Following actions should be performed:

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

Check for null and unique values for test and train sets

Apply label encoder.

Perform dimensionality reduction.

Predict your test_df values using xgboost

Our process flow will be

1. check for null value
2. Check for unique value
3. Convert the columns/feature with unique values <=2 to category
4. Apply label endoder
5. If for any column the variance is equal to zero then we need to remove those variables
6. Perform scalling
7. Perform PCA 7.1 fit_transform for train data 7.2 transform for test data
8. Modelling

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.feature_selection import VarianceThreshold
```

In [2]:
```
dfRawTest=pd.read_csv(r"E:\data_sciene_course\simplilearn\Machine Learing\Project 1
dfRawTest.head()
```

Out[2]:

	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

In [3]:
```
dfRawTest.drop("ID", axis=1, inplace=True)
dfRawTest.shape
```

Out[3]: (4209, 376)

In [4]:
```
x_test=dfRawTest
```

In [5]:
```
dfRawTrain=pd.read_csv(r"E:\data_sciene_course\simplilearn\Machine Learing\Project 1
```

```
dfRawTrain.head()
```

Out[5]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0

5 rows × 378 columns

In [6]:
```
dfRawTrain.isnull().any().sum()
```

Out[6]:
```
0
```

In [7]:
```
dfRawTest.isnull().any().sum()
```

Out[7]:
```
0
```

Both the Train and test data has no null value

In [8]:
```
list(set(dfRawTrain.columns)-set(dfRawTest.columns))
```

Out[8]:
```
['y', 'ID']
```

In [9]:
```
#Dropping Target column from Train dataset
x_train1=dfRawTrain.drop(["y", "ID"], axis=1)
y_target1=dfRawTrain.y
```

In [10]:
```
x_train1.dtypes
```

Out[10]:
```
X0      object
X1      object
X2      object
X3      object
X4      object
         ...
X380     int64
X382     int64
X383     int64
X384     int64
X385     int64
Length: 376, dtype: object
```

Check for unique values for test and train sets

In [11]:
```
#Calculating the columns with Unique value less than
x_Train_Unique=x_train1.columns[x_train1.nunique(dropna=False)<=2]
x_Train_Unique
```

Out[11]:
```
Index(['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=368)
```

In [12]:
```
x_Train_Unique.value_counts().count()
```

Out[12]:
```
368
```

In [13]:
```
# Creating a function to check the unique value <=2 and coverting these columns to c
def unique(Data):
    DataToBeCategorical=Data.columns[Data.nunique(dropna=False)<=2];
    Data[DataToBeCategorical]=Data[DataToBeCategorical].astype('category')
    return DataToBeCategorical, Data.dtypes
```

In [14]:
```
unique(x_train1)
```

Out[14]:
```
(Index(['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
        ...
        'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
        'X385'],
       dtype='object', length=368),
 X0        object
 X1        object
 X2        object
 X3        object
 X4        object
           ...
 X380    category
 X382    category
 X383    category
 X384    category
 X385    category
 Length: 376, dtype: object)
```

In [15]:
```
x_train1.dtypes
```

Out[15]:
```
X0        object
X1        object
X2        object
X3        object
X4        object
          ...
X380    category
X382    category
X383    category
X384    category
X385    category
Length: 376, dtype: object
```

In [16]:
```
x_train2=x_train1
```

In [17]:
```
x_train2.dtypes
```

Out[17]:
```
X0        object
X1        object
X2        object
X3        object
X4        object
```

```
          ...
X380    category
X382    category
X383    category
X384    category
X385    category
Length: 376, dtype: object
```

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

To use VarianceThreshold(threshold=0.0) method of sklearn we need to change all the features to

numerical form as it does not work with string or object data

We are using OrdinalEncoder() rather than labelEndoder(). As labelEndoder() needs to be applied indivisually to each column but OrdinalEncoder() can be applied to the whole dataframe directly

In [18]:
```
## Import ordinal encoder
from sklearn.preprocessing import OrdinalEncoder
# label_encoder object knows how to understand word labels.
enc = OrdinalEncoder()
# Encode labels
x_train2_OrdinalEncoded=enc.fit_transform(x_train2)
```

In [19]:
```
x_train2.dtypes
```

Out[19]:
```
X0        object
X1        object
X2        object
X3        object
X4        object
          ...
X380    category
X382    category
X383    category
X384    category
X385    category
Length: 376, dtype: object
```

In [20]:
```
x_train2_OrdinalEncoded.shape
```

Out[20]:
```
(4209, 376)
```

In [21]:
```
x_train3_OrdinalEncoded=pd.DataFrame(x_train2_OrdinalEncoded, columns=x_train2.colum
x_train3_OrdinalEncoded
```

Out[21]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X
0	32.0	23.0	17.0	0.0	3.0	24.0	9.0	14.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
1	32.0	21.0	19.0	4.0	3.0	28.0	11.0	14.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	
2	20.0	24.0	34.0	2.0	3.0	27.0	9.0	23.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
3	20.0	21.0	34.0	5.0	3.0	27.0	11.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4	20.0	23.0	34.0	5.0	3.0	12.0	3.0	13.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X
4204	8.0	20.0	16.0	2.0	3.0	0.0	3.0	16.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	
4205	31.0	16.0	40.0	3.0	3.0	0.0	7.0	7.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	
4206	8.0	23.0	38.0	0.0	3.0	0.0	6.0	4.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
4207	9.0	19.0	25.0	5.0	3.0	0.0	11.0	20.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4208	46.0	19.0	3.0	2.0	3.0	0.0	6.0	22.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	

4209 rows × 376 columns

In [22]:
```
x_train3_OrdinalEncoded.dtypes
```

Out[22]:
```
X0      float64
X1      float64
X2      float64
X3      float64
X4      float64
         ...
X380    float64
X382    float64
X383    float64
X384    float64
X385    float64
Length: 376, dtype: object
```

In [27]:
```
###Using VarianceThreshold(threshold=0.0) to check the columns with variance less th
selector = VarianceThreshold(threshold=0.0)#Here threshold is kept 0 as per problem
x_train_LVR=selector.fit(x_train3_OrdinalEncoded)
```

In [28]:
```
x_train3_OrdinalEncoded.columns[x_train_LVR.get_support()]
```

Out[28]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10', 'X12',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=364)
```

In [29]:
```
#Getting the columns with 0 variance
columns_with_Zero_VR=[columns for columns in x_train3_OrdinalEncoded.columns
                      if columns not in x_train3_OrdinalEncoded.columns[x_train_LVR.g
len(columns_with_Zero_VR)
```

Out[29]:
```
12
```

In [30]:
```
columns_with_Zero_VR
```

Out[30]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
```

```
'X297',
'X330',
'X347']
```

In [31]:
```
#Droping the columns with Zero Variance
x_train3_OrdinalEncoded.drop(columns_with_Zero_VR, axis=1,inplace=True)
x_train3_OrdinalEncoded
```

Out[31]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X
0	32.0	23.0	17.0	0.0	3.0	24.0	9.0	14.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
1	32.0	21.0	19.0	4.0	3.0	28.0	11.0	14.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	
2	20.0	24.0	34.0	2.0	3.0	27.0	9.0	23.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
3	20.0	21.0	34.0	5.0	3.0	27.0	11.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4	20.0	23.0	34.0	5.0	3.0	12.0	3.0	13.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	8.0	20.0	16.0	2.0	3.0	0.0	3.0	16.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	
4205	31.0	16.0	40.0	3.0	3.0	0.0	7.0	7.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	
4206	8.0	23.0	38.0	0.0	3.0	0.0	6.0	4.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.0	
4207	9.0	19.0	25.0	5.0	3.0	0.0	11.0	20.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4208	46.0	19.0	3.0	2.0	3.0	0.0	6.0	22.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	

4209 rows × 364 columns

In [32]:
```
376-364
```

Out[32]: 12

In [33]:
```
#Calling the unique funtion we created above to covert the unique colums <=2 to cate
unique(x_test)
```

Out[33]:
```
(Index(['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
        ...
        'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
        'X385'],
       dtype='object', length=368),
 X0        object
 X1        object
 X2        object
 X3        object
 X4        object
           ...
 X380    category
 X382    category
 X383    category
 X384    category
 X385    category
 Length: 376, dtype: object)
```

In [34]:
```
### Converting Test data as per processing of Train data
from sklearn.preprocessing import OrdinalEncoder
```

```
enc1 = OrdinalEncoder()
x_test2_OrdinalEncoded=enc1.fit_transform(x_test)
```

In [35]:
```
x_test2_OrdinalEncoded.shape
```

Out[35]: (4209, 376)

In [36]:
```
x_test3_OrdinalEncoded=pd.DataFrame(x_test2_OrdinalEncoded, columns=dfRawTest.column
x_test3_OrdinalEncoded.head()
```

Out[36]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X38(
0	21.0	23.0	34.0	5.0	3.0	26.0	0.0	22.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.(
1	42.0	3.0	8.0	0.0	3.0	9.0	6.0	24.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.(
2	21.0	23.0	17.0	5.0	3.0	0.0	9.0	9.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.(
3	21.0	13.0	34.0	5.0	3.0	31.0	11.0	13.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.(
4	45.0	20.0	17.0	2.0	3.0	30.0	8.0	12.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.(

5 rows × 376 columns

In [37]:
```
x_test3_OrdinalEncoded.dtypes
```

Out[37]:
```
X0      float64
X1      float64
X2      float64
X3      float64
X4      float64
         ...
X380    float64
X382    float64
X383    float64
X384    float64
X385    float64
Length: 376, dtype: object
```

In [38]:
```
#Droping the same 12 Nos of columns from test datawhich were dropped from the train
#to match the dimension
x_test3_OrdinalEncoded.drop(columns_with_Zero_VR, axis=1,inplace=True)
x_test3_OrdinalEncoded
```

Out[38]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X
0	21.0	23.0	34.0	5.0	3.0	26.0	0.0	22.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	
1	42.0	3.0	8.0	0.0	3.0	9.0	6.0	24.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
2	21.0	23.0	17.0	5.0	3.0	0.0	9.0	9.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	
3	21.0	13.0	34.0	5.0	3.0	31.0	11.0	13.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	
4	45.0	20.0	17.0	2.0	3.0	30.0	8.0	12.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	6.0	9.0	17.0	5.0	3.0	1.0	9.0	4.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X
4205	42.0	1.0	8.0	3.0	3.0	1.0	9.0	24.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	
4206	47.0	23.0	17.0	5.0	3.0	1.0	3.0	22.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
4207	7.0	23.0	17.0	0.0	3.0	1.0	2.0	16.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
4208	42.0	1.0	8.0	2.0	3.0	1.0	6.0	17.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	

4209 rows × 364 columns

In [39]:
```
x_test3_OrdinalEncoded.shape,x_train3_OrdinalEncoded.shape
```

Out[39]: ((4209, 364), (4209, 364))

Both the test and train data have the same shape

Dimesion reduction through PCA

In [40]:
```
#Scalling for Train data
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
Scalling=MinMaxScaler()
#Scalling=StandardScaler()
x_scalled=Scalling.fit_transform(x_train3_OrdinalEncoded)
x_scalled_df=pd.DataFrame(x_scalled, columns=x_train3_OrdinalEncoded.columns, index=
x_scalled_df.head()
```

Out[40]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X
0	0.695652	0.884615	0.395349	0.000000	1.0	0.857143	0.818182	0.583333	0.0	0.0	...	0.0	
1	0.695652	0.807692	0.441860	0.666667	1.0	1.000000	1.000000	0.583333	0.0	0.0	...	1.0	
2	0.434783	0.923077	0.790698	0.333333	1.0	0.964286	0.818182	0.958333	0.0	0.0	...	0.0	
3	0.434783	0.807692	0.790698	0.833333	1.0	0.964286	1.000000	0.166667	0.0	0.0	...	0.0	
4	0.434783	0.884615	0.790698	0.833333	1.0	0.428571	0.272727	0.541667	0.0	0.0	...	0.0	

5 rows × 364 columns

PCA for dimensionality reduction

If 0 < n_components < 1 and svd_solver == 'full', select the number of components such that
the amount of variance that needs to be explained is greater than the percentage specified
by n_components.

link https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

In [41]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components = .99)#Giving input directly amount of variace I want ot caputr
x_train2_trans= pca.fit_transform(x_scalled_df)
```

In [42]:
```
x_train2_trans.shape, x_train3_OrdinalEncoded.shape
```

Out[42]:
```
((4209, 134), (4209, 364))
```

In [43]:
```
print(f"Nos of component to capture 99% of variance is {x_train2_trans.shape[1]}")
```

```
Nos of component to capture 99% of variance is 134
```

PCA for test data

In [44]:
```
#Scalling test data
x_scalled_test=Scalling.transform(x_test3_OrdinalEncoded)
x_scalled_df_test=pd.DataFrame(x_scalled_test, columns=x_test3_OrdinalEncoded.column
x_scalled_df_test.head()
```

Out[44]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X
0	0.456522	0.884615	0.790698	0.833333	1.0	0.928571	0.000000	0.916667	0.0	0.0	...	0.0	
1	0.913043	0.115385	0.186047	0.000000	1.0	0.321429	0.545455	1.000000	0.0	0.0	...	0.0	
2	0.456522	0.884615	0.395349	0.833333	1.0	0.000000	0.818182	0.375000	0.0	0.0	...	0.0	
3	0.456522	0.500000	0.790698	0.833333	1.0	1.107143	1.000000	0.541667	0.0	0.0	...	0.0	
4	0.978261	0.769231	0.395349	0.333333	1.0	1.071429	0.727273	0.500000	0.0	0.0	...	1.0	

5 rows × 364 columns

In [45]:
```
# Doing PCA with the PCA instance created for Train data
x_test2_trans= pca.transform(x_scalled_df_test)
```

In [46]:
```
 x_train2_trans.shape,x_test2_trans.shape
```

Out[46]:
```
((4209, 134), (4209, 134))
```

Now both Train and test data is ready for the model

In [47]:
```
from xgboost import XGBRegressor
import xgboost as xgb
```

In [48]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_train2_trans, y_target1, test_
```

In [92]:
```
xg=XGBRegressor()
xg.fit(x_train,y_train)
```

Out[92]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
```

```
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [93]:
```
y_pred = xg.predict(x_test)
```

In [94]:
```
#Creating a function for RMSLE
from sklearn.metrics import mean_squared_log_error, mean_absolute_error
def RMSLE (y_test,y_pred):
    return np.mean(mean_squared_log_error(y_test,y_pred))
#Creating funtion to display the score
def score(model):
    train_preds=model.predict(x_train)
    test_preds=model.predict(x_test)
    score={"Train MAE":mean_absolute_error(y_train, train_preds),
           "Test MAE":mean_absolute_error(y_test,test_preds),
           "Train RMSLE":RMSLE(y_train,train_preds),
           "Test RMSLE":RMSLE(y_test, test_preds),
           "Train R^2":model.score(x_train,y_train),
           "Test R^2":model.score(x_test,y_test)}
    return score
```

In [95]:
```
score(xg)
```

Out[95]:
```
{'Train MAE': 1.4297480070127164,
 'Test MAE': 6.505379767338623,
 'Train RMSLE': 0.0005128565208799913,
 'Test RMSLE': 0.0075627461778320455,
 'Train R^2': 0.9630272262776178,
 'Test R^2': 0.44843124871369744}
```

In [62]:
```
clf = XGBRegressor(n_estimators = 500)

clf.fit(x_train, y_train,
        eval_set = [(x_train, y_train), (x_test, y_test)],
        eval_metric = "rmse",
        early_stopping_rounds = 10, verbose=50)
```

```
[0]     validation_0-rmse:71.10375      validation_1-rmse:71.06438
[25]    validation_0-rmse:4.79400       validation_1-rmse:9.26920
```

Out[62]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=500, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [63]:
```
clf.best_score
```

Out[63]:
```
9.096573
```

In [64]:
```
clf.best_iteration
```

Out[64]: 15

In [80]:
```
score(clf)
```

Out[80]:
```
{'Train MAE': 4.043809234283782,
 'Test MAE': 6.119319904173355,
 'Train RMSLE': 0.0028665135729865994,
 'Test RMSLE': 0.006859134276009143,
 'Train R^2': 0.7841037434605267,
 'Test R^2': 0.4933959698510477}
```

Randomised Search CV

In [69]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error,make_scorer
ftwo_scorer = make_scorer(mean_absolute_error)
model = XGBRegressor(eval_metric = 'rmse')

param_grid = {
        'max_depth': np.arange(.01,1,.05),
        'min_child_weight': np.arange(1,15,1),
        'gamma': np.arange(0.0,10.0,0.005),
        'learning_rate': np.arange(0.1,1,.5),
        'subsample': np.arange(0.01,1.0,0.01),
        'colsample_bylevel': np.round(np.arange(0.1,1.0,0.01)),
        'colsample_bytree': np.arange(0.1,1.0,0.01)
}
```

In [73]:
```
parameters = {
         "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
         "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
         "min_child_weight" : [ 1, 3, 5, 7 ],
         "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
         "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
    }
```

In [74]:
```
kfold=KFold(n_splits=5, shuffle=True, random_state=10)
grid_search = RandomizedSearchCV(model, parameters, n_iter = 500, cv=kfold)
grid_result = grid_search.fit(x_train2_trans,y_target1)
```

In [75]:
```
grid_result.best_estimator_
```

Out[75]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False,
             eval_metric='rmse', gamma=0.3, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.05, max_delta_step=0,
             max_depth=5, min_child_weight=7, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [77]:
```
grid_result.best_params_
```

Out[77]:
```
{'min_child_weight': 7,
 'max_depth': 5,
 'learning_rate': 0.05,
```

```
 'gamma': 0.3,
 'colsample_bytree': 0.7}
```

grid_result.best*params*

{'min_child_weight': 7, 'max_depth': 5, 'learning_rate': 0.05, 'gamma': 0.3, 'colsample_bytree': 0.7}

grid_result.best*estimator*

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False, eval_metric='rmse', gamma=0.3, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.05, max_delta_step=0, max_depth=5, min_child_weight=7, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=8, num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)

In [97]:
```
score(grid_result)
```

Out[97]:
```
{'Train MAE': 4.525880482432607,
 'Test MAE': 4.696882171268418,
 'Train RMSLE': 0.003795848512199609,
 'Test RMSLE': 0.004098759670841517,
 'Train R^2': 0.695668771890033,
 'Test R^2': 0.6835244294478833}
```

We can see that mostly the Training and Test Evaluation scores are similar. So model is not over fitting

GridSearchCV is not perfomed due to time and computing factor

In [96]:
```
score(xg)
```

Out[96]:
```
{'Train MAE': 1.4297480070127164,
 'Test MAE': 6.505379767338623,
 'Train RMSLE': 0.0005128565208799913,
 'Test RMSLE': 0.0075627461778320455,
 'Train R^2': 0.9630272262776178,
 'Test R^2': 0.44843124871369744}
```

RandomisedSearchCV is working far better than our default model

In [103…
```
# Predicting on test set
p_test = grid_result.predict(x_test2_trans)
p_test
```

Out[103…
```
array([ 77.403336,  92.90406 ,  78.10075 , ...,  93.66875 , 103.10958 ,
        92.93207 ], dtype=float32)
```

In [104…
```
Predicted_Data = pd.DataFrame()
Predicted_Data['y'] = p_test
Predicted_Data.head()
```

Out[104…

	y
0	77.403336

	y
1	92.904060
2	78.100754
3	77.078545
4	110.154442

Submitted by Mintu Medhi

In [ ]: