pip install xgboost : Install XGboost iif not installed.

Data preprocessing

Load datasets ¶

Loading train.csv to train our model and observe the shape

In [212]:
```
import pandas as pd
import numpy as np
```

In [185]:
```
mercedesBenz_df = pd.read_csv("E:\\Simplilean\\ML\\ProjectSubmision\\MercedesBenz\\Datasets\\train.csv", sep=",")
```

In [153]:
```
mercedesBenz_df.shape
```

Out[153]: (4209, 378)

Loanding test.csv to predict our new data using XGboostRegressor model and

Checking shape

Visualizing in table using head()

This test data will be used to predict values after our model XGboost Regressor got trained

In [193]:
```
new_df = pd.read_csv("E:\\Simplilean\\ML\\ProjectSubmision\\MercedesBenz\\Datasets\\test.csv", sep=",")
```

In [194]:
```
new_df.shape
```

Out[194]: (4209, 377)

In [156]:
```
new_df.head()
```

Out[156]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0

5 rows × 377 columns

Spliting train data into X, y to train our model.

printing values using head() to make confimratiom

In [186]:
```
y = mercedesBenz_df.iloc[:,1:2]
```

In [187]:
```
mercedesBenz_df.pop('y')
X= mercedesBenz_df
```

In [188]:
```
X.head(10)
```

Out[188]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	0	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	0
1	6	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	0
2	7	az	w	n	c	d	x	j	x	0	...	0	0	0	0	0	0	1
3	9	az	t	n	f	d	x	l	e	0	...	0	0	0	0	0	0	0
4	13	az	v	n	f	d	h	d	n	0	...	0	0	0	0	0	0	0
5	18	t	b	e	c	d	g	h	s	0	...	0	0	1	0	0	0	0
6	24	al	r	e	f	d	f	h	s	0	...	0	0	0	0	0	0	0
7	25	o	l	as	f	d	f	j	a	0	...	0	0	0	0	0	0	0
8	27	w	s	as	e	d	f	i	h	0	...	1	0	0	0	0	0	0
9	30	j	b	aq	c	d	f	a	e	0	...	0	0	1	0	0	0	0

10 rows × 377 columns

In [189]:
```
y.head(10)
```

Out[189]:

	y
0	130.81
1	88.53
2	76.26
3	80.62
4	78.02
5	92.93
6	128.76
7	91.91
8	108.67
9	126.99

Written Custom class to remove 'ID' column.

Because , to add this object to Pipeline for future predicting values

In [190]:
```
class RemoveIDClass():

    def __init__(self):
        print('RemoveIDClass initiated. Use fit transfrom to remove columns')

    def fit(self, X):
         print(' Use transfrom to drop columns')
         return self

    def transform(self, X):
        self.X=X
        self.X= self.X.drop('ID', axis=1)
        return self.X
```

Applying RemoveIDClass to remove 'ID'

and apply same thing for final predicting test data i.e. new_df here

In [191]:
```
column_selection= RemoveIDClass()
column_selection.fit(X)
X=column_selection.transform(X)
```

```
RemoveIDClass initiated. Use fit transfrom to remove columns
 Use transfrom to drop columns
```

In [195]:
```
new_df= column_selection.transform(new_df)
```

In [196]:
```
new_df.head()
```

Out[196]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0	0
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0	0
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0	0
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0	0

5 rows × 376 columns

Spliting X, y int0 train and test with 70, 30 %

In [169]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3, random_stat
e=41)
```

Task 1: Apply label encoder.

In [80]:
```
from feature_engine import categorical_encoders as ce

# set up the encoder
encoder = ce.OneHotCategoricalEncoder(drop_last=False)

# fit the encoder
encoder.fit(X_train)

# transform the data
encoded_X_train= encoder.transform(X_train)
encoded_X_test= encoder.transform(X_test)

encoder.encoder_dict_
```

Out[80]:
```
{'X0': array(['y', 'n', 'ay', 'j', 'al', 'az', 'z', 'f', 's', 'aj', 'e', 'a
x',
        'ak', 'x', 'w', 'ap', 'o', 't', 'r', 'a', 'af', 'u', 'd', 'ai',
        'i', 'v', 'ba', 'am', 'at', 'h', 'aq', 'l', 'as', 'm', 'c', 'b',
        'bc', 'k', 'aw', 'q', 'au', 'ao', 'ad', 'g', 'ac', 'ab', 'aa'],
       dtype=object),
 'X1': array(['aa', 'e', 'b', 'r', 's', 'c', 'o', 'l', 'v', 'y', 'z', 'u',
'i',
        'a', 'm', 'd', 'f', 'h', 'w', 't', 'n', 'p', 'j', 'k', 'g', 'ab',
        'q'], dtype=object),
 'X2': array(['q', 'as', 'f', 'ay', 'e', 'm', 'ai', 'aq', 'ae', 'r', 'a', 'a
h',
        'i', 'al', 'k', 'ak', 'n', 's', 'aw', 't', 'b', 'd', 'y', 'ap',
        'x', 'g', 'h', 'ao', 'au', 'z', 'ac', 'at', 'ag', 'an', 'av', 'am',
        'af', 'aa', 'o', 'c', 'j', 'p', 'ar', 'l'], dtype=object),
 'X3': array(['c', 'f', 'd', 'a', 'b', 'g', 'e'], dtype=object),
 'X4': array(['d'], dtype=object),
 'X5': array(['n', 'q', 'w', 's', 'j', 'af', 'p', 'l', 'k', 'c', 'd', 'i',
'r',
        'ab', 'ad', 'ag', 'm', 'ae', 'ac', 'aa', 'ah', 'v', 'o', 'f', 'g',
        'x', 'u'], dtype=object),
 'X6': array(['j', 'a', 'h', 'd', 'g', 'i', 'l', 'k', 'c', 'f', 'e', 'b'],
       dtype=object),
 'X8': array(['h', 'm', 'v', 'n', 'g', 'o', 'p', 'e', 'r', 'u', 'q', 'j',
'i',
        'w', 'c', 'a', 'f', 't', 'y', 'k', 'b', 'x', 's', 'l', 'd'],
       dtype=object)}
```

Applying same label encoder to new_Df for final predicting values

and visualizing data using head()

In [170]:
```
encoded_new_df= encoder.transform(new_df)
```

In [171]:
```
encoded_new_df.head(10)
```

Out[171]:

	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	...	X8_a	X8_f	X8_t	X8_y	X8_k	X8_
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	1	...	0	0	0	1	0	
2	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	
7	0	0	0	1	0	0	0	0	0	0	...	1	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
9	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	

10 rows × 558 columns

In [81]:
```
X_train.head()
```

Out[81]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X
2468	y	aa	q	c	d	n	j	h	0	0	...	1	0	0	0	0	0	
3023	n	e	as	c	d	q	j	m	0	0	...	0	0	1	0	0	0	
3925	y	b	f	c	d	w	a	v	0	0	...	0	0	1	0	0	0	
3999	ay	aa	as	c	d	w	h	n	0	0	...	1	0	0	0	0	0	
3196	j	aa	ay	c	d	s	j	g	1	0	...	1	0	0	0	0	0	

5 rows × 376 columns

In [82]:
```
encoded_X_train.head()
```

Out[82]:

	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	...	X8_a	X8_f	X8_t	X8_y	X8_k
2468	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3023	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0
3925	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3999	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0
3196	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 558 columns

Task 2: Missing values handling : Check for null and unique values for test and train sets

In [83]:
```
from sklearn.impute import SimpleImputer
imputer= SimpleImputer()
imputer.fit(encoded_X_train)
misingvalue_encoded_X_train = imputer.transform(encoded_X_train)
misingvalue__encoded_X_test= imputer.transform(encoded_X_test)
```

Applying same missing values to new_Df for final predicting values

In [172]:
```
misingvalue__encoded_new_df = imputer.transform(encoded_new_df)
```

In [84]:
```
misingvalue_encoded_X_train.shape
```

Out[84]:
```
(2946, 558)
```

In [85]:
```
misingvalue__encoded_X_test.shape
```

Out[85]:
```
(1263, 558)
```

In [86]:
```
type(misingvalue__encoded_X_test)
```

Out[86]:
```
numpy.ndarray
```

Task 3: If for any column(s), the variance is equal to zero, then you need to remove those variable(s)

Applying Varaince on data sets and visualizing using head()

In [87]:
```
from sklearn.feature_selection import VarianceThreshold
variance = VarianceThreshold()

variance.fit(misingvalue_encoded_X_train)
variance.transform(misingvalue_encoded_X_train)
variance.transform(misingvalue__encoded_X_test)
variance.variances_
```

Out[87]:
```
array([0.01173937, 0.        , 0.06881193, 0.05855662, 0.24506286,
       0.00067843, 0.00169434, 0.00741199, 0.00774624, 0.08874841,
       0.12657157, 0.00270817, 0.07709534, 0.0199518 , 0.00169434,
       0.0054016 , 0.21702065, 0.03374027, 0.04372493, 0.00405673,
       0.17717697, 0.01140788, 0.00033933, 0.00640783, 0.17717697,
       0.00304566, 0.17717697, 0.03215885, 0.00033933, 0.00067843,
       0.01140788, 0.00033933, 0.06736417, 0.0100796 , 0.18724516,
       0.24090197, 0.01173937, 0.02125242, 0.10905659, 0.16966451,
       0.19687804, 0.04125089, 0.00741199, 0.04403315, 0.00640783,
       0.02351964, 0.01306304, 0.24297044, 0.00101729, 0.00135593,
       0.04526372, 0.0054016 , 0.01140788, 0.23506726, 0.00237046,
       0.02545399, 0.00169434, 0.07111637, 0.02738003, 0.07254915,
       0.09442251, 0.021577  , 0.00101729, 0.03405587, 0.04403315,
       0.01173937, 0.00640783, 0.02641805, 0.0492376 , 0.17953913,
       0.01635607, 0.00135593, 0.09442251, 0.24427014, 0.00135593,
       0.        , 0.00640783, 0.        , 0.00674278, 0.00101729,
       0.00101729, 0.        , 0.00674278, 0.00033933, 0.18397228,
       0.00338291, 0.05196594, 0.00774624, 0.21321833, 0.05855662,
       0.00741199, 0.16596692, 0.00169434, 0.00203251, 0.01438302,
       0.        , 0.01537058, 0.03719912, 0.00067843, 0.02384261,
       0.00270817, 0.02125242, 0.13012328, 0.20397766, 0.15749813,
       0.04341648, 0.23481734, 0.23481734, 0.04125089, 0.00640783,
       0.00270817, 0.        , 0.00304566, 0.03719912, 0.24999067,
       0.039074  , 0.10931135, 0.039074  , 0.02738003, 0.21386646,
       0.10777936, 0.02125242, 0.02481013, 0.04403315, 0.24166244,
       0.04125089, 0.08018663, 0.04125089, 0.01569931, 0.18043728,
       0.03374027, 0.1524763 , 0.00101729, 0.04125089, 0.02125242,
       0.04495642, 0.16438516, 0.08130382, 0.03215885, 0.00067843,
       0.16557287, 0.07426089, 0.20397766, 0.20397766, 0.18043728,
       0.01372349, 0.00135593, 0.15729153, 0.0409406 , 0.21243298,
       0.05825946, 0.00439329, 0.03374027, 0.00067843, 0.19640759,
       0.0070775 , 0.02384261, 0.22530003, 0.0054016 , 0.01041201,
       0.01733949, 0.0202773 , 0.01733949, 0.04495642, 0.24674798,
       0.04495642, 0.131763  , 0.08573571, 0.09602501, 0.00405673,
       0.00135593, 0.02060257, 0.24971195, 0.24601379, 0.07906575,
       0.00033933, 0.24929899, 0.00270817, 0.24971195, 0.01306304,
       0.01140788, 0.02897873, 0.02222547, 0.00270817, 0.00741199,
       0.14024251, 0.18132967, 0.01635607, 0.00033933, 0.00033933,
       0.01897391, 0.00033933, 0.06033472, 0.08983703, 0.00033933,
       0.01537058, 0.00405673, 0.00169434, 0.00741199, 0.08847568,
       0.0054016 , 0.00674278, 0.21577626, 0.06649275, 0.24678658,
       0.00808027, 0.02125242, 0.24686309, 0.21640133, 0.08352714,
       0.03215885, 0.00304566, 0.04063007, 0.0409406 , 0.00607265,
       0.01668411, 0.04372493, 0.        , 0.15955138, 0.        ,
       0.00033933, 0.00607265, 0.07878495, 0.00741199, 0.00338291,
       0.08929318, 0.00674278, 0.00640783, 0.09442251, 0.        ,
       0.2447725 , 0.18132967, 0.00135593, 0.0070775 , 0.24719596,
       0.23899851, 0.00067843, 0.00135593, 0.00607265, 0.01864749,
       0.06938943, 0.00033933, 0.00169434, 0.00033933, 0.00033933,
       0.24274091, 0.00135593, 0.04372493, 0.037825  , 0.08683446,
       0.00135593, 0.00741199, 0.        , 0.00033933, 0.00033933,
       0.00237046, 0.037825  , 0.20236168, 0.01041201, 0.19858528,
       0.03969712, 0.00135593, 0.00033933, 0.04372493, 0.        ,
       0.00304566, 0.00439329, 0.12345148, 0.037825  , 0.16908621,
       0.05557468, 0.01504163, 0.00033933, 0.        , 0.        ,
       0.01140788, 0.00674278, 0.        , 0.10726685, 0.00033933,
       0.00033933, 0.        , 0.00472962, 0.00472962, 0.16298902,
```

```
0.04618422, 0.0100796 , 0.06967783, 0.01306304, 0.04156095,
0.00270817, 0.00941408, 0.00640783, 0.00338291, 0.23997283,
0.00371993, 0.21150625, 0.24467387, 0.0280202 , 0.15667037,
0.00674278, 0.00033933, 0.00067843, 0.00640783, 0.18150746,
0.02448785, 0.0100796 , 0.24297044, 0.0054016 , 0.03215885,
0.10956587, 0.04063007, 0.24601379, 0.        , 0.05075565,
0.00067843, 0.02287302, 0.24882462, 0.00472962, 0.11435806,
0.24985885, 0.00674278, 0.        , 0.02092761, 0.00908097,
0.02125242, 0.07283502, 0.00908097, 0.02190135, 0.04279889,
0.        , 0.0489333 , 0.04031932, 0.22572496, 0.20934561,
0.0489333 , 0.00203251, 0.16869952, 0.23174841, 0.14601236,
0.00033933, 0.24697612, 0.03152467, 0.07426089, 0.03279211,
0.24999712, 0.18622061, 0.00338291, 0.00338291, 0.00135593,
0.04832401, 0.06063027, 0.        , 0.00741199, 0.01471244,
0.00067843, 0.01897391, 0.17570397, 0.21577626, 0.05527522,
0.21552461, 0.02222547, 0.00974695, 0.00774624, 0.00741199,
0.00203251, 0.00033933, 0.00135593, 0.06736417, 0.0489333 ,
0.06269262, 0.03844996, 0.01635607, 0.04248975, 0.07709534,
0.05196594, 0.02448785, 0.03719912, 0.0070775 , 0.00439329,
0.07737752, 0.06620182, 0.04341648, 0.02351964, 0.06298632,
0.06649275, 0.00270817, 0.00506572, 0.00774624, 0.00371993,
0.01733949, 0.00841407, 0.00405673, 0.00941408, 0.00640783,
0.00304566, 0.00573724, 0.01537058, 0.00371993, 0.00405673,
0.00203251, 0.00841407, 0.00101729, 0.00203251, 0.00135593,
0.00270817, 0.00405673, 0.00033933, 0.00270817, 0.00135593,
0.00304566, 0.00033933, 0.00033933, 0.00033933, 0.00033933,
0.15353346, 0.00841407, 0.12029244, 0.05226793, 0.12633294,
0.02641805, 0.01766684, 0.12345148, 0.09281171, 0.00506572,
0.00841407, 0.00974695, 0.0431078 , 0.03279211, 0.00908097,
0.00067843, 0.00607265, 0.00741199, 0.01339338, 0.0054016 ,
0.00506572, 0.00237046, 0.00640783, 0.00371993, 0.00203251,
0.00067843, 0.00033933, 0.00135593, 0.23899851, 0.0199518 ,
0.01140788, 0.01864749, 0.07709534, 0.08847568, 0.01537058,
0.10777936, 0.03279211, 0.01140788, 0.00101729, 0.00674278,
0.00135593, 0.0054016 , 0.06033472, 0.03374027, 0.02125242,
0.00169434, 0.00741199, 0.0054016 , 0.00338291, 0.00203251,
0.00169434, 0.00203251, 0.00270817, 0.00135593, 0.00405673,
0.00101729, 0.00304566, 0.00304566, 0.00135593, 0.00405673,
0.00135593, 0.00101729, 0.00033933, 0.00033933, 0.00033933,
0.00033933, 0.00033933, 0.00033933, 0.00101729, 0.00033933,
0.00033933, 0.24847619, 0.19126041, 0.06678346, 0.09469017,
0.01372349, 0.04984551, 0.03563037, 0.        , 0.0489333 ,
0.0504525 , 0.05105857, 0.04587762, 0.02993519, 0.04495642,
0.04771379, 0.04434114, 0.04218038, 0.0280202 , 0.04710266,
0.04771379, 0.04771379, 0.04187078, 0.04279889, 0.04618422,
0.04954167, 0.04832401, 0.04125089, 0.02673894, 0.02319645,
0.0504525 , 0.00439329, 0.00135593, 0.00033933, 0.00067843,
0.00033933, 0.18501481, 0.04862877, 0.04464889, 0.12537614,
0.18876643, 0.10235207, 0.09735409, 0.00974695, 0.00741199,
0.00472962, 0.00338291, 0.00674278, 0.02770023, 0.03625856,
0.04801902, 0.05287123, 0.03025355, 0.03719912, 0.02287302,
0.04862877, 0.05166371, 0.02673894, 0.02897873, 0.06181014,
0.0525697 , 0.04434114, 0.02287302, 0.04832401, 0.0525697 ,
0.02833994, 0.02641805, 0.037825  , 0.04403315, 0.02416535,
0.05587391, 0.02092761, 0.02416535])
```

Apply same varaince object on new_df

In [174]:
```
variance.transform(misingvalue__encoded_new_df)
```

Out[174]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [88]:
```
pd.DataFrame(data = misingvalue_encoded_X_train).head()
```

Out[88]:

	0	1	2	3	4	5	6	7	8	9	...	548	549	550	551	552	553	554	555
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 558 columns

In [89]:
```
pd.DataFrame(data = misingvalue__encoded_X_test).head()
```

Out[89]:

	0	1	2	3	4	5	6	7	8	9	...	548	549	550	551	552	553	554	555
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 558 columns

Task 4: Perform dimensionality reduction.

Applying PCA to dimensionality reduction and print values

In [201]:
```
from sklearn.decomposition import PCA
pca= PCA(n_components=0.95)

final_X_train = pca.fit_transform(misingvalue_encoded_X_train)

final_X_test = pca.transform(misingvalue__encoded_X_test)


variance_factor = pca.explained_variance_ratio_
```

Applying PCA on new_df

In [202]:
```
 final_new_df= pca.transform(misingvalue__encoded_new_df)
```

In [203]:
```
varaince=np.cumsum(np.round(variance_factor, decimals=4)*100)
varaince
```

Out[203]:
```
array([11.5 , 19.26, 26.52, 32.33, 37.22, 41.52, 44.88, 47.75, 50.23,
       52.34, 54.4 , 56.11, 57.62, 59.03, 60.42, 61.72, 62.92, 64.02,
       65.01, 65.94, 66.83, 67.68, 68.49, 69.27, 70.  , 70.71, 71.39,
       72.04, 72.67, 73.29, 73.88, 74.44, 74.96, 75.46, 75.94, 76.39,
       76.83, 77.24, 77.65, 78.06, 78.44, 78.82, 79.19, 79.55, 79.9 ,
       80.23, 80.56, 80.87, 81.17, 81.46, 81.74, 82.02, 82.29, 82.56,
       82.82, 83.07, 83.31, 83.55, 83.79, 84.02, 84.24, 84.46, 84.68,
       84.89, 85.1 , 85.31, 85.51, 85.71, 85.91, 86.11, 86.3 , 86.49,
       86.68, 86.87, 87.06, 87.24, 87.42, 87.6 , 87.78, 87.96, 88.14,
       88.31, 88.48, 88.65, 88.82, 88.99, 89.15, 89.31, 89.47, 89.63,
       89.79, 89.95, 90.1 , 90.25, 90.4 , 90.55, 90.7 , 90.85, 90.99,
       91.13, 91.27, 91.41, 91.55, 91.68, 91.81, 91.94, 92.07, 92.2 ,
       92.33, 92.45, 92.57, 92.69, 92.81, 92.93, 93.05, 93.16, 93.27,
       93.38, 93.49, 93.6 , 93.71, 93.82, 93.92, 94.02, 94.12, 94.22,
       94.32, 94.42, 94.52, 94.61, 94.7 , 94.79, 94.88, 94.97, 95.06])
```

In [204]:
```
final_X_train = pd.DataFrame(data = final_X_train)
final_X_train.head()
```

Out[204]:

	0	1	2	3	4	5	6	7	
0	0.960959	-0.408379	-1.877273	-0.254270	0.513301	0.512033	1.046224	0.654329	0.9152
1	-0.752272	1.052960	0.836741	0.077246	-1.047652	-0.059477	-1.834881	-0.033379	0.7217
2	2.968586	0.597727	-0.194642	0.238958	-0.793592	0.188629	-1.355411	0.286202	0.8568
3	-0.796384	1.620700	-1.563388	-0.505662	1.292720	1.189746	-0.407920	1.497050	1.0206
4	0.486709	0.137798	-1.145470	0.187931	0.645374	0.800998	2.484812	0.663025	-0.0237

5 rows × 135 columns

In [205]:
```
final_X_test = pd.DataFrame(data = final_X_test)
final_X_test.head()
```

Out[205]:

	0	1	2	3	4	5	6	7	
0	3.356964	0.780834	0.508713	-0.740357	-1.494637	-0.455610	-0.431170	-1.028857	-0.7725
1	1.672618	1.268216	-0.160392	1.044819	-1.166278	-0.318978	0.829785	-0.009819	-0.1397
2	-2.359431	1.657815	1.913290	-0.249068	-0.135485	0.652335	0.134101	-0.897848	0.2408
3	1.851941	-0.210400	-0.424963	-1.833808	2.352375	0.898610	0.244001	-0.549630	0.0109
4	-3.020196	-0.073882	-0.183182	-0.139958	-0.218389	-1.509716	-0.756099	0.212126	-1.1548

5 rows × 135 columns

In [206]:
```
final_X_train.shape
```

Out[206]: (2946, 135)

In [207]:
```
y_train.shape
```

Out[207]: (2946, 1)

Task 5: Predict your test_df values using XGBoost.

i. Create object XGBRegressor

ii. fit final_X_train and y_train

iii. Predict for final_X_test

iv. Check MSE for prediction

v. Hyperparameter tunning to find optimised model

vi. Predict for the given test.csv i.e. new_df here on optimised model

In [101]:
```
from xgboost import XGBRegressor

xgb = XGBRegressor()

xgb.fit(final_X_train,y_train)
y_preds = xgb.predict(final_X_test)
```

In [102]:
```
y_preds
```

Out[102]:
```
array([ 90.128044, 102.15285 , 119.29104 , ..., 101.01388 , 100.81295 ,
        92.27072 ], dtype=float32)
```

In [103]:
```
print(xgb)
```

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

In [104]:
```
y_test
```

Out[104]:

	y
2107	89.91
3683	104.87
137	110.02
1021	108.64
1299	104.37
...	...
2242	115.18
1131	118.42
2676	104.28
2206	88.25
2481	128.87

1263 rows × 1 columns

iv. Calculating mean_squared_error

In [105]:
```
from sklearn.metrics import mean_squared_error

rms = np.sqrt(mean_squared_error(y_test, y_preds))
```

In [106]:
```
rms
```

Out[106]:
```
9.265854990325133
```

v. Using GridSearchCV to hyper parameter tunning with differenct hyper paramters

In [110]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV


# Define your param grid
parameters = {'max_depth': (5,7,10), 'booster' :('gbtree','gblinear'),'learning_rate':(0.03,0.05,0.07), 'n_estimators':(100,200,250)}

#Define your Search object here:
Search_object = GridSearchCV(xgb, parameters)

#Fit your search object with your traing data
Search_object.fit(final_X_train, y_train)

Search_object.best_params_
```

Out[110]:
```
{'booster': 'gbtree',
 'learning_rate': 0.03,
 'max_depth': 5,
 'n_estimators': 250}
```

Found that {'booster': 'gbtree', 'learning_rate': 0.03, 'max_depth': 5, 'n_estimators': 250} are best params

So, Find the score

In [111]:
```
print("best XGboost regression from grid search: %f" % Search_object.best_estimator_.score(final_X_test, y_test))
```

```
best XGboost regression from grid search: 0.503886
```

Predicting final X test values using Optimized XGBRegressor model

In [112]:
```
xgb_optimized = XGBRegressor(booster='gbtree',learning_rate= 0.03, max_depth=
5, n_estimators= 250)
print(xgb_optimized)
xgb_optimized.fit(final_X_train,y_train)


y_preds_optimized = xgb_optimized.predict(final_X_test)
```

```
XGBRegressor(base_score=None, booster='gbtree', colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=Non
e,
             learning_rate=0.03, max_delta_step=None, max_depth=5,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=250, n_jobs=None, num_parallel_tree=None,
             objective='reg:squarederror', random_state=None, reg_alpha=None,
             reg_lambda=None, scale_pos_weight=None, subsample=None,
             tree_method=None, validate_parameters=False, verbosity=None)
```

Printing predicitons

In [208]:
```
y_preds_optimized = pd.DataFrame(data = y_preds_optimized)
print(y_preds_optimized)
```

```
               0
0      92.019440
1     110.998375
2     107.257530
3     110.801277
4     110.453041
...          ...
1258  106.840363
1259  110.301445
1260  101.018021
1261   93.686005
1262   93.001305

[1263 rows x 1 columns]
```

vi. Predicting values for final_new_df using oprimized trained XGBRegressor.

In [178]:
```
final_new_df = pd.DataFrame(data = final_new_df)
final_new_df.head()
```

Out[178]:

	0	1	2	3	4	5	6	7	
0	-0.410161	-3.089787	0.611238	3.306720	-0.608157	3.628391	-0.555998	0.185647	-1.50310
1	3.918156	0.636312	1.344518	-0.487245	0.875199	-0.306045	-2.128026	-1.104256	-0.99765
2	-1.293756	-0.794236	0.545889	1.919805	0.248940	2.964675	-0.889490	0.842970	-1.26485
3	-0.449481	-3.074606	0.463103	3.315172	-0.715547	3.547886	-0.692836	0.137772	-1.67249
4	-2.860020	1.425717	0.808587	-1.509947	0.334650	0.356097	-0.198072	0.152543	-0.00536

5 rows × 135 columns

In [179]:
```
y_preds_new_df = xgb_optimized.predict(final_new_df)
```

printing predicitions for final new_df

In [209]:
```
y_preds_new_df = pd.DataFrame(data=y_preds_new_df)
print(y_preds_new_df)
```

```
               0
0      78.897202
1      95.644554
2      79.189850
3      79.186584
4     111.400620
...          ...
4204  105.252731
4205   94.326477
4206   96.404037
4207  110.042572
4208   93.147545

[4209 rows x 1 columns]
```

Using pipeline for future predicitons datatsets

and adding all objects to pipeline

In [210]:
```
from sklearn.pipeline import Pipeline, make_pipeline

mercedesBenz_XGboostRegressor_pipeline = make_pipeline(column_selection,
                     encoder,
                     imputer,
                     variance,
                     pca,
                     xgb_optimized)
```

Storing Pipeline object using Pickle

In [116]:
```
import pickle

# Save your object
pickle.dump(mercedesBenz_XGboostRegressor_pipeline,open("mercedesBenz_XGboostR
egressor.pkl", 'wb'))
```

this is how, we load pipeline for future predictions

In [118]:
```
# Load you object
trained_xgboostRegressor = pickle.load(open("mercedesBenz_XGboostRegressor.pk
l", 'rb'))
```

In [119]:
```
new_df = pd.read_csv("E:\\Simplilean\\ML\\ProjectSubmision\\MercedesBenz\\Data
sets\\new_test.csv", sep=",")
```

In [211]:
```
type(trained_xgboostRegressor)
```

Out[211]:
```
sklearn.pipeline.Pipeline
```

In [135]:
```
trained_xgboostRegressor.named_steps
```

Out[135]:
```
{'removeidclass': <__main__.RemoveIDClass at 0x1cacf898>,
 'onehotcategoricalencoder': OneHotCategoricalEncoder(drop_last=False, top_ca
tegories=None,
                          variables=['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X
6',
                                     'X8']),
 'simpleimputer': SimpleImputer(add_indicator=False, copy=True, fill_value=No
ne,
               missing_values=nan, strategy='mean', verbose=0),
 'variancethreshold': VarianceThreshold(threshold=0.0),
 'pca': PCA(copy=True, iterated_power='auto', n_components=0.95, random_state
=None,
     svd_solver='auto', tol=0.0, whiten=False),
 'xgbregressor': XGBRegressor(base_score=0.5, booster='gbtree', colsample_byl
evel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.03, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=250, n_jobs=0, num_parallel_tree=1,
              objective='reg:squarederror', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=Non
e,
              validate_parameters=False, verbosity=None)}
```

Finally we apply loaded object to on new future predictions datatsets

In [ ]:
```
trained_xgboostRegressor.predict(new_df)
```

End of Project

Thank you.