    0    0    0    0    0    0  …      0      0
3690    0    0    0    1    0    0    0    0    0    0  …      0      0
7597    0    0    0    0    0    0    0    0    0    0  …      0      0
322     0    0    0    0    0    0    0    0    0    0  …      0      0
3103    0    0    0    1    0    0    0    0    0    0  …      0      0

      X8_17  X8_18  X8_19  X8_20  X8_21  X8_22  X8_23  X8_24
ID
2011      0      0      0      0      0      0      1      0
3690      0      1      0      0      0      0      0      0
7597      0      0      0      0      0      0      0      0
322       0      0      0      0      0      0      0      0
3103      0      0      0      0      0      0      0      0

[5 rows x 543 columns]
```

[514]:
```
y_train.head()
```

[514]:
```
ID
2011     88.96
3690     89.90
7597     92.59
322     108.84
3103    111.15
Name: y, dtype: float64
```

[515]:
```
x_test.head()
```

[515]:
```
      X10  X12  X13  X14  X15  X16  X17  X18  X19  X20  …  X8_15  X8_16  \
ID                                                      …
2140    0    0    0    0    0    0    0    0    1    0  …      0      0
310     0    0    0    0    0    0    0    0    0    1  …      0      0
4779    1    0    0    0    0    0    0    0    0    0  …      0      0
385     0    0    0    0    0    0    1    0    0    0  …      0      0
5180    0    0    0    0    0    0    0    0    0    1  …      0      0

      X8_17  X8_18  X8_19  X8_20  X8_21  X8_22  X8_23  X8_24
ID
2140      0      0      0      0      0      0      0      0
310       0      0      0      0      0      1      0      0
4779      0      0      0      0      0      0      0      0
```

```
385       0      0      0      0      0      0      0      0
5180      1      0      0      0      0      0      0      0

[5 rows x 543 columns]
```

[516]:
```
y_test.head()
```

[516]:
```
ID
2140     97.94
310      96.41
4779    105.83
385      79.09
5180    108.69
Name: y, dtype: float64
```

[ ]:
```

```

[517]:
```
XGBR_Model_4= XGBRegressor(base_score=0.5, booster='gbtree',␣
↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[518]:
```
XGBR_Model_4.fit(x_train, y_train)
```

[518]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[519]:
```
pred = XGBR_Model_4.predict(x_test)
```

[520]:
```
r2_score(y_test, pred)
```

[520]:
```
0.5982197068074213
```

[ ]:
```

```

[521]:
```
# Let's See if we can use PCA on data with dummy variables to reduce dimensions␣
↪and still get accuracy close to this.
```

[ ]:

[522]:
```
# PCA:
```

[528]:
```
pca_2 = PCA(n_components= 0.99)
```

[529]:
```
pca_2_result = pca_2.fit_transform(x_dummy)
```

[530]:
```
pca_2_result
```

[530]:
```
array([[ 8.17092193e-01, -1.35926461e+00,  1.93980323e+00, …,
        -1.22764591e-01,  2.80940435e-02, -2.22326646e-01],
       [-1.01498528e-01, -1.29439536e+00, -9.14891423e-02, …,
         2.30191387e-01,  1.91058288e-03, -1.77685773e-01],
       [-6.68523927e-01, -2.43478060e+00,  1.69808551e+00, …,
         3.66251606e-03, -8.76945321e-02, -1.19278619e-01],
       …,
       [-1.03441762e+00, -4.84180675e-01,  1.80068794e+00, …,
        -3.64025730e-02, -1.41229392e-02, -2.88435861e-02],
       [ 3.90266235e-01, -1.17253942e+00, -3.10414176e+00, …,
        -8.96313589e-02,  3.32681750e-02,  1.99912401e-02],
       [ 9.63564876e-01, -9.00336985e-01, -9.06489023e-01, …,
        -2.70507554e-02, -2.92804705e-02, -2.50616125e-02]])
```

[531]:
```
pca_2_result.shape
```

[531]:
```
(4209, 218)
```

[533]:
```
np.sum(pca_2.explained_variance_ratio_)
```

[533]:
```
0.9901745951185926
```

[ ]:

[534]:
```
### Train Test Split:

pca_2_result.shape
```

[534]:
```
(4209, 218)
```

[535]:
```
y.shape
```

[535]:
```
(4209,)
```

[536]:
```
x_train, x_test, y_train, y_test = train_test_split(pca_2_result, y, test_size=␣
↪0.2, random_state= 42)
```

[537]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 218)
(842, 218)
(3367,)
(842,)
```

[538]:
```
x_train
```

[538]:
```
array([[-1.4278624 ,  0.64548255,  2.29645895, …, -0.138267  ,
        -0.11193458,  0.03215213],
       [-1.93015669, -0.29011213,  0.59183733, …,  0.29790631,
         0.10102458,  0.2768754 ],
       [ 0.85510938, -0.80627092,  2.2602882 , …, -0.0145267 ,
         0.02536868, -0.00437477],
       …,
       [ 1.17073239, -1.38305776, -1.77050338, …,  0.05092168,
        -0.16211578, -0.06952964],
       [ 0.53356227, -0.69417238, -2.12762357, …,  0.06070186,
         0.05843849,  0.0203837 ],
       [ 2.23176552, -0.68529378, -0.45970559, …,  0.02002003,
        -0.11532239, -0.00550201]])
```

[539]:
```
x_test
```

[539]:
```
array([[ 1.34039212, -1.89903008, -2.6847891 , …, -0.07946088,
        -0.0374303 , -0.03027228],
       [-0.4169664 , -2.80610142,  0.1241259 , …, -0.02693305,
        -0.06557611, -0.00796897],
       [ 0.68685357,  0.35114241, -0.25831204, …, -0.07884676,
        -0.07603062,  0.06455336],
       …,
       [ 1.01770431, -0.18906457, -0.41868967, …, -0.0640775 ,
        -0.05065421, -0.02059913],
       [-0.11958256, -1.23867553,  2.38847224, …,  0.01667588,
        -0.02003622,  0.00297176],
       [-1.29360794,  1.65573016, -1.33224764, …,  0.11602609,
        -0.11917746,  0.11274231]])
```

[540]:
```
y_train
```

[540]:
```
ID
2011     88.96
3690     89.90
7597     92.59
```

```
322     108.84
3103    111.15
         …
6879    109.42
898      78.25
6214     92.18
7558     91.92
1712     87.71
Name: y, Length: 3367, dtype: float64
```

[541]:
```
y_test
```

[541]:
```
ID
2140     97.94
310      96.41
4779    105.83
385      79.09
5180    108.69
         …
1280    113.68
7972     88.85
1810     89.60
7206     89.23
3922    109.49
Name: y, Length: 842, dtype: float64
```

[ ]:
```
```

[542]:
```
XGBR_Model_5= XGBRegressor(base_score=0.5, booster='gbtree',␣
↪colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=np.nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[543]:
```
XGBR_Model_5.fit(x_train, y_train)
```

[543]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
```

```
                tree_method='exact', validate_parameters=1, verbosity=None)
```

[544]:
```
pred = XGBR_Model_5.predict(x_test)
```

[545]:
```
r2_score(y_test, pred)
```

[545]:
```
0.5205444142381193
```

[ ]:
```

```

1.0.14 We got

- 55.05% Accuracy on XGBRegressor Model Built on 43 Features found from PCA (PCA on 70 Significant Features Obtained from OLS).
- 60.07% Accuracy on XGBRegressor Model Built on 70 Significant Features Obtained from OLS.
- 59.71% Accuracy on XGBRegressor Model Built on all 364 Features of Data (Scaled Features).
- 59.82% Accuracy on XGBRegressor Model Built on all features after getting Dummy Variables for Categorical Columns (Total 543 features).
- 52.05% Accuracy on XGBRegressor Model Built on 218 Features found from PCA(PCA on all features after getting Dummy Variables for Categorical Columns).

As we can see, XGBR_Model_2 is Best in both Accuracy and also using Less Features than Original Data.

So, We will Use XGBR_Model_2 to Make Predictions on Unseen Test Data.

[ ]:
```

```

Note: We Will have to Perform all the data cleaning and manipulation steps, that we performed on data that we used to build XGBR_Model_2 model, on Unseen Test Data to get Unseen Test Data in Same Form as Training Data for XGBR_Model_2.

[ ]:
```

```

1.1 Prediction on Unseen Test Data:

[557]:
```
unseen_test = pd.read_csv("test.csv")
```

[558]:
```
unseen_test.head()
```

[558]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	…	X375	X376	X377	X378	X379	X380	\
0	1	az	v	n	f	d	t	a	w	0	…	0	0	0	1	0	0	
1	2	t	b	ai	a	d	b	g	y	0	…	0	0	1	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	…	0	0	0	1	0	0	
3	4	az	l	n	f	d	z	l	n	0	…	0	0	0	1	0	0	
4	5	w	s	as	c	d	y	i	m	0	…	1	0	0	0	0	0	

```
   X382  X383  X384  X385
0     0     0     0     0
1     0     0     0     0
2     0     0     0     0
3     0     0     0     0
4     0     0     0     0

[5 rows x 377 columns]
```

[ ]:

[559]:
```
unseen_test.dtypes
```

[559]:
```
ID       int64
X0      object
X1      object
X2      object
X3      object
         …
X380     int64
X382     int64
X383     int64
X384     int64
X385     int64
Length: 377, dtype: object
```

[ ]:

1) Missing Value Treatment:

[560]:
```
unseen_test.isna().sum().sum()
```

[560]:
```
0
```

[ ]:

2) Setting "ID" Column as Data Frame Index:

[561]:
```
unseen_test = unseen_test.set_index("ID")
```

[562]:
```
unseen_test.head()
```

[562]:
```
    X0 X1  X2 X3 X4 X5 X6 X8  X10  X11  …  X375  X376  X377  X378  X379  \
ID                                       …
1   az  v   n  f  d  t  a  w    0    0  …     0     0     0     1     0
2    t  b  ai  a  d  b  g  y    0    0  …     0     0     1     0     0
3   az  v  as  f  d  a  j  j    0    0  …     0     0     0     1     0
4   az  l   n  f  d  z  l  n    0    0  …     0     0     0     1     0
```

```
5    w  s  as  c  d  y  i  m    0    0  …    1    0    0    0    0
```

ID	X380	X382	X383	X384	X385
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

[5 rows x 376 columns]

[ ]:

3) Applying Label Encoder on Categorical Features:

[566]:
```
# We already have "cat_col" list which have names of Categorical Columns to␣
↪Encode.

# We also have "le", object of Label Encoder.
```

[564]:
```
cat_col
```

[564]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[567]:
```
for col in cat_col:
    unseen_test[col] = le.fit_transform(unseen_test[col])
```

[568]:
```
unseen_test.dtypes
```

[568]:
```
X0      int32
X1      int32
X2      int32
X3      int32
X4      int32
        …
X380    int64
X382    int64
X383    int64
X384    int64
X385    int64
Length: 376, dtype: object
```

[ ]:

4) Dropping Columns with 0 Variance:

[569]:
```
cols_to_drop
```

[569]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

[570]:
```
# We dropped these columns From Training Data as it had no Variance.
# We will have to Drop thses columns from new data too, even if these columns␣
 ↪have variance in new data.
```

[572]:
```
unseen_test = unseen_test.drop(cols_to_drop, axis= 1)
```

[573]:
```
unseen_test.head()
```

[573]:

ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	…	X375	X376	X377	X378	\
											…					
1	21	23	34	5	3	26	0	22	0	0	…	0	0	0	1	
2	42	3	8	0	3	9	6	24	0	0	…	0	0	1	0	
3	21	23	17	5	3	0	9	9	0	0	…	0	0	0	1	
4	21	13	34	5	3	31	11	13	0	0	…	0	0	0	1	
5	45	20	17	2	3	30	8	12	0	0	…	1	0	0	0	

ID	X379	X380	X382	X383	X384	X385
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

[5 rows x 364 columns]

[574]:
```
unseen_test.shape
```

[574]:
```
(4209, 364)
```

[ ]:
```

```

5) Keeping only those Features which were found Significant to Target Variable by OLS:

[575]:
```
significant_cols
```

[575]:
```
['X0',
 'X5',
 'X45',
 'X47',
 'X48',
 'X52',
 'X54',
 'X74',
 'X75',
 'X76',
 'X79',
 'X95',
 'X104',
 'X111',
 'X113',
 'X115',
 'X117',
 'X118',
 'X119',
 'X120',
 'X123',
 'X128',
 'X130',
 'X133',
 'X134',
 'X136',
 'X142',
 'X143',
 'X147',
 'X152',
 'X156',
 'X157',
 'X158',
 'X163',
 'X174',
 'X178',
 'X179',
 'X180',
 'X186',
 'X189',
 'X194',
 'X201',
 'X204',
 'X206',
 'X209',
```

```
 'X210',
 'X217',
 'X222',
 'X226',
 'X236',
 'X240',
 'X249',
 'X250',
 'X263',
 'X272',
 'X301',
 'X310',
 'X314',
 'X315',
 'X326',
 'X336',
 'X337',
 'X338',
 'X339',
 'X364',
 'X365',
 'X368',
 'X375',
 'X376',
 'X383']
```

[576]:
```
len(significant_cols)
```

[576]:
```
70
```

[ ]:
```

```

[577]:
```
unseen_test_new = unseen_test[significant_cols].copy()
```

[578]:
```
unseen_test_new.head()
```

[578]:

ID	X0	X5	X45	X47	X48	X52	X54	X74	X75	X76	…	X336	X337	X338	\
1	21	26	0	0	0	0	1	1	0	1	…	0	0	0	
2	42	9	1	0	0	0	0	1	0	0	…	0	0	0	
3	21	0	0	0	0	0	1	1	0	1	…	0	0	0	
4	21	31	0	0	0	0	1	1	0	1	…	0	0	0	
5	45	30	0	0	0	0	0	1	0	0	…	1	1	0	

ID	X339	X364	X365	X368	X375	X376	X383
1	0	0	0	0	0	0	0

```
2      0     0     0     1     0     0     0
3      0     0     0     0     0     0     0
4      0     0     0     0     0     0     0
5      0     0     0     0     1     0     0

[5 rows x 70 columns]
```

[579]:
```
unseen_test_new.shape
```

[579]:
```
(4209, 70)
```

[ ]:
```

```

6) Scalling: Note: We always have to Use the Same Scalar Object that we used on Training data to transform New Data.

We should not Create new Scalar object for New Data.

As we have Over-Written same Scalar Object multiple times while Model Building, we will create a Scalar Object and Fit it on Same Training Data as before.

Scaled Data Used in XGBR_Model_2 was scaled by using Data in new_x Data Frame.

So, we will Use Same dataframe to Fit the Scalar Object and then Transform our New Data Using That Scalar.

[580]:
```
new_x.head()
```

[580]:
```
    X0  X5  X45  X47  X48  X52  X54  X74  X75  X76  …  X336  X337  X338  \
ID                                                  …
0   32  24    0    0    0    0    0    1    0    0  …     0     0     0
6   32  28    0    0    0    0    0    1    0    0  …     1     1     0
7   20  27    0    0    0    0    1    1    1    1  …     0     0     0
9   20  27    0    0    0    0    1    1    0    1  …     0     0     0
13  20  12    0    0    0    0    1    1    0    1  …     0     0     0

    X339  X364  X365  X368  X375  X376  X383
ID
0      0     0     0     0     0     0     0
6      0     0     0     0     1     0     0
7      0     0     0     0     0     0     0
9      0     0     0     0     0     0     0
13     0     0     0     0     0     0     0

[5 rows x 70 columns]
```

[ ]:
```

```

[581]:
```
scalar =StandardScaler()
```

[582]:
```
scalar.fit(new_x)
```

[582]:
```
StandardScaler()
```

[583]:
```
temp = scalar.fit_transform(unseen_test_new)
unseen_scaled = pd.DataFrame(temp, index=unseen_test_new.index, columns=␣
 ↪unseen_test_new.columns)
unseen_scaled.head()
```

[583]:
```
          X0        X5       X45       X47       X48       X52       X54  \
ID
1  -0.625211  1.266652 -0.580551 -0.106267 -0.156003 -0.210753  4.904444
2   0.754609 -0.695011  1.722502 -0.106267 -0.156003 -0.210753 -0.203897
3  -0.625211 -1.733538 -0.580551 -0.106267 -0.156003 -0.210753  4.904444
4  -0.625211  1.843611 -0.580551 -0.106267 -0.156003 -0.210753  4.904444
5   0.951726  1.728219 -0.580551 -0.106267 -0.156003 -0.210753 -0.203897

         X74       X75       X76  …      X336      X337      X338      X339  \
ID                                  …
1   0.046291 -0.200715  4.904444  … -0.382416 -1.055468 -0.088895 -0.015416
2   0.046291 -0.200715 -0.203897  … -0.382416 -1.055468 -0.088895 -0.015416
3   0.046291 -0.200715  4.904444  … -0.382416 -1.055468 -0.088895 -0.015416
4   0.046291 -0.200715  4.904444  … -0.382416 -1.055468 -0.088895 -0.015416
5   0.046291 -0.200715 -0.203897  …  2.614955  0.947447 -0.088895 -0.015416

        X364      X365      X368      X375      X376      X383
ID
1  -0.059804 -0.057769 -0.259733 -0.695420 -0.228583 -0.021804
2  -0.059804 -0.057769  3.850105 -0.695420 -0.228583 -0.021804
3  -0.059804 -0.057769 -0.259733 -0.695420 -0.228583 -0.021804
4  -0.059804 -0.057769 -0.259733 -0.695420 -0.228583 -0.021804
5  -0.059804 -0.057769 -0.259733  1.437979 -0.228583 -0.021804

[5 rows x 70 columns]
```

[ ]:
```

```

7) Using XGBR_Model_2 to Make Predictions on New Data:

[584]:
```
XGBR_Model_2.get_params
```

[584]:
```
<bound method XGBModel.get_params of XGBRegressor(base_score=0.5,
booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.4, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.02, max_delta_step=0, max_depth=3,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=500, n_jobs=4, num_parallel_tree=1, random_state=0,
```

```
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                tree_method='exact', validate_parameters=1, verbosity=None)>
```

[585]:
```
Predictions = XGBR_Model_2.predict(unseen_scaled)
```

[586]:
```
Predictions
```

[586]:
```
array([ 99.57243 , 113.51955 ,  96.249695, …,  92.27821 , 110.89687 ,
        91.844406], dtype=float32)
```

[587]:
```
Predictions.shape
```

[587]:
```
(4209,)
```

[ ]:

8) Adding Predictions With Original Data in Data Frame:

[588]:
```
# Loading Unseen Test Data again from file as we have made some manipulations␣
↪to columns.
```

[589]:
```
new_data = pd.read_csv("test.csv")
```

[590]:
```
new_data.head()
```

[590]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	…	X375	X376	X377	X378	X379	X380	\
0	1	az	v	n	f	d	t	a	w	0	…	0	0	0	1	0	0	
1	2	t	b	ai	a	d	b	g	y	0	…	0	0	1	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	…	0	0	0	1	0	0	
3	4	az	l	n	f	d	z	l	n	0	…	0	0	0	1	0	0	
4	5	w	s	as	c	d	y	i	m	0	…	1	0	0	0	0	0	

	X382	X383	X384	X385
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

[5 rows x 377 columns]

[ ]:

[591]:
```
new_data["y"] = Predictions
```

[ ]:

[592]:
```
new_data.head()
```

[592]:
```
   ID  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X376  X377  X378  X379  X380  X382  \
0   1  az  v   n  f  d  t  a  w    0  …     0     0     1     0     0     0
1   2   t  b  ai  a  d  b  g  y    0  …     0     1     0     0     0     0
2   3  az  v  as  f  d  a  j  j    0  …     0     0     1     0     0     0
3   4  az  l   n  f  d  z  l  n    0  …     0     0     1     0     0     0
4   5   w  s  as  c  d  y  i  m    0  …     0     0     0     0     0     0

   X383  X384  X385           y
0     0     0     0   99.572433
1     0     0     0  113.519547
2     0     0     0   96.249695
3     0     0     0   77.344536
4     0     0     0  110.068550

[5 rows x 378 columns]
```

[ ]:

9) Saving New Data with Predictions to "csv" file:

[593]:
```
new_data.to_csv("test_with_predictions.csv", index= None)
```