In [5]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:

```
train_df=pd.read_csv("train.csv",encoding="utf-8")
test_df=pd.read_csv("test.csv",encoding='utf-8')
```

In [7]:

```
pd.set_option('display.max_columns',500)
```

In [8]:

```
train_df.head()
```

Out[8]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	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	0	1
3	9	80.62	az	t	n	f	d	x	l	e	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

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

In [9]:

```
train_df.columns
```

Out[9]:

```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X38
3', 'X384',
       'X385'],
      dtype='object', length=378)
```

In [10]:

```
train_df.dtypes
```

Out[10]:

```
ID          int64
y         float64
X0         object
X1         object
X2         object
           ...
X380        int64
X382        int64
X383        int64
X384        int64
X385        int64
Length: 378, dtype: object
```

Observe the Data

we can see X0,X1...X8 are categorical

X10,X11...X385 are numerical - Presence/Absence of a Feature

The objective is to Predict y(carTestingTime)

In [11]:

```
pd.set_option('display.max_rows',500)
train_df.var()
```

Out[11]:

```
ID     5.941936e+06
y      1.607667e+02
X10    1.313092e-02
X11    0.000000e+00
X12    6.945713e-02
X13    5.462335e-02
X14    2.448929e-01
X15    4.750593e-04
X16    2.607237e-03
X17    7.546747e-03
X18    7.780720e-03
X19    8.965997e-02
X20    1.224296e-01
X21    2.607237e-03
X22    7.941395e-02
X23    2.024755e-02
X24    1.897527e-03
X26    4.965595e-03
X27    2.167142e-01
X28    3.149733e-02
X29    4.116360e-02
X30    4.494827e-03
X31    1.784108e-01
X32    1.104448e-02
X33    2.375861e-04
X34    5.435912e-03
X35    1.784108e-01
X36    4.494827e-03
X37    1.784108e-01
X38    3.216333e-02
X39    2.375861e-04
X40    7.124196e-04
X41    1.127676e-02
X42    2.375861e-04
X43    6.702548e-02
X44    1.127676e-02
X45    1.891677e-01
X46    2.405915e-01
X47    1.266806e-02
X48    2.183952e-02
X49    1.072316e-01
X50    1.682812e-01
X51    2.008584e-01
X52    4.051148e-02
X53    6.844152e-03
X54    4.159778e-02
X55    5.200810e-03
X56    2.070297e-02
X57    1.313092e-02
X58    2.444393e-01
X59    7.124196e-04
X60    1.423823e-03
X61    4.397771e-02
X62    5.905777e-03
X63    1.127676e-02
X64    2.344678e-01
X65    2.134210e-03
X66    2.635749e-02
X67    1.897527e-03
```

```
X68     6.804065e-02
X69     2.904660e-02
X70     7.367338e-02
X71     9.287924e-02
X73     1.956359e-02
X74     7.124196e-04
X75     3.481721e-02
X76     4.159778e-02
X77     1.243646e-02
X78     5.670901e-03
X79     2.455572e-02
X80     5.018657e-02
X81     1.766189e-01
X82     1.681757e-02
X83     1.186801e-03
X84     9.287924e-02
X85     2.416252e-01
X86     1.423823e-03
X87     9.496670e-04
X88     7.078463e-03
X89     7.124196e-04
X90     7.312662e-03
X91     1.660732e-03
X92     9.496670e-04
X93     0.000000e+00
X94     7.312662e-03
X95     2.375861e-04
X96     1.834087e-01
X97     4.259273e-03
X98     5.420295e-02
X99     8.481960e-03
X100    2.138795e-01
X101    6.025462e-02
X102    6.844152e-03
X103    1.690946e-01
X104    1.897527e-03
X105    2.370780e-03
X106    1.289955e-02
X107    0.000000e+00
X108    1.451681e-02
X109    3.876753e-02
X110    9.496670e-04
X111    2.455572e-02
X112    2.843581e-03
X113    2.183952e-02
X114    1.247954e-01
X115    2.040719e-01
X116    1.580596e-01
X117    4.677274e-02
X118    2.351137e-01
X119    2.351137e-01
X120    4.051148e-02
X122    7.078463e-03
X123    2.607237e-03
X124    4.750593e-04
X125    3.079812e-03
X126    3.745482e-02
X127    2.500357e-01
X128    3.985835e-02
X129    1.075906e-01
X130    3.985835e-02
```

```
X131    2.590773e-02
X132    2.145094e-01
X133    1.088435e-01
X134    2.183952e-02
X135    2.635749e-02
X136    4.159778e-02
X137    2.433587e-01
X138    3.920419e-02
X139    8.234595e-02
X140    3.876753e-02
X141    1.405530e-02
X142    1.770047e-01
X143    3.679694e-02
X144    1.551541e-01
X145    1.423823e-03
X146    3.920419e-02
X147    2.183952e-02
X148    4.289762e-02
X150    1.645707e-01
X151    7.823404e-02
X152    3.127510e-02
X153    7.124196e-04
X154    1.652641e-01
X155    7.066685e-02
X156    2.028408e-01
X157    2.028408e-01
X158    1.770047e-01
X159    1.336219e-02
X160    1.186801e-03
X161    1.586353e-01
X162    3.920419e-02
X163    2.113977e-01
X164    5.859468e-02
X165    4.494827e-03
X166    3.216333e-02
X167    9.496670e-04
X168    1.975364e-01
X169    6.609727e-03
X170    2.365213e-02
X171    2.252785e-01
X172    5.905777e-03
X173    9.648436e-03
X174    1.704703e-02
X175    2.183952e-02
X176    1.681757e-02
X177    4.762890e-02
X178    2.467665e-01
X179    4.569998e-02
X180    1.330640e-01
X181    8.505956e-02
X182    9.494490e-02
X183    4.023607e-03
X184    1.423823e-03
X185    1.842139e-02
X186    2.487635e-01
X187    2.437420e-01
X189    7.744517e-02
X190    2.375861e-04
X191    2.492121e-01
X192    2.370780e-03
X194    2.487635e-01
```

```
   X195    1.150892e-02
   X196    1.011424e-02
   X197    3.127510e-02
   X198    2.252009e-02
   X199    2.843581e-03
   X200    6.609727e-03
   X201    1.461667e-01
   X202    1.831631e-01
   X203    1.658801e-02
   X204    2.375861e-04
   X205    2.375861e-04
   X206    1.887861e-02
   X207    2.375861e-04
   X208    5.901034e-02
   X209    9.117898e-02
   X210    2.375861e-04
   X211    1.474739e-02
   X212    5.435912e-03
   X213    1.897527e-03
   X214    6.844152e-03
   X215    8.889775e-02
   X216    5.905777e-03
   X217    7.312662e-03
   X218    2.148670e-01
   X219    6.293661e-02
   X220    2.463157e-01
   X221    8.014579e-03
   X222    2.183952e-02
   X223    2.470074e-01
   X224    2.167142e-01
   X225    8.755952e-02
   X226    3.127510e-02
   X227    3.079812e-03
   X228    3.745482e-02
   X229    3.833041e-02
   X230    5.200810e-03
   X231    1.589862e-02
   X232    4.116360e-02
   X233    0.000000e+00
   X234    1.610617e-01
   X235    0.000000e+00
   X236    4.750593e-04
   X237    6.609727e-03
   X238    7.685234e-02
   X239    6.844152e-03
   X240    2.843581e-03
   X241    8.775104e-02
   X242    7.312662e-03
   X243    7.078463e-03
   X244    9.287924e-02
   X245    7.124196e-04
   X246    2.418420e-01
   X247    1.831631e-01
   X248    1.423823e-03
   X249    7.546747e-03
   X250    2.472643e-01
   X251    2.388537e-01
   X252    7.124196e-04
   X253    1.423823e-03
   X254    5.200810e-03
   X255    1.910705e-02
```

```
X256    6.783784e-02
X257    2.375861e-04
X258    2.370780e-03
X259    2.375861e-04
X260    2.375861e-04
X261    2.435900e-01
X262    1.423823e-03
X263    4.116360e-02
X264    3.789284e-02
X265    8.563817e-02
X266    1.423823e-03
X267    8.948889e-03
X268    0.000000e+00
X269    4.750593e-04
X270    2.375861e-04
X271    2.134210e-03
X272    3.613805e-02
X273    2.015935e-01
X274    9.881392e-03
X275    1.986199e-01
X276    3.701635e-02
X277    1.423823e-03
X278    4.750593e-04
X279    4.116360e-02
X280    2.375861e-04
X281    2.607237e-03
X282    4.023607e-03
X283    1.208970e-01
X284    3.942236e-02
X285    1.634555e-01
X286    5.167103e-02
X287    1.566860e-02
X288    2.375861e-04
X289    0.000000e+00
X290    0.000000e+00
X291    1.034697e-02
X292    8.948889e-03
X293    0.000000e+00
X294    1.093787e-01
X295    2.375861e-04
X296    2.375861e-04
X297    0.000000e+00
X298    4.494827e-03
X299    4.494827e-03
X300    1.641534e-01
X301    4.462441e-02
X302    1.127676e-02
X304    7.006250e-02
X305    1.313092e-02
X306    4.181471e-02
X307    2.134210e-03
X308    9.415366e-03
X309    7.078463e-03
X310    2.607237e-03
X311    2.403589e-01
X312    4.259273e-03
X313    2.104576e-01
X314    2.453926e-01
X315    2.792811e-02
X316    1.573375e-01
X317    7.546747e-03
```

```
X318    7.124196e-04
X319    4.750593e-04
X320    7.078463e-03
X321    1.818042e-01
X322    2.138524e-02
X323    9.182184e-03
X324    2.444393e-01
X325    5.670901e-03
X326    3.127510e-02
X327    1.118631e-01
X328    3.854903e-02
X329    2.458669e-01
X330    0.000000e+00
X331    5.293902e-02
X332    7.124196e-04
X333    2.342595e-02
X334    2.486588e-01
X335    3.551935e-03
X336    1.111555e-01
X337    2.497867e-01
X338    6.844152e-03
X339    2.375861e-04
X340    2.183952e-02
X341    8.014579e-03
X342    2.183952e-02
X343    7.227350e-02
X344    8.481960e-03
X345    2.183952e-02
X346    4.527009e-02
X347    0.000000e+00
X348    4.997405e-02
X349    4.289762e-02
X350    2.240671e-01
X351    2.089300e-01
X352    5.124746e-02
X353    2.134210e-03
X354    1.617692e-01
X355    2.357460e-01
X356    1.475408e-01
X357    1.186801e-03
X358    2.447207e-01
X359    3.083030e-02
X360    7.066685e-02
X361    3.282833e-02
X362    2.496467e-01
X363    1.855988e-01
X364    2.843581e-03
X365    2.843581e-03
X366    1.186801e-03
X367    4.912285e-02
X368    5.880257e-02
X369    4.750593e-04
X370    6.609727e-03
X371    1.405530e-02
X372    4.750593e-04
X373    1.887861e-02
X374    1.757146e-01
X375    2.172329e-01
X376    5.399258e-02
X377    2.157528e-01
X378    2.024755e-02
```

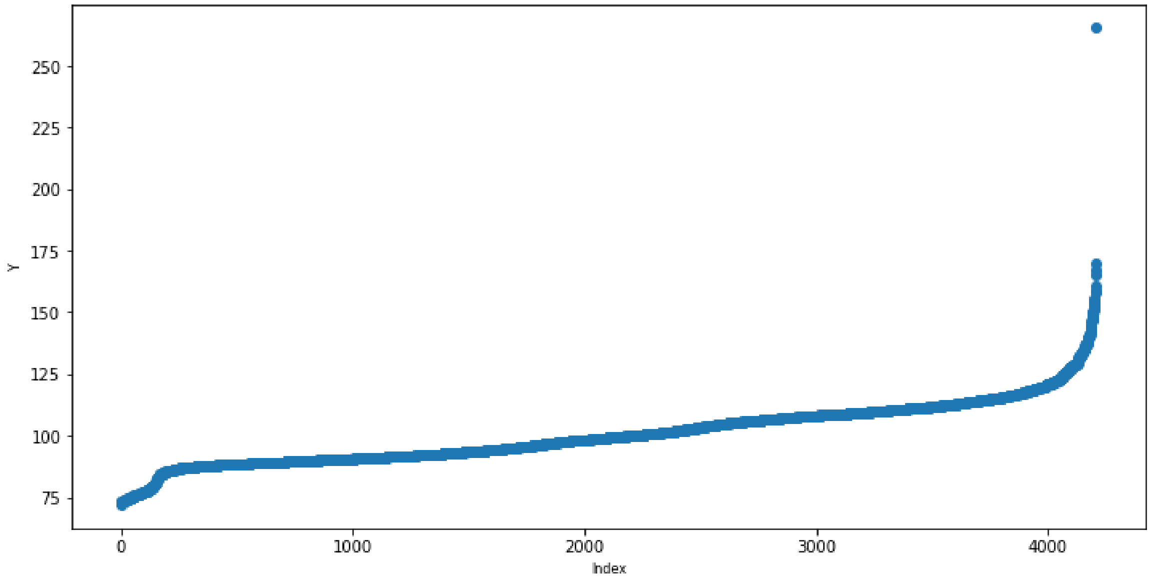
```
X379     9.415366e-03
X380     8.014579e-03
X382     7.546747e-03
X383     1.660732e-03
X384     4.750593e-04
X385     1.423823e-03
dtype: float64
```

In [12]:

```
plt.figure(figsize=(12,6))
plt.scatter(range(train_df.shape[0]),np.sort(train_df.y.values))

plt.xlabel('Index',fontsize=8)
plt.ylabel('Y',fontsize=8)
```

Out[12]:

```
Text(0, 0.5, 'Y')
```



In [13]:

```
#Check the Statistics for the target column
train_df['y'].describe()
```

Out[13]:

```
count    4209.000000
mean      100.669318
std        12.679381
min        72.110000
25%        90.820000
50%        99.150000
75%       109.010000
max       265.320000
Name: y, dtype: float64
```

In [14]:

```
#Count of Datatypes present in the dataset
dtype_df=train_df.dtypes.reset_index()
dtype_df.columns=['Count','Column Type']
dtype_df.groupby("Column Type").aggregate('count').reset_index()
```

Out[14]:

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

In [15]:

```
numerics=['int16','int32','int64','float16','float32','float64']
objects=['O']
```

In [16]:

```
df_train_num=train_df.select_dtypes(include=numerics)
df_train_object=train_df.select_dtypes(include=objects)
```

In [17]:

```
df_test_num=test_df.select_dtypes(include=numerics)
df_test_object=test_df.select_dtypes(include=objects)
```

In [18]:

```
print('-----------------------------------')
print(df_train_num.columns)
print('-----------------------------------')
print(df_train_object.columns)
```

```
-----------------------------------
Index(['ID', 'y', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16',
'X17',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X38
3', 'X384',
       'X385'],
      dtype='object', length=370)
-----------------------------------
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='objec
t')
```

In [19]:

```
#get all the unique values for every categorical columns in Training Dataset

for col_name in df_train_object.columns:
    print('The unique values in '+col_name+ ' are',df_train_object[col_name].nun
ique())
    print(df_train_object[col_name].unique())
    print('--------------------------------------------------')
```

```
The unique values in X0 are 47
['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'a
k' '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']
--------------------------------------------------
The unique values in X1 are 27
['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']
--------------------------------------------------
The unique values in X2 are 44
['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']
--------------------------------------------------
The unique values in X3 are 7
['a' 'e' 'c' 'f' 'd' 'b' 'g']
--------------------------------------------------
The unique values in X4 are 4
['d' 'b' 'c' 'a']
--------------------------------------------------
The unique values in X5 are 29
['u' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac' 'ad' 'a
e'
 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
--------------------------------------------------
The unique values in X6 are 12
['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']
--------------------------------------------------
The unique values in X8 are 25
['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']
--------------------------------------------------
```

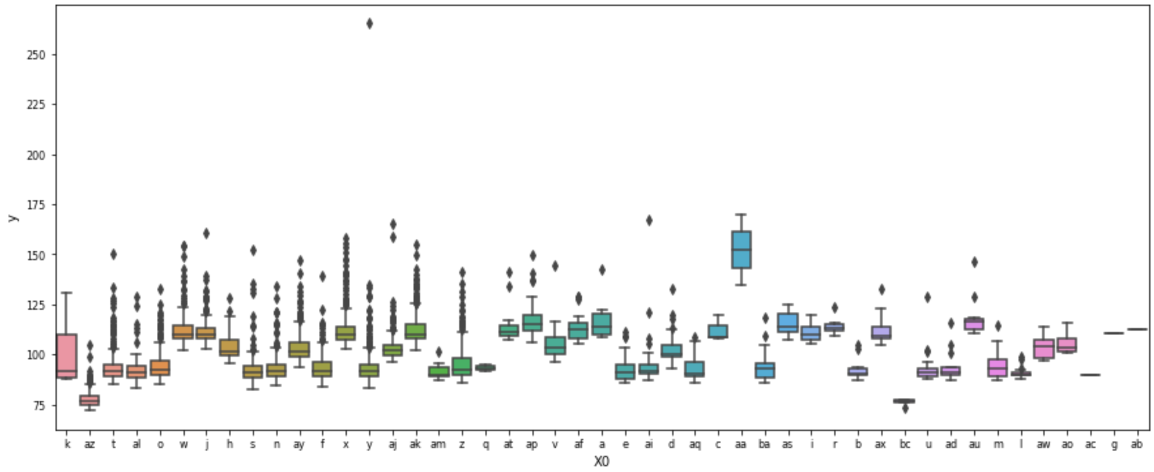
In [20]:

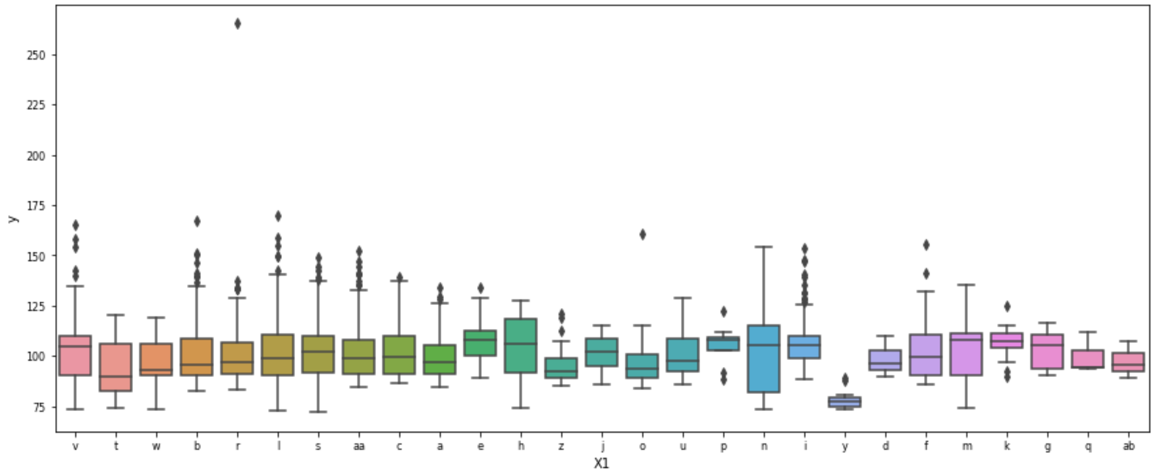
```
#get all the unique values for every categorical columns in Testing Dataset

for col_name in df_test_object.columns:
    print('The unique values in '+col_name+ ' are',df_test_object[col_name].nuni
que())
    print(df_test_object[col_name].unique())
    print('--------------------------------------------------')
```

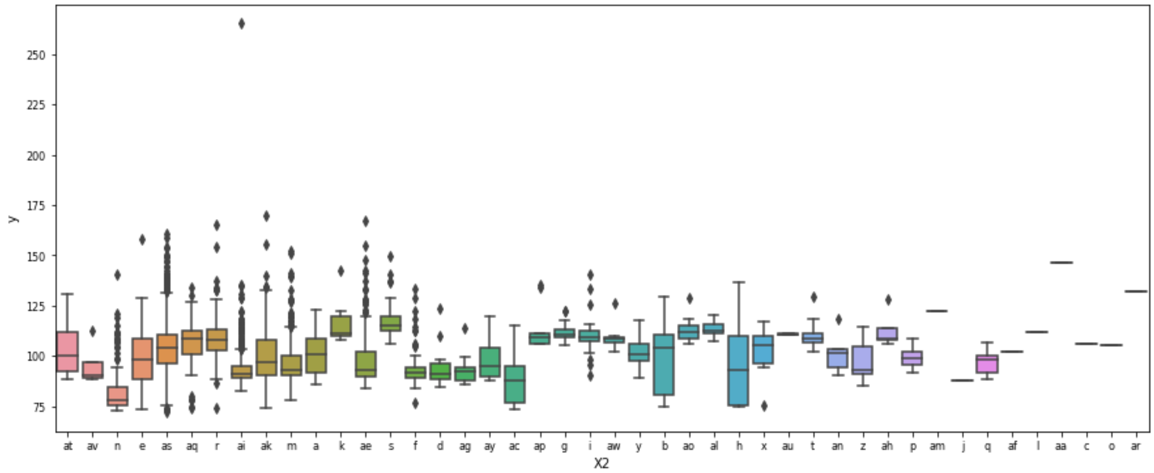
```
The unique values in X0 are 49
['az' 't' 'w' 'y' 'x' 'f' 'ap' 'o' 'ay' 'al' 'h' 'z' 'aj' 'd' 'v' 'a
k'
 'ba' 'n' 'j' 's' 'af' 'ax' 'at' 'aq' 'av' 'm' 'k' 'a' 'e' 'ai' 'i'
'ag'
 'b' 'am' 'aw' 'as' 'r' 'ao' 'u' 'l' 'c' 'ad' 'au' 'bc' 'g' 'an' 'a
e' 'p'
 'bb']
--------------------------------------------------
The unique values in X1 are 27
['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']
--------------------------------------------------
The unique values in X2 are 45
['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']
--------------------------------------------------
The unique values in X3 are 7
['f' 'a' 'c' 'e' 'd' 'g' 'b']
--------------------------------------------------
The unique values in X4 are 4
['d' 'b' 'a' 'c']
--------------------------------------------------
The unique values in X5 are 32
['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']
--------------------------------------------------
The unique values in X6 are 12
['a' 'g' 'j' 'l' 'i' 'd' 'f' 'h' 'c' 'k' 'e' 'b']
--------------------------------------------------
The unique values in X8 are 25
['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']
--------------------------------------------------
```

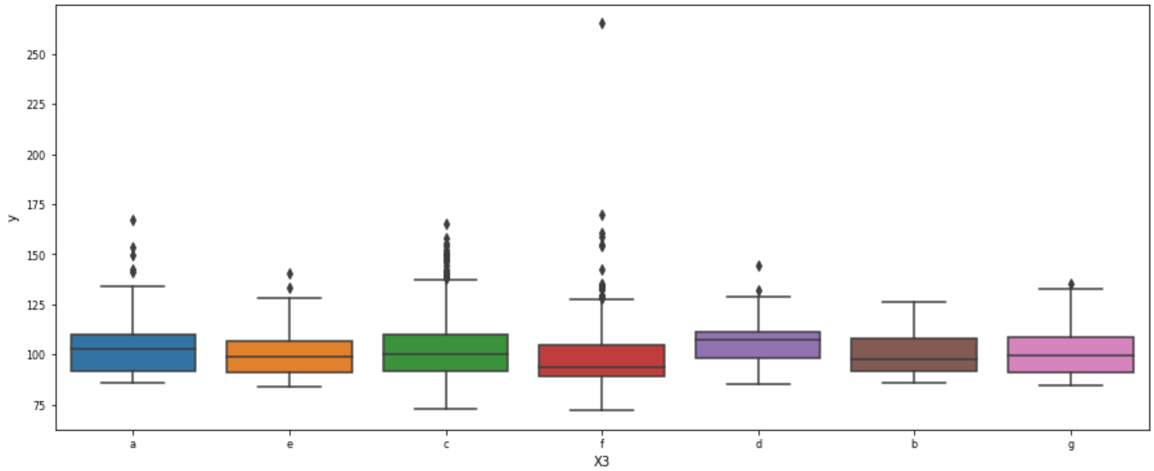
In [21]:

```
#Analyse the testing time for each categorical columns
cols=['X0','X1','X2','X3','X4','X5','X6','X8']

for col in cols:
    plt.figure(figsize=(15,6))

    sns.boxplot(x=col,y='y',data=train_df)

    plt.xlabel(col,fontsize=10)
    plt.ylabel('y',fontsize=10)
    plt.xticks(fontsize=8)
    plt.yticks(fontsize=8)
```

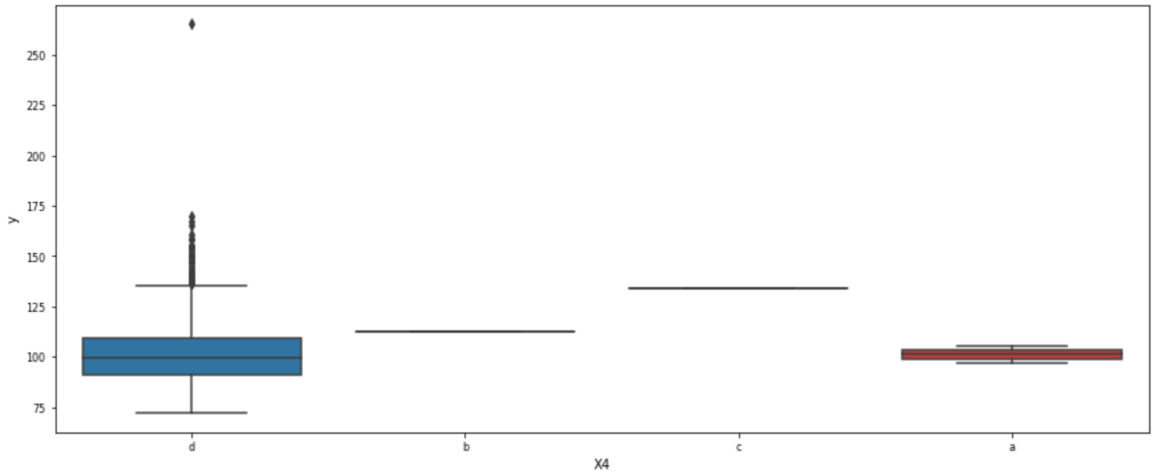

250
225
200
175
150
125
100
75
y
k az t al o w j h s n ay f x y aj ak am z q at ap v af a e ai d aq c aa ba as i r b ax bc u ad au m l aw ao ac g ab
X0


250
225
200
175
150
125
100
75
y
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
X1

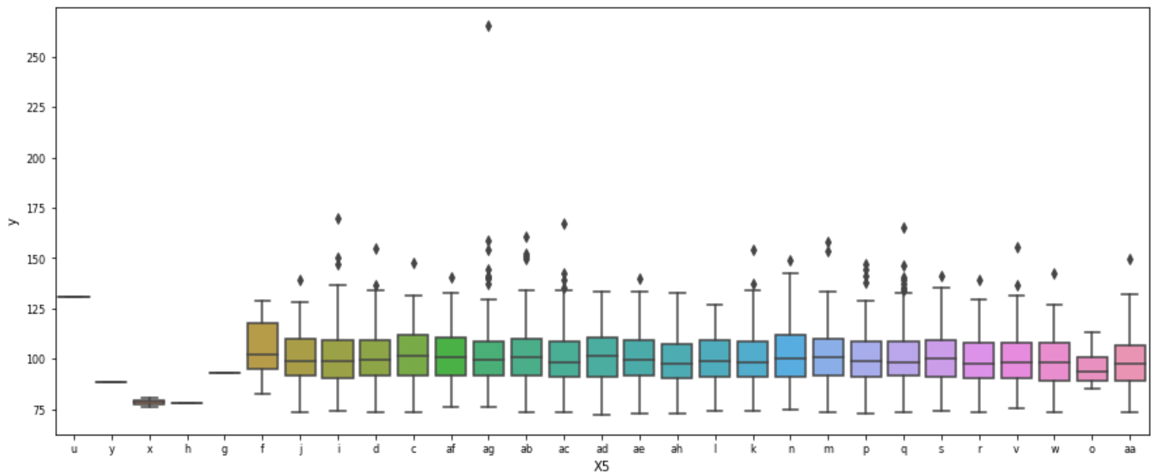

250
225
200
175
150
125
100
75
y
at av n e as aq r ai ak m a k ae s f d ag ay ac ap g i aw y b ao al h x au t an z ah p am j q af l aa c o ar
X2



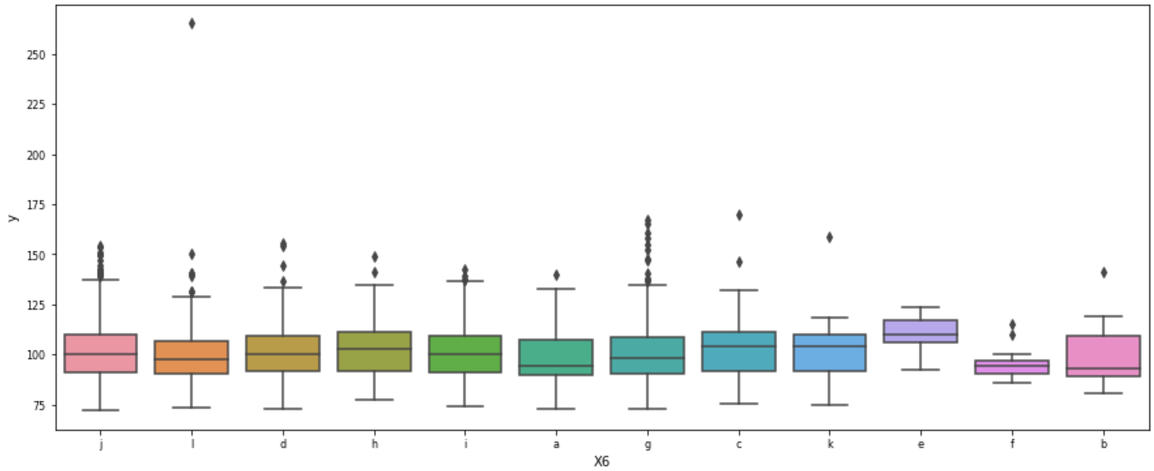
250
225
200
175
150
125
100
75
y
a
e
c
f
d
b
g
X3

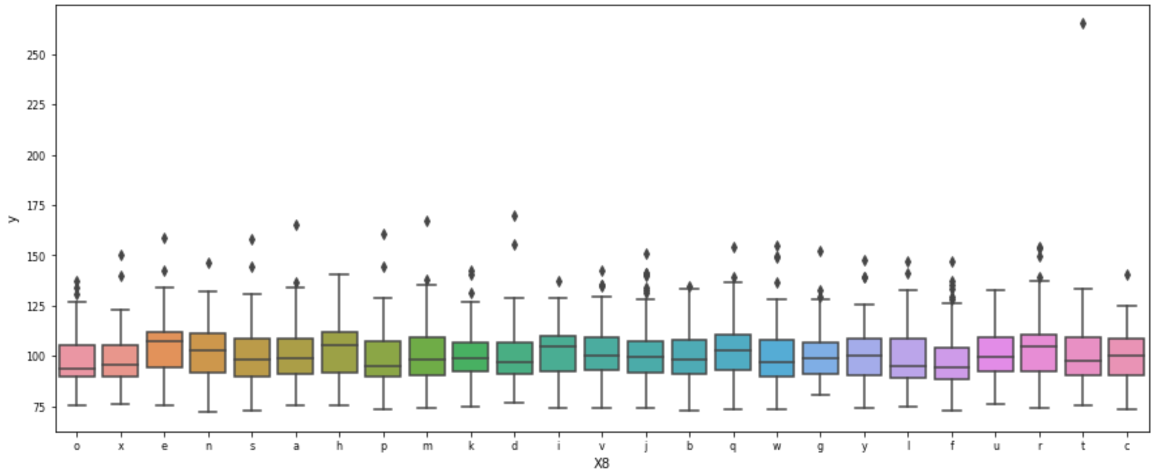


250
225
200
175
150
125
100
75
y
d
b
c
a
X4



250
225
200
175
150
125
100
75
y
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
X5

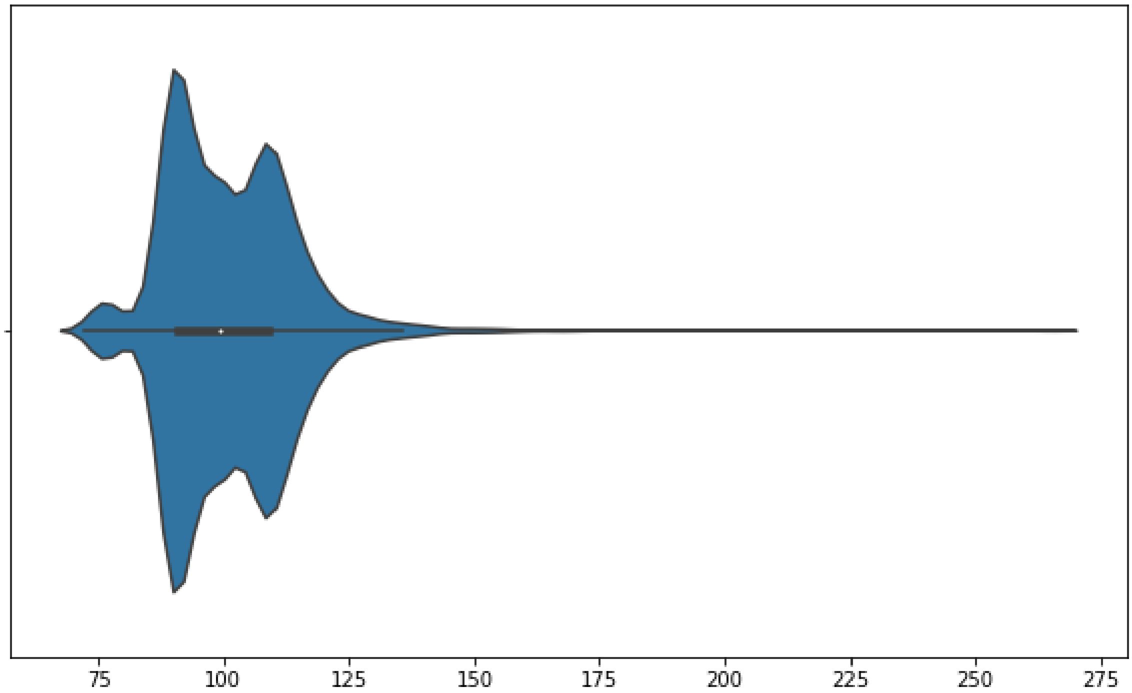




Outlier Detection/Removal

In [22]:

```
plt.figure(figsize=(10,6))
sns.violinplot(train_df['y'].values);
```



In [23]:

```
#Calculate Q1 (25th %ile)
Q1=np.percentile(train_df.loc[:,'y'],25)

#calculate Q3 (75th%ile)
Q3=np.percentile(train_df.loc[:,'y'],75)

#calculate Q2(50th%ile)
Q2=np.percentile(train_df.loc[:,'y'],50)

#Use the interqurtiles to calculate an outler step
step=(Q3-Q1)*3


print ('The 25%ile is {},50%ile is {},75%ile is {} and step is {}'.format(Q1,Q2,
Q3,step))
```

```
The 25%ile is 90.82,50%ile is 99.15,75%ile is 109.01 and step is 54.
570000000000036
```

In [24]:

```
outlier_lower_idx=train_df[train_df['y']<=(Q1-step)].index
outlier_upper_idx=train_df[train_df['y']>(Q3+step)].index
```

In [25]:

```
no_lower_outliers=len(outlier_lower_idx)
no_upper_outliers=len(outlier_upper_idx)


print('No of outliers in the lower extreme side is : {}'.format(no_lower_outlier
s))
print('No of outliers in the upper extreme side is : {} '.format(no_upper_outlie
rs))

print('Lower Outlier % {}'.format(no_lower_outliers/train_df.shape[0]*100))
print('Upper Outlier % {}'.format(no_upper_outliers/train_df.shape[0]*100))
```

```
No of outliers in the lower extreme side is : 0
No of outliers in the upper extreme side is : 4
Lower Outlier % 0.0
Upper Outlier % 0.09503444998812069
```

In [26]:

```
#Observe the severity of the outler sample
train_df.iloc[outlier_upper_idx]['y'].sort_values(ascending=False)
```

Out[26]:

```
883     265.32
342     169.91
1459    167.45
3133    165.52
Name: y, dtype: float64
```

In [27]:

```
#only one outlier
train_df[train_df.y>=170]
```

Out[27]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X
883	1770	265.32	y	r	ai	f	d	ag	l	t	0	0	0	0	0	0	0	

In [28]:

```
train_df['y'].describe()
```

Out[28]:

```
count    4209.000000
mean      100.669318
std        12.679381
min        72.110000
25%        90.820000
50%        99.150000
75%       109.010000
max       265.320000
Name: y, dtype: float64
```

In [29]:

```
print('Count of Duplicates={}'.format(len(train_df[train_df.drop(['ID','y'],axis
=1).duplicated()])))
print('% of duplicates ={}'.format(len(train_df[train_df.drop(['ID','y'],axis=1)
.duplicated()])/train_df.shape[0]))
```

```
Count of Duplicates=298
% of duplicates =0.07080066524114992
```

In [30]:

```
def av_dups(x):
    Y.loc[list(x.index)]=Y.loc[list(x.index)].mean()
```

In [31]:

```
X=train_df.drop(['y'],axis=1)
Y=train_df['y']
dups=X[X.duplicated(keep=False)]
dups.groupby(dups.columns.tolist()).apply(av_dups)
train_df.drop(X[X.duplicated()].index.values,axis=0,inplace=True)
```

In [32]:

```
X=train_df.drop(['y'],axis=1)
Y=train_df['y']
X.reset_index(inplace=True,drop=True)
Y.reset_index(inplace=True,drop=True)
```

In [33]:

```
X.shape,Y.shape
```

Out[33]:

```
((4209, 377), (4209,))
```

Categorical to Numerical

In [34]:

```
df1=pd.DataFrame([[1,2],[3,4]],columns=list('AB'))
df2=pd.DataFrame([[5,6],[7,8]],columns=list('AB'))
data=df1.append(df2,ignore_index=True)
data
```

Out[34]:

	A	B
0	1	2
1	3	4
2	5	6
3	7	8

In [35]:

```
data=train_df.append(test_df,ignore_index=True)
data=pd.get_dummies(data)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/pandas/core/fram
e.py:7138: FutureWarning: Sorting because non-concatenation axis is
not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=T
rue'.

  sort=sort,
```

In [36]:

```
data.index
```

Out[36]:

```
RangeIndex(start=0, stop=8418, step=1)
```

In [37]:

```
train,test=data[0:len(train_df)],data[len(train_df):]
```

In [38]:

```
train.shape,test.shape
```

Out[38]:

```
((4209, 581), (4209, 581))
```

In [39]:

```
#separate Fetaures and Target columns
X_train1=train.drop(['y','ID'],axis=1)
Y_train1=train.y
X_test1=test.drop(['y','ID'],axis=1)
```

In [40]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,r2_score
```

In [41]:

```
#instantiate the RF Regressor
rf_reg=RandomForestRegressor()
```

In [42]:

```
X_train,X_test,y_train,y_test=train_test_split(X_train1,X_test1,test_size=0.25,r
andom_state=1)
```

In [43]:

```
%%time
rf_reg.fit(X_train,y_train)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/ensembl
e/forest.py:245: FutureWarning: The default value of n_estimators wi
ll change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

CPU times: user 10.8 s, sys: 111 ms, total: 10.9 s
Wall time: 10.8 s
```

Out[43]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=Non
e,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=
None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=Non
e,
                      verbose=0, warm_start=False)
```

In [44]:

```
%%time

#predict Training Samples
y_pred=rf_reg.predict(X_train)



print('\n Training Score :')
print('mean Squared Error : %.2f'% mean_squared_error(y_train,y_pred))
print ('R2 Score : %.2f'% r2_score(y_train,y_pred))

y_pred=rf_reg.predict(X_test)

print('\n Training Score :')
print('mean Squared Error : %.2f'% mean_squared_error(y_test,y_pred))
print ('R2 Score : %.2f'% r2_score(y_test,y_pred))
```

```
 Training Score :
mean Squared Error : 0.01
R2 Score : 0.77

 Training Score :
mean Squared Error : 0.06
R2 Score : -0.14
CPU times: user 433 ms, sys: 66.9 ms, total: 499 ms
Wall time: 331 ms
```

In [45]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

Not Able to Run Xtreme Gradient Boosting as there is some issue with my MAC Laptop, SO i am just Writing the code for the same.

In [46]:

```
#pip install xgboost
import xgboost as xgb
from sklearn.metrics import mean_absolute_error
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent cal
l last)
<ipython-input-46-de2b690ee1b3> in <module>
      1 #pip install xgboost
----> 2 import xgboost as xgb
      3 from sklearn.metrics import mean_absolute_error

ModuleNotFoundError: No module named 'xgboost'
```

In [ ]:

```
model=xgb.XGBRegressor()
model.fit(X_train,y_train)
model.score(X_test,y_test)
model.score(X_train,y_train)
```

In [ ]:

```
#pandas Profiling
import pandas_profiling as pp
```

In [ ]:

```
pip install pandas_profiling
```

In [ ]:

```
report=pp.ProfileReport(train_df)
#report.to_file('profile_report.html')
```

In [ ]:

```
report.to_file('profile_report.html')
```

In [ ]:

In [ ]:

In [ ]: