In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
train_df = pd.read_csv('train_mercdz.csv')
```

In [3]:
```
test_df = pd.read_csv('test_mercdz.csv')
```

In [4]:
```
train_df.head()
```

Out[4]:

	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 [5]:
```
test_df.head()
```

Out[5]:

	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 [6]:
```
train_df.shape, test_df.shape
```

Out[6]:
```
((4209, 378), (4209, 377))
```

In [7]:
```
train_df.dtypes, test_df.dtypes
```

Out[7]:
```
(ID        int64
 y       float64
 X0       object
 X1       object
 X2       object
          ...
 X380      int64
 X382      int64
 X383      int64
 X384      int64
 X385      int64
 Length: 378, dtype: object,
 ID       int64
 X0      object
 X1      object
 X2      object
 X3      object
          ...
 X380     int64
 X382     int64
 X383     int64
 X384     int64
 X385     int64
 Length: 377, dtype: object)
```

In [8]:
```
train_df.var(), test_df.var()
```

Out[8]:
```
(ID      5.941936e+06
 y       1.607667e+02
 X10     1.313092e-02
 X11     0.000000e+00
 X12     6.945713e-02
             ...
 X380    8.014579e-03
 X382    7.546747e-03
 X383    1.660732e-03
 X384    4.750593e-04
 X385    1.423823e-03
 Length: 370, dtype: float64,
 ID      5.871311e+06
 X10     1.865006e-02
 X11     2.375861e-04
 X12     6.885074e-02
 X13     5.734498e-02
             ...
 X380    8.014579e-03
 X382    8.715481e-03
 X383    4.750593e-04
 X384    7.124196e-04
 X385    1.660732e-03
 Length: 369, dtype: float64)
```

In [9]:
```
train_df.columns, test_df.columns
```

Out[9]:
```
(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),
 Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
        ...
        'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
        'X385'],
       dtype='object', length=377))
```

In [10]:
```
## checked for null value in train data and test data
```

In [11]:
```
train_df.isnull().sum().sum(), test_df.isnull().sum().sum()
```

Out[11]:
```
(0, 0)
```

In [12]:
```
dtype_data=train_df.dtypes.reset_index()
dtype_data.columns = ["Count", "Column Type"]
dtype_data.groupby("Column Type").aggregate('count').reset_index()
```

Out[12]:

	Column Type	Count
0	int64	369
1	float64	1
2	object	8

In [13]:
```
dtype_data=test_df.dtypes.reset_index()
dtype_data.columns = ["Count", "Column Type"]
dtype_data.groupby("Column Type").aggregate('count').reset_index()
```

Out[13]:

	Column Type	Count
0	int64	369
1	object	8

In [14]:
```
# since majority of the columns are integers with 8 categorical columns and 1 float column
 (target variable) in train data
# in test data 8 categorical columns
```

In [15]:
```
train_df.iloc[:15,:15]
```

Out[15]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0
1	6	88.53	k	t	av	e	d	y	l	o	0	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
5	18	92.93	t	b	e	c	d	g	h	s	0	0	0	0	1
6	24	128.76	al	r	e	f	d	f	h	s	0	0	0	0	1
7	25	91.91	o	l	as	f	d	f	j	a	0	0	0	0	1
8	27	108.67	w	s	as	e	d	f	i	h	0	0	0	0	1
9	30	126.99	j	b	aq	c	d	f	a	e	0	0	0	0	1
10	31	102.09	h	r	r	f	d	f	h	p	0	0	1	0	0
11	32	98.12	al	r	e	f	d	f	h	o	0	0	0	0	1
12	34	82.62	s	b	ai	c	d	f	g	m	0	0	0	0	0
13	36	94.12	al	r	e	f	d	j	h	o	0	0	0	0	1
14	37	99.15	o	s	as	e	d	j	g	m	0	0	0	0	1

In [16]:
```
test_df.iloc[:15,:15]
```

Out[16]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15
0	1	az	v	n	f	d	t	a	w	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	0	0	0	1	0
3	4	az	l	n	f	d	z	l	n	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	0	0	0	1	0
5	8	y	aa	ai	e	d	x	g	s	0	0	0	0	0	0
6	10	x	b	ae	d	d	x	d	y	0	0	0	0	0	0
7	11	f	s	ae	c	d	h	d	a	0	0	0	1	0	0
8	12	ap	l	s	c	d	h	j	n	0	0	0	0	0	0
9	14	o	v	as	f	d	g	f	v	0	0	0	0	1	0
10	15	ap	l	s	c	d	g	d	n	0	0	0	0	0	0
11	16	ay	b	b	a	d	g	l	r	0	0	0	0	0	0
12	17	al	r	e	f	d	g	h	o	0	0	0	0	1	0
13	19	o	v	ae	g	d	g	g	j	0	0	0	0	0	0
14	20	h	a	ak	f	d	g	l	t	0	0	1	0	0	0

In [17]:
```
# checked for the unique value
```

In [18]:
```
obj_dtype = train_df.dtypes[train_df.dtypes=='object'].index
for i in obj_dtype:
    print(i, train_df[i].unique())
```

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

In [19]:
```
obj_dtype = test_df.dtypes[test_df.dtypes=='object'].index
for i in obj_dtype:
    print(i, test_df[i].unique())
```

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

In [20]:
```
#Encoding the catagorical value
```

In [21]:
```
cat_columns = ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [22]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [23]:
```
le0 = LabelEncoder()
le0.fit(list(train_df.X0)+list(test_df.X0))
train_df.X0 = le0.transform(train_df.X0)
test_df.X0 = le0.transform(test_df.X0)
```

In [24]:
```
le1 = LabelEncoder()
le1.fit(list(train_df.X1)+list(test_df.X1))
train_df.X1 = le1.transform(train_df.X1)
test_df.X1 = le1.transform(test_df.X1)
```

In [25]:
```
le2 = LabelEncoder()
le2.fit(list(train_df.X2)+list(test_df.X2))
train_df.X2 = le2.transform(train_df.X2)
test_df.X2 = le2.transform(test_df.X2)
```

In [26]:
```
le3 = LabelEncoder()
le3.fit(list(train_df.X3)+list(test_df.X3))
train_df.X3 = le3.transform(train_df.X3)
test_df.X3 = le3.transform(test_df.X3)
```

In [27]:
```
le4 = LabelEncoder()
le4.fit(list(train_df.X4)+list(test_df.X4))
train_df.X4 = le4.transform(train_df.X4)
test_df.X4 = le4.transform(test_df.X4)
```

In [28]:
```
le5 = LabelEncoder()
le5.fit(list(train_df.X5)+list(test_df.X5))
train_df.X5 = le5.transform(train_df.X5)
test_df.X5 = le5.transform(test_df.X5)
```

In [29]:
```
le6 = LabelEncoder()
le6.fit(list(train_df.X6)+list(test_df.X6))
train_df.X6 = le6.transform(train_df.X6)
test_df.X6 = le6.transform(test_df.X6)
```

In [30]:
```
le8 = LabelEncoder()
le8.fit(list(train_df.X8)+list(test_df.X8))
train_df.X8 = le8.transform(train_df.X8)
test_df.X8 = le8.transform(test_df.X8)
```

In [31]:
```
## Encoded the float value of target variable in train data
```

In [32]:
```
le9 = LabelEncoder()
train_df['y'] =le9.fit_transform(train_df['y'])
```

In [33]:
```
train_df.head()
```

Out[33]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	2466	37	23	20	0	3	27	9	14	...	0	0	1	0	0	0	0	0	0	0
1	6	366	37	21	22	4	3	31	11	14	...	1	0	0	0	0	0	0	0	0	0
2	7	69	24	24	38	2	3	30	9	23	...	0	0	0	0	0	0	1	0	0	0
3	9	133	24	21	38	5	3	30	11	4	...	0	0	0	0	0	0	0	0	0	0
4	13	106	24	23	38	5	3	14	3	13	...	0	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [34]:
```
test_df.head()
```

Out[34]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	24	23	38	5	3	26	0	22	0	...	0	0	0	1	0	0	0	0	0	0
1	2	46	3	9	0	3	9	6	24	0	...	0	0	1	0	0	0	0	0	0	0
2	3	24	23	19	5	3	0	9	9	0	...	0	0	0	1	0	0	0	0	0	0
3	4	24	13	38	5	3	32	11	13	0	...	0	0	0	1	0	0	0	0	0	0
4	5	49	20	19	2	3	31	8	12	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

In [35]:
```
## checked for unique value with zero in train data
```

In [36]:
```
unique_value_dict = {}
for col in train_df.columns:
    if col not in ["ID", "y", "X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        unique_value = str(np.sort(train_df[col].unique()).tolist())
        t_list = unique_value_dict.get(unique_value, [])
        t_list.append(col)
        unique_value_dict[unique_value] = t_list[:]
for unique_val, columns in unique_value_dict.items():
    print("Columns containing the unique values : ",unique_val)
    print(columns)
    print("-------------------------------------------------------------")
```

```
Columns containing the unique values :  [0, 1]
['X10', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20', 'X21', 'X22', 'X23',
'X24', 'X26', 'X27', 'X28', 'X29', 'X30', 'X31', 'X32', 'X33', 'X34', 'X35', 'X36', 'X37', 'X
38', 'X39', 'X40', 'X41', 'X42', 'X43', 'X44', 'X45', 'X46', 'X47', 'X48', 'X49', 'X50', 'X5
1', 'X52', 'X53', 'X54', 'X55', 'X56', 'X57', 'X58', 'X59', 'X60', 'X61', 'X62', 'X63', 'X6
4', 'X65', 'X66', 'X67', 'X68', 'X69', 'X70', 'X71', 'X73', 'X74', 'X75', 'X76', 'X77', 'X7
8', 'X79', 'X80', 'X81', 'X82', 'X83', 'X84', 'X85', 'X86', 'X87', 'X88', 'X89', 'X90', 'X9
1', 'X92', 'X94', 'X95', 'X96', 'X97', 'X98', 'X99', 'X100', 'X101', 'X102', 'X103', 'X104',
'X105', 'X106', 'X108', 'X109', 'X110', 'X111', 'X112', 'X113', 'X114', 'X115', 'X116', 'X11
7', 'X118', 'X119', 'X120', 'X122', 'X123', 'X124', 'X125', 'X126', 'X127', 'X128', 'X129',
'X130', 'X131', 'X132', 'X133', 'X134', 'X135', 'X136', 'X137', 'X138', 'X139', 'X140', 'X14
1', 'X142', 'X143', 'X144', 'X145', 'X146', 'X147', 'X148', 'X150', 'X151', 'X152', 'X153',
'X154', 'X155', 'X156', 'X157', 'X158', 'X159', 'X160', 'X161', 'X162', 'X163', 'X164', 'X16
5', 'X166', 'X167', 'X168', 'X169', 'X170', 'X171', 'X172', 'X173', 'X174', 'X175', 'X176',
'X177', 'X178', 'X179', 'X180', 'X181', 'X182', 'X183', 'X184', 'X185', 'X186', 'X187', 'X18
9', 'X190', 'X191', 'X192', 'X194', 'X195', 'X196', 'X197', 'X198', 'X199', 'X200', 'X201',
'X202', 'X203', 'X204', 'X205', 'X206', 'X207', 'X208', 'X209', 'X210', 'X211', 'X212', 'X21
3', 'X214', 'X215', 'X216', 'X217', 'X218', 'X219', 'X220', 'X221', 'X222', 'X223', 'X224',
'X225', 'X226', 'X227', 'X228', 'X229', 'X230', 'X231', 'X232', 'X234', 'X236', 'X237', 'X23
8', 'X239', 'X240', 'X241', 'X242', 'X243', 'X244', 'X245', 'X246', 'X247', 'X248', 'X249',
'X250', 'X251', 'X252', 'X253', 'X254', 'X255', 'X256', 'X257', 'X258', 'X259', 'X260', 'X26
1', 'X262', 'X263', 'X264', 'X265', 'X266', 'X267', 'X269', 'X270', 'X271', 'X272', 'X273',
'X274', 'X275', 'X276', 'X277', 'X278', 'X279', 'X280', 'X281', 'X282', 'X283', 'X284', 'X28
5', 'X286', 'X287', 'X288', 'X291', 'X292', 'X294', 'X295', 'X296', 'X298', 'X299', 'X300',
'X301', 'X302', 'X304', 'X305', 'X306', 'X307', 'X308', 'X309', 'X310', 'X311', 'X312', 'X31
3', 'X314', 'X315', 'X316', 'X317', 'X318', 'X319', 'X320', 'X321', 'X322', 'X323', 'X324',
'X325', 'X326', 'X327', 'X328', 'X329', 'X331', 'X332', 'X333', 'X334', 'X335', 'X336', 'X33
7', 'X338', 'X339', 'X340', 'X341', 'X342', 'X343', 'X344', 'X345', 'X346', 'X348', 'X349',
'X350', 'X351', 'X352', 'X353', 'X354', 'X355', 'X356', 'X357', 'X358', 'X359', 'X360', 'X36
1', 'X362', 'X363', 'X364', 'X365', 'X366', 'X367', 'X368', 'X369', 'X370', 'X371', 'X372',
'X373', 'X374', 'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384', 'X38
5']
-------------------------------------------------------------
Columns containing the unique values :  [0]
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X34
7']
-------------------------------------------------------------
```

In [37]:
```
# removed the columns with unique values zero and prepared features and target
```

In [38]:
```
features = train_df.drop(['y','ID','X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X2
90', 'X293', 'X297', 'X330', 'X347'],axis=1)
target = train_df[['y']]
```

In [39]:
```
features.shape,target.shape
```

Out[39]:
```
((4209, 364), (4209, 1))
```

In [40]:
```
from sklearn.model_selection import train_test_split
```

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(features,target, train_size=.8)
```

In [42]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[42]:
```
((3367, 364), (842, 364), (3367, 1), (842, 1))
```

In [43]:
```
## perform dimentional reduction
```

In [44]:
```
from sklearn.ensemble import RandomForestClassifier

my_rf_classifier = RandomForestClassifier()
```

In [45]:
```
my_rf_classifier.fit(X_train,y_train)
```

```
<ipython-input-45-c44482f685a4>:1: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples,), for example using ravel
().
  my_rf_classifier.fit(X_train,y_train)
```

Out[45]:
```
RandomForestClassifier()
```

In [46]:
```
my_rf_preds = my_rf_classifier.predict(X_test)
```

In [47]:
```
from sklearn.decomposition import PCA
```

In [48]:
```
pca = PCA(n_components=.9)
```

In [49]:
```
pca.fit(X_train)
```

Out[49]:
```
PCA(n_components=0.9)
```

In [50]:
```
pca.explained_variance_ratio_
```

Out[50]:
```
array([0.40649473, 0.22023184, 0.13127402, 0.1060638 , 0.08266735])
```

In [51]:
```
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [52]:
```
X_train_pca.shape, X_test_pca.shape
```

Out[52]:
```
((3367, 5), (842, 5))
```

In [53]:
```
X_train_pca
```

Out[53]:
```
array([[-21.21943008,  -9.21814904,  -9.30480999,  -3.68538107,
          4.70324379],
       [-12.22347412,   1.71574428,   3.98774322,  11.14641567,
         -0.63722698],
       [ 25.17616276,  -6.56538012,  -2.91848216,   4.87629802,
         -7.39070466],
       ...,
       [ -4.37292657,   0.57541661,  -5.38494447,  12.49000034,
         -8.09982517],
       [-19.2541749 ,   2.64066502,  -2.32137973,  -5.62146032,
          2.17367005],
       [ -0.64241582,  17.75598707,  -0.27717285,  10.28449227,
         -2.06703049]])
```

In [54]:
```
# using the same train_df data target to predict in the test_df data
# drop the columns which we have drop in the train_df data
```

In [55]:
```
features_test = test_df.drop(['ID','X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X2
90', 'X293', 'X297', 'X330', 'X347'],axis=1)
target_test = train_df[['y']]
```

In [56]:
```
features_test.dtypes
```

Out[56]:
```
X0      int32
X1      int32
X2      int32
X3      int32
X4      int32
        ...
X380    int64
X382    int64
X383    int64
X384    int64
X385    int64
Length: 364, dtype: object
```

In [57]:
```
from sklearn.model_selection import train_test_split
```

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(features_test,target_test, train_size=.8
)
```

In [59]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[59]:
```
((3367, 364), (842, 364), (3367, 1), (842, 1))
```

In [60]:
```
from xgboost import XGBClassifier, XGBRFClassifier
```

In [61]:
```
my_xgb_clf = XGBRFClassifier(booster = 'gbtree')
```

In [62]:
```
my_xgb_clf.fit(X_train, y_train)
```

```
C:\Users\jsanb\anaconda4\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of l
abel encoder in XGBClassifier is deprecated and will be removed in a future release. To remov
e this warning, do the following: 1) Pass option use_label_encoder=False when constructing XG
BClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\jsanb\anaconda4\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarn
ing: A column-vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[13:15:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.
cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'mu
lti:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd li
ke to restore the old behavior.
```

Out[62]:
```
XGBRFClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bytree=1, enable_categorical=False, gamma=0,
                gpu_id=-1, importance_type=None, interaction_constraints='',
                max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
                monotone_constraints='()', n_estimators=100, n_jobs=4,
                num_parallel_tree=100, objective='multi:softprob',
                predictor='auto', random_state=0, reg_alpha=0,
                scale_pos_weight=None, tree_method='exact',
                validate_parameters=1, verbosity=None)
```

In [63]:
```
my_xgb_preds = my_xgb_clf.predict(X_test)
```

In [ ]: