In [1]:
```python
#import required libraries
import numpy as np
import pandas as pd
```

In [3]:
```python
#Load train dataset
data=pd.read_csv('train.csv')
datatest=pd.read_csv('test.csv')
data.head()
```

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [4]:
```python
#Get details for data

data.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
```

In [6]:
```python
#Remove columns with variance 0 as those columns will not impact the prediction and mostly those will have constant data
#to reduce number of cols remve those var=0 cols

data.var()[data.var()==0].index.values
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_7712\312826853.py:4: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  data.var()[data.var()==0].index.values
```

Out[6]:
```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

In [8]:
```python
#drop 0 var columns
datafinal=data.drop(data.var()[data.var()==0].index.values, axis=1)
datafinal.info()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_7712\2472663354.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  datafinal=data.drop(data.var()[data.var()==0].index.values, axis=1)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 366 entries, ID to X385
dtypes: float64(1), int64(357), object(8)
memory usage: 11.8+ MB
```

In [9]:
```python
datafinal.head()
```

Out[9]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0	0	0

5 rows × 366 columns

In [10]:
```python
#get features and labels

features=datafinal.iloc[:,2:]
label=datafinal.iloc[:,1].values
```

In [11]:
```python
features.head()
```

Out[11]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	k	v	at	a	d	u	j	o	0	0	...	0	0	1	0	0	0	0	0	0	0
1	k	t	av	e	d	y	l	o	0	0	...	1	0	0	0	0	0	0	0	0	0
2	az	w	n	c	d	x	j	x	0	0	...	0	0	0	0	0	0	1	0	0	0
3	az	t	n	f	d	x	l	e	0	0	...	0	0	0	0	0	0	0	0	0	0
4	az	v	n	f	d	h	d	n	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 364 columns

In [12]:
```python
label
```

Out[12]:
```
array([130.81,  88.53,  76.26, ..., 109.22,  87.48, 110.85])
```

In [13]:
```python
#Get object columns from feature column for LE and OHE

features.describe(include="O")#Get object columns from feature column for LE and OHE

features.describe(include="O")
```

Out[13]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

In [14]:
```python
#Get only object column names

objcols=features.describe(include=['object']).columns.values
objcols
```

Out[14]:
```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)
```

In [15]:
```python
#Apply Label Encoder  for object columns. OHE is not use das this will create PC's instead of using direct variables

from sklearn.preprocessing import LabelEncoder
#from sklearn.preprocessing import OneHotEncoder

le=LabelEncoder()

for i in objcols:
  features[i]= le.fit_transform(features[i])

#ohe=OneHotEncoder(categorical_features=[0,1,2,3,4,5,6,8])

fea=features.values
#featureohe=fea #ohe.fit_transform(fea).toarray()

#featureohe
fea
```

Out[15]:
```
array([[32, 23, 17, ...,  0,  0,  0],
       [32, 21, 19, ...,  0,  0,  0],
       [20, 24, 34, ...,  0,  0,  0],
       ...,
       [ 8, 23, 38, ...,  0,  0,  0],
       [ 9, 19, 25, ...,  0,  0,  0],
       [46, 19,  3, ...,  0,  0,  0]], dtype=int64)
```

In [16]:
```python
#Standardize the data using StandardScaler

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
featuresstd = sc.fit_transform(fea)
featuresstd
```

Out[16]:
```
array([[ 0.16301209,  1.39348787, -0.02812155, ..., -0.04081511,
        -0.02180363, -0.03778296],
       [ 0.16301209,  1.15902093,  0.15538793, ..., -0.04081511,
        -0.02180363, -0.03778296],
       [-0.71055977,  1.51072134,  1.53170902, ..., -0.04081511,
        -0.02180363, -0.03778296],
       ...,
       [-1.58413164,  1.39348787,  1.89872798, ..., -0.04081511,
        -0.02180363, -0.03778296],
       [-1.51133398,  0.924554  ,  0.70591637, ..., -0.04081511,
        -0.02180363, -0.03778296],
       [ 1.18217927,  0.924554  , -1.31268791, ..., -0.04081511,
        -0.02180363, -0.03778296]])
```

In [17]:
```python
#USe PCA for dimensionality reduction
#Use no of components as ALL column to identify correct number of componnets
from sklearn.decomposition import PCA

pca = PCA(n_components=364, svd_solver='full')
pca.fit(featuresstd,label)
```

Out[17]:
```
PCA(n_components=364, svd_solver='full')
```

In [18]:
```python
#check explained variance ratio
pca.explained_variance_ratio_
```

Out[18]:
```
array([6.89266892e-02, 5.68841213e-02, 4.53745695e-02, 3.42677135e-02,
       3.26430877e-02, 3.16266067e-02, 2.86252442e-02, 2.12375123e-02,
       1.97041337e-02, 1.78319550e-02, 1.64006646e-02, 1.56428622e-02,
       1.46274226e-02, 1.44833491e-02, 1.34828580e-02, 1.29516175e-02,
       1.24383219e-02, 1.17310333e-02, 1.12105051e-02, 1.07727793e-02,
       9.92512898e-03, 9.69449157e-03, 9.42523076e-03, 9.09867975e-03,
       8.74223908e-03, 8.43069482e-03, 7.90204943e-03, 7.63217614e-03,
       7.33541876e-03, 7.14905572e-03, 6.92957269e-03, 6.76677298e-03,
       6.52534438e-03, 6.41494089e-03, 6.22692040e-03, 5.99196153e-03,
       5.88086185e-03, 5.74693562e-03, 5.63702738e-03, 5.53409616e-03,
       5.50641149e-03, 5.40082599e-03, 5.33910600e-03, 5.24611598e-03,
       5.10194247e-03, 5.03232420e-03, 4.95885463e-03, 4.72691565e-03,
       4.64475228e-03, 4.56521484e-03, 4.39624472e-03, 4.32947533e-03,
       4.30322432e-03, 4.23762131e-03, 4.20211958e-03, 4.15473420e-03,
       4.06699537e-03, 4.03462750e-03, 3.91873995e-03, 3.88747084e-03,
       3.81759321e-03, 3.75560105e-03, 3.72440051e-03, 3.65910562e-03,
       3.59566547e-03, 3.55229656e-03, 3.49700953e-03, 3.46182278e-03,
       3.40471717e-03, 3.34157815e-03, 3.30984908e-03, 3.25521932e-03,
       3.24084510e-03, 3.21045090e-03, 3.16671200e-03, 3.16169105e-03,
       3.09963730e-03, 3.07579140e-03, 3.05317642e-03, 3.03955238e-03,
       3.00255847e-03, 2.98903148e-03, 2.95834824e-03, 2.92906873e-03,
       2.90789282e-03, 2.89246840e-03, 2.87088871e-03, 2.84861071e-03,
       2.82893540e-03, 2.80661865e-03, 2.79489424e-03, 2.76964748e-03,
       2.74977677e-03, 2.72538185e-03, 2.72396743e-03, 2.66915947e-03,
       2.64849773e-03, 2.63035478e-03, 2.60995440e-03, 2.60665425e-03,
       2.57383600e-03, 2.54968974e-03, 2.53621249e-03, 2.51953928e-03,
       2.50398175e-03, 2.48457976e-03, 2.44053200e-03, 2.42183601e-03,
       2.40353580e-03, 2.38432335e-03, 2.34517965e-03, 2.31166118e-03,
       2.30049772e-03, 2.27223313e-03, 2.25613123e-03, 2.23532575e-03,
       2.21465563e-03, 2.20028258e-03, 2.15055308e-03, 2.13667795e-03,
       2.10335610e-03, 2.09410362e-03, 2.05842058e-03, 2.04122098e-03,
       2.02468809e-03, 1.98912003e-03, 1.93876262e-03, 1.91967571e-03,
       1.91801301e-03, 1.87935329e-03, 1.85171337e-03, 1.81419321e-03,
       1.79357485e-03, 1.78621289e-03, 1.75392321e-03, 1.71752943e-03,
       1.70719881e-03, 1.67983148e-03, 1.65808186e-03, 1.61860975e-03,
       1.61139410e-03, 1.57706719e-03, 1.54261681e-03, 1.53538284e-03,
       1.49678823e-03, 1.49384890e-03, 1.47852266e-03, 1.42828100e-03,
       1.41108081e-03, 1.37376476e-03, 1.33814992e-03, 1.28803283e-03,
       1.26972355e-03, 1.26213163e-03, 1.20721187e-03, 1.16191771e-03,
       1.14336841e-03, 1.12752230e-03, 1.09498612e-03, 1.05869705e-03,
       9.99910624e-04, 9.89110199e-04, 9.86361463e-04, 9.54546927e-04,
       9.45056062e-04, 9.11514885e-04, 8.99355342e-04, 8.80886749e-04,
       8.63653466e-04, 8.59439761e-04, 8.29410991e-04, 8.24059997e-04,
       7.98201807e-04, 7.91126650e-04, 7.78713087e-04, 7.44845247e-04,
       7.24330743e-04, 7.08041140e-04, 6.85829934e-04, 6.72145117e-04,
       6.55718352e-04, 6.40733883e-04, 6.12000447e-04, 5.86075445e-04,
       5.58289468e-04, 5.45679895e-04, 5.41944228e-04, 5.30507026e-04,
       5.23125193e-04, 5.09290043e-04, 4.93967305e-04, 4.75601629e-04,
       4.59394696e-04, 4.52812516e-04, 4.39558529e-04, 4.26949429e-04,
       4.14907621e-04, 4.07139378e-04, 3.99902280e-04, 3.96288548e-04,
       3.86933450e-04, 3.72623799e-04, 3.62443593e-04, 3.54579912e-04,
       3.38415435e-04, 3.29616574e-04, 3.16880863e-04, 3.07011359e-04,
       2.91609975e-04, 2.84735221e-04, 2.74172380e-04, 2.65105542e-04,
       2.57082857e-04, 2.44896369e-04, 2.37787938e-04, 2.29725881e-04,
       2.28237225e-04, 2.26852369e-04, 2.18016571e-04, 2.09597981e-04,
       1.98544338e-04, 1.74896477e-04, 1.63275485e-04, 1.56435191e-04,
       1.48275745e-04, 1.45817565e-04, 1.39375105e-04, 1.25010833e-04,
       1.16516633e-04, 1.08763512e-04, 1.00668042e-04, 9.32800134e-05,
       8.37733596e-05, 7.81804284e-05, 6.89060898e-05, 6.38124700e-05,
       6.20529750e-05, 5.69132679e-05, 4.99048590e-05, 4.85560153e-05,
       4.49553468e-05, 4.31149301e-05, 4.23039470e-05, 3.76702109e-05,
       3.67759864e-05, 3.16512054e-05, 3.01120917e-05, 2.60667672e-05,
       2.04349916e-05, 1.43314337e-05, 1.24786542e-05, 1.17912074e-05,
       8.13427984e-06, 7.14835152e-06, 4.73619419e-06, 4.54880789e-06,
       3.00516288e-06, 2.48700880e-06, 1.42103632e-06, 1.35933926e-32,
       8.38029781e-33, 5.08999239e-33, 4.55497042e-33, 3.76627450e-33,
       3.23823417e-33, 3.01837632e-33, 2.76550273e-33, 2.50515637e-33,
       2.38122953e-33, 2.26135014e-33, 2.21923567e-33, 2.08457005e-33,
       2.03920062e-33, 1.85142658e-33, 1.71959278e-33, 1.67996255e-33,
       1.53304700e-33, 1.41630040e-33, 1.28870373e-33, 1.22654169e-33,
       1.11705940e-33, 1.04317009e-33, 8.93864219e-34, 8.84553942e-34,
       8.34617072e-34, 8.08473788e-34, 6.82770110e-34, 5.84253396e-34,
       4.94727921e-34, 4.71849301e-34, 4.08359405e-34, 4.03440250e-34,
       4.00385439e-34, 3.65288464e-34, 3.27612038e-34, 3.15784019e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.12692316e-34, 3.12692316e-34, 3.12692316e-34,
       3.12692316e-34, 3.11581852e-34, 2.57830148e-34, 1.79267127e-34,
       1.36805395e-34, 1.28001992e-34, 1.17151312e-34, 7.66074370e-35,
       7.28333644e-35, 3.79310451e-35, 1.04570734e-35, 4.87445840e-37])
```

In [19]:
```python
#Get mean of explained variance ratio to get the threshold value for varince
np.mean(pca.explained_variance_ratio_)#Get mean of explained variance ratio to get the threshold value for varince
np.mean(pca.explained_variance_ratio_)
```

Out[19]:
```
0.002747252747252748
```

In [20]:
```python
#Check number of components which has explained variance >threshold value
pca.explained_variance_ratio_>0.002747252747252748
```

Out[20]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False])
```

In [22]:
```python
data.var()[data.var()==0].index.values
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_7712\1856094577.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  data.var()[data.var()==0].index.values
```

Out[22]:
```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

In [23]:
```python
print(len(pca.explained_variance_ratio_[pca.explained_variance_ratio_>0.002747252747252748]))
```

```
93
```

In [24]:
```python
#PCA with n components 93 which has explained variance ratio >threshold

pca = PCA(n_components=93,svd_solver='full')
pca.fit(featuresstd,label)

finalFeatures = pca.transform(featuresstd)
finalFeatures.shape
```

Out[24]:
```
(4209, 93)
```

In [25]:
```python
finalFeatures
```

Out[25]:
```
array([[12.25446667, -2.94420951, -0.95784987, ...,  0.86912516,
        -0.73892033, -0.80982148],
       [-0.10254567,  0.36606164,  0.99722405, ...,  0.38555584,
        -2.33082016, -0.62422555],
       [10.27831815, 21.10492781, -5.01329942, ...,  0.08878323,
        -0.28542568, -1.90053779],
       ...,
       [ 0.43696828,  0.89752935,  3.44522225, ..., -0.55307597,
         0.05632946, -0.13451454],
       [-1.34167187,  0.58925266, -0.10965489, ..., -0.75894816,
        -0.12206212,  0.02963942],
       [-2.16821336, -1.06043018, -0.28807512, ..., -0.15311066,
        -0.34216957, -0.3637978 ]])
```

In [44]:
```python
label.shape
```

Out[44]:
```
(4209,)
```

In [50]:
```python
datatest.head()
```

Out[50]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

In [51]:
```python
#Remove same columns as we did for tain dataset
#'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
      # 'X293', 'X297', 'X330', 'X347'

datafinal=datatest.drop(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290','X293', 'X297', 'X330', 'X347'], axis=1)
datafinal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 365 entries, ID to X385
dtypes: int64(357), object(8)
memory usage: 11.7+ MB
```

In [52]:
```python
#Get feature columns

testFeatures=datafinal.iloc[:,1:]

#get object columns

objcolstest=testFeatures.describe(include=['object']).columns.values
objcolstest
```

Out[52]:
```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)
```

In [53]:
```python
#Apply Label Encoder  for object columns.

from sklearn.preprocessing import LabelEncoder
#from sklearn.preprocessing import OneHotEncoder

le=LabelEncoder()

for i in objcolstest:
  testFeatures[i]= le.fit_transform(testFeatures[i])

f1=testFeatures.values


f1.ndim
```

Out[53]:
```
2
```

In [65]:
```python
testdataforfile=pd.concat([datatest.iloc[:,0],pd.concat([pd.DataFrame(data=label, columns=['y']),datatest.iloc[:,1:]], axis=1)],axis=1)

testdataforfile.head()
```

Out[65]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	130.81	az	v	n	f	d	t	a	w	...	0	0	0	1	0	0	0	0	0	0
1	2	88.53	t	b	ai	a	d	b	g	y	...	0	0	1	0	0	0	0	0	0	0
2	3	76.26	az	v	as	f	d	a	j	j	...	0	0	0	1	0	0	0	0	0	0
3	4	80.62	az	l	n	f	d	z	l	n	...	0	0	0	1	0	0	0	0	0	0
4	5	78.02	w	s	as	c	d	y	i	m	...	1	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [66]:
```python
testdataforfile.to_csv("testwithpredvalue.csv",index=False)
```