In [45]:

```
#DESCRIPTION

#Reduce the time a Mercedes-Benz spends on the test bench.

#Problem Statement Scenario:
#Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

#To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

#You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

#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.
```

```python
#Predict your test_df values using XGBoost.



# Import the required libraries
# linear algebra
import numpy as np
import pandas as pd
# for dimensionality reduction
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
#conda install -c conda-forge xgboost

!pip install xgboost
color = sns.color_palette()



# Step2: Read the data from train.csv

df_train = pd.read_csv('train.csv')
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
```

```
Requirement already satisfied: xgboost in /Volumes/Samsung_T5/Anaconda/
anaconda3/lib/python3.7/site-packages (1.2.0)
Requirement already satisfied: numpy in /Volumes/Samsung_T5/Anaconda/an
aconda3/lib/python3.7/site-packages (from xgboost) (1.18.1)
Requirement already satisfied: scipy in /Volumes/Samsung_T5/Anaconda/an
aconda3/lib/python3.7/site-packages (from xgboost) (1.4.1)
Size of training set: 4209 rows and 378 columns
```

In [46]:

```python
# print few rows
df_train.head()
df_train.shape
```

Out[46]:

```
(4209, 378)
```

In [47]:
```
# Understand the data types we have

# iterate through all the columns which has X in the name of the column
cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))

print('Feature types:')
df_train[cols].dtypes.value_counts()
```

```
Number of features: 376
Feature types:
```

Out[47]:
```
int64     368
object      8
dtype: int64
```

In [48]:
```
# Count the data in each of the columns

counts = [[], [], []]
for c in cols:
    typ = df_train[c].dtype
    uniq = len(np.unique(df_train[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X28
9', 'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

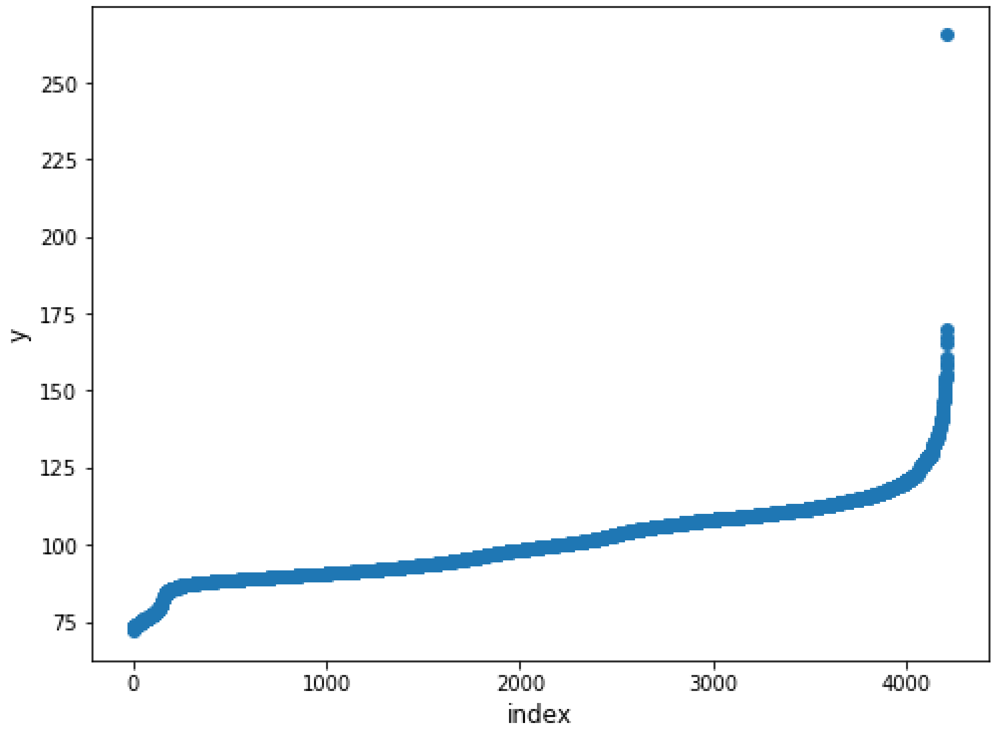
In [5]:
```
# Collect the Y values into an array

# seperate the y from the data as we will use this to learn as
# the prediction output
y_train = df_train['y'].values
```

In [6]:
```
#removing constant columns from data set
df_train = df_train.drop(['X11', 'X93', 'X107', 'X233', 'X235', 'X268',
 'X289', 'X290', 'X293', 'X297', 'X330', 'X347'], axis=1)
```

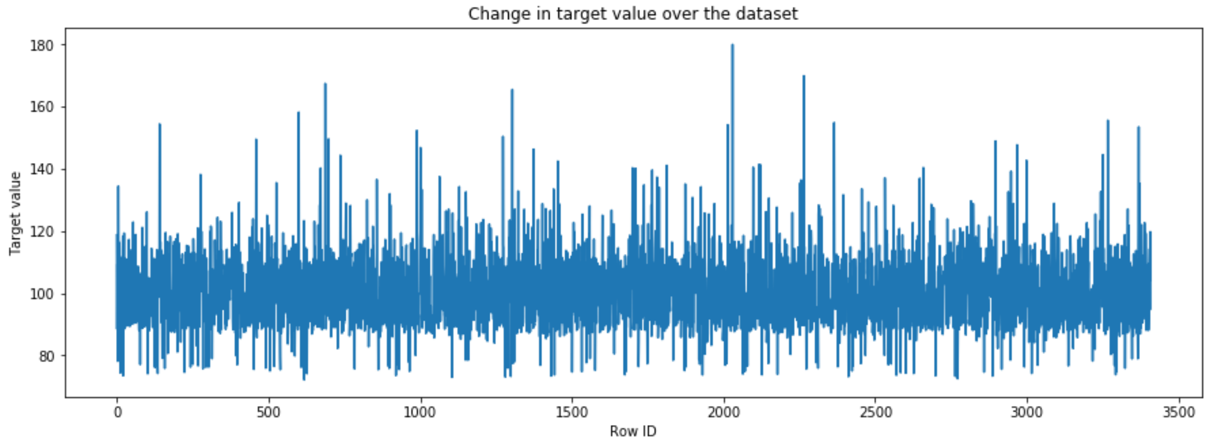
In [8]:
```
#checking if the constnat column is removed or not
print('X11' in df_train)
```

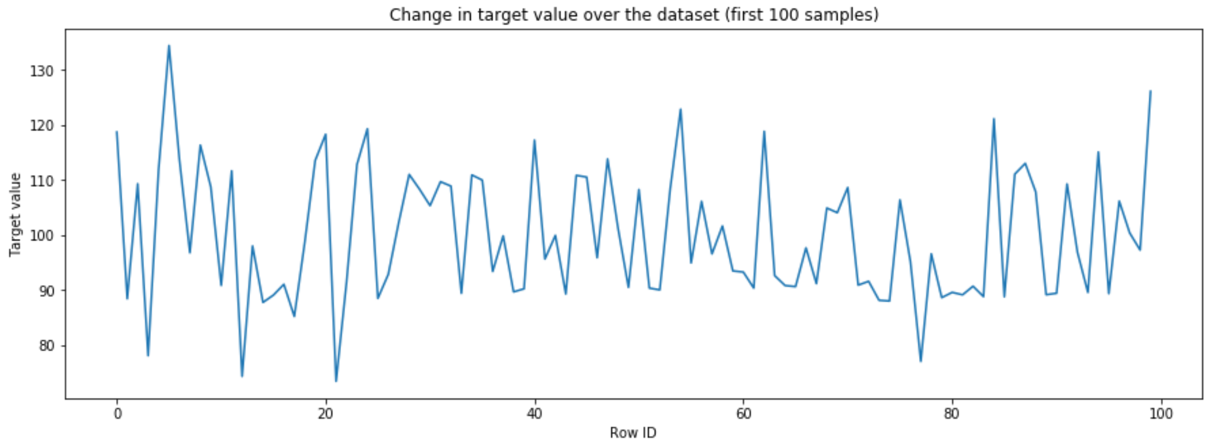
```
False
```

In [49]:
```
# checking the distribution with the target
plt.figure(figsize=(8,6))
plt.scatter(range(df_train.shape[0]), np.sort(df_train.y.values))
plt.xlabel('index', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.show()
```



In [78]:
```
#Distribution of target variable
plt.figure(figsize=(15, 5))
plt.plot(y_train)
plt.xlabel('Row ID')
plt.ylabel('Target value')
plt.title('Change in target value over the dataset')
plt.show()

plt.figure(figsize=(15, 5))
plt.plot(y_train[:100])
plt.xlabel('Row ID')
plt.ylabel('Target value')
plt.title('Change in target value over the dataset (first 100 samples)'
)
print()
```

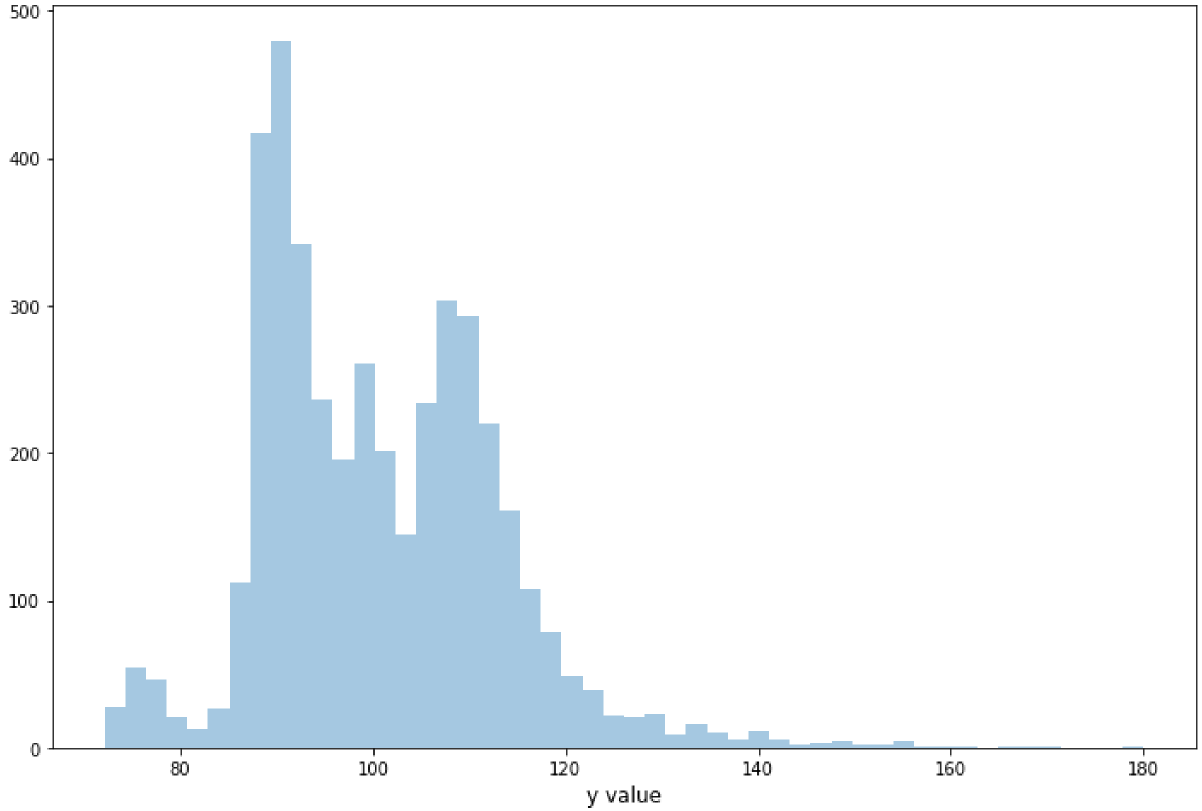




In [50]:
```
ulimit = 180
df_train['y'].loc[df_train['y']>ulimit] = ulimit

plt.figure(figsize=(12,8))
sns.distplot(df_train.y.values, bins=50, kde=False)
plt.xlabel('y value', fontsize=12)
plt.show()
```

```
/Volumes/Samsung_T5/Anaconda/anaconda3/lib/python3.7/site-packages/pand
```

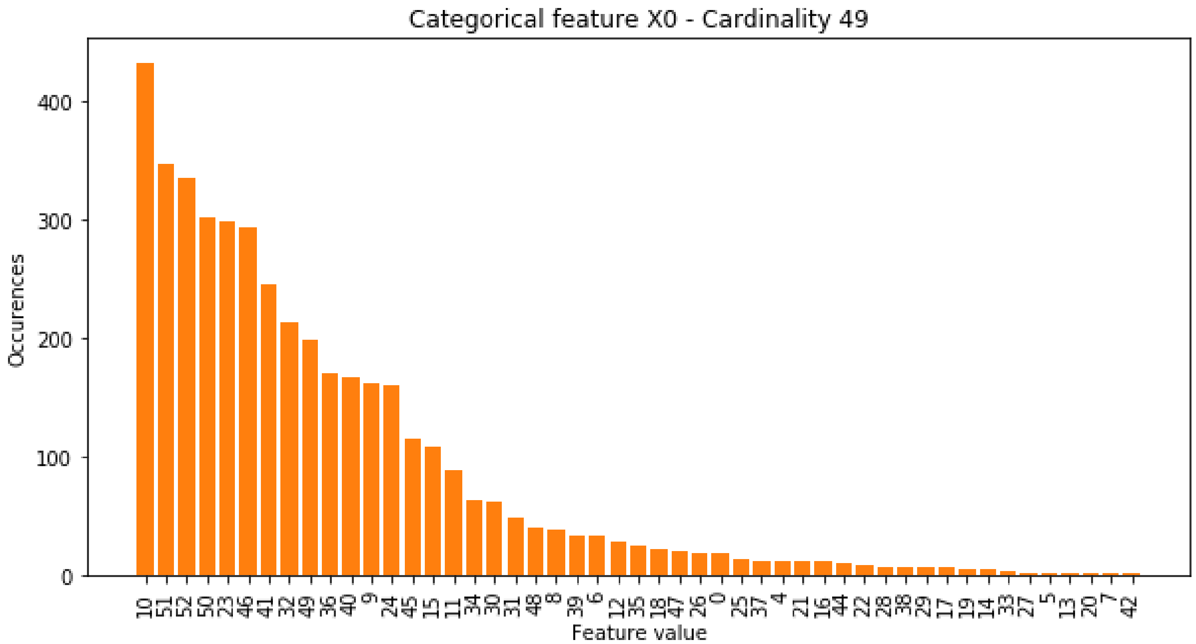
```
as/core/indexing.py:670: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

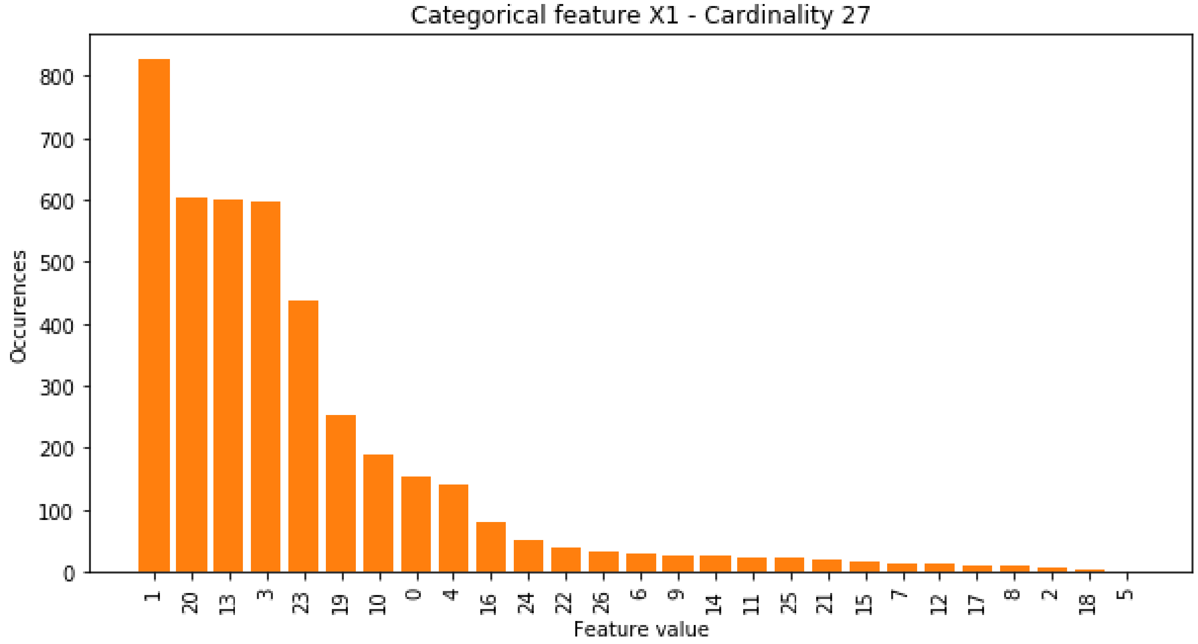


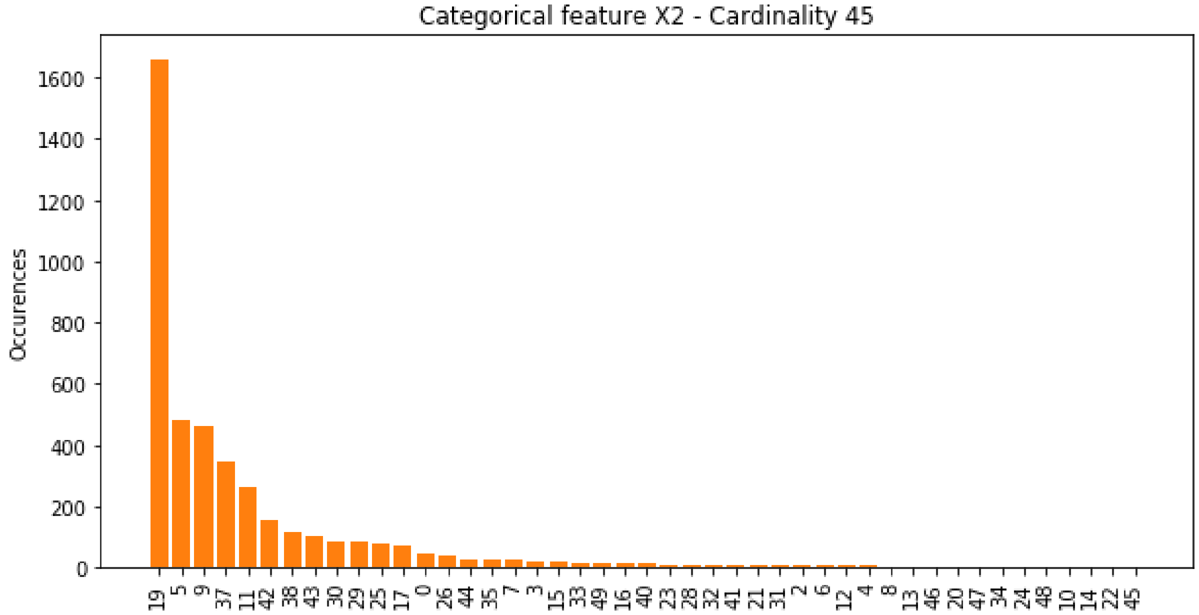
In [80]:

```
import seaborn as sns
pal = sns.color_palette()
for c in counts[2]:
    value_counts = df_train[c].value_counts()
    fig, ax = plt.subplots(figsize=(10, 5))
    plt.title('Categorical feature {} - Cardinality {}'.format(c, len(np.unique(df_train[c]))))
```

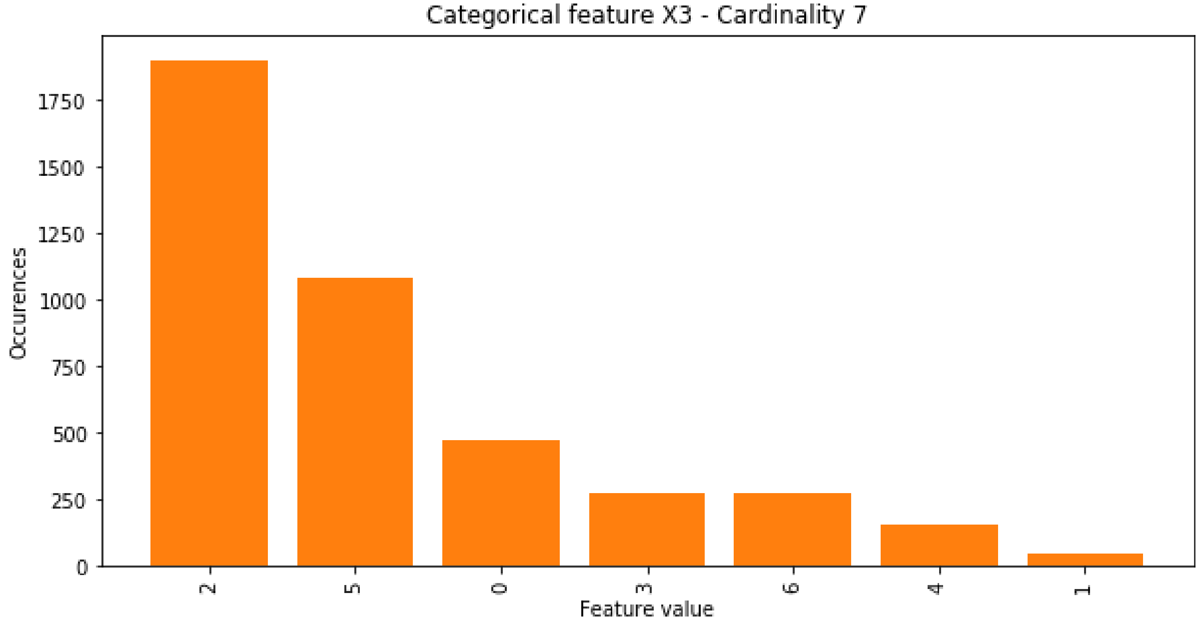
```
    plt.xlabel('Feature value')
    plt.ylabel('Occurences')
    plt.bar(range(len(value_counts)), value_counts.values, color=pal[1
])
    ax.set_xticks(range(len(value_counts)))
    ax.set_xticklabels(value_counts.index, rotation='vertical')
    plt.show()
```

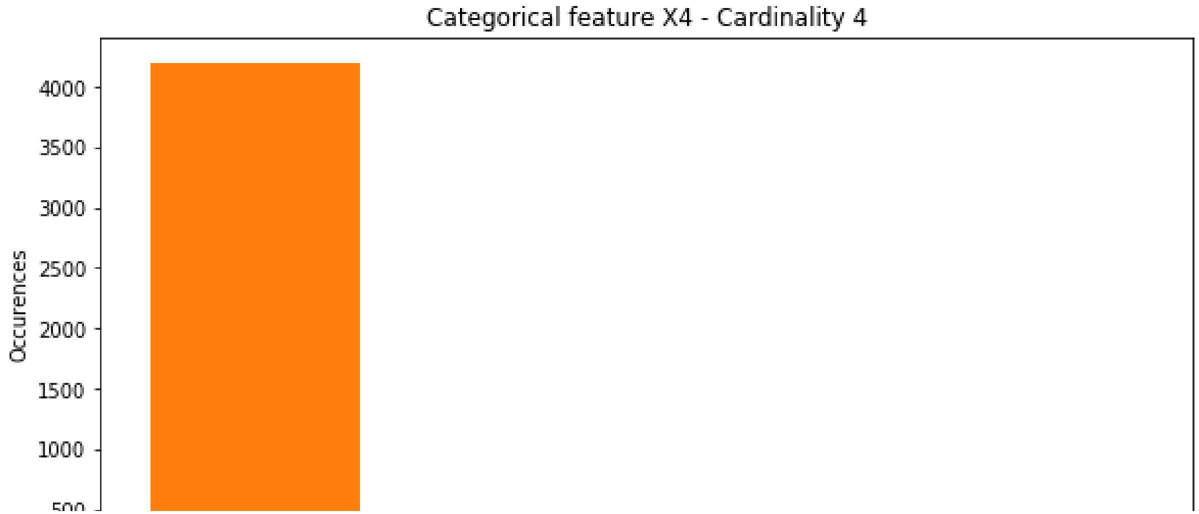


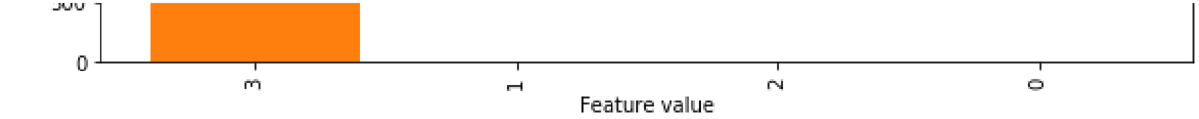

Categorical feature X1 - Cardinality 27
Occurences
0
100
200
300
400
500
600
700
800
1 20 13 3 23 19 10 0 4 16 24 22 26 6 9 14 11 25 21 15 7 12 17 8 2 18 5
Feature value

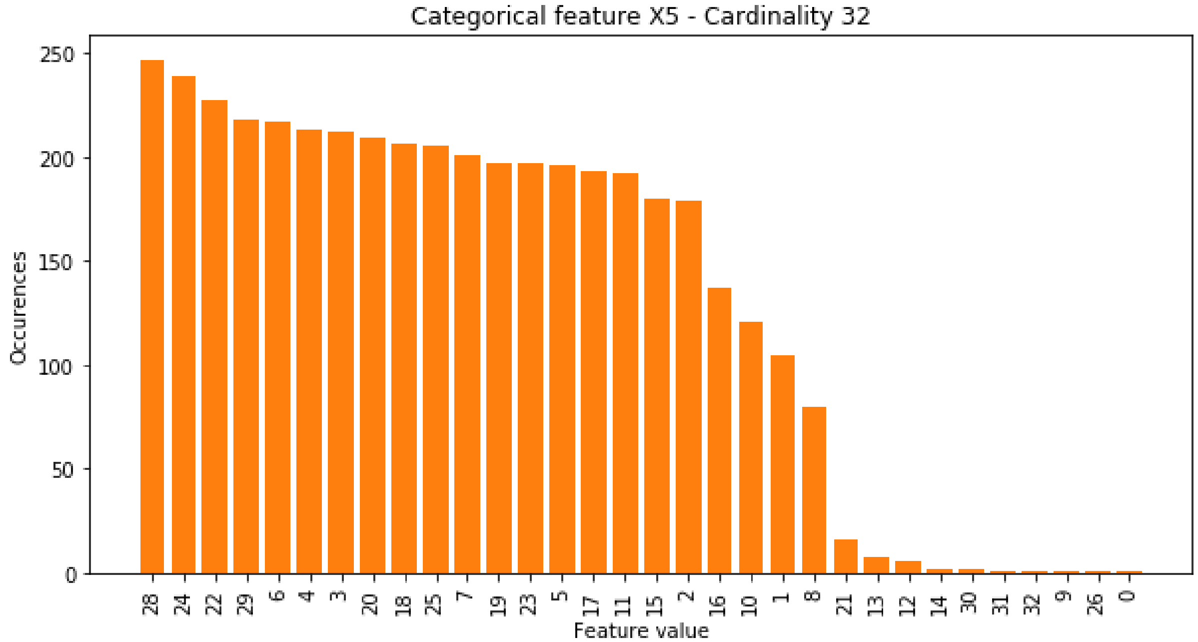

Categorical feature X2 - Cardinality 45
Occurences
0
200
400
600
800
1000
1200
1400
1600
19 5 9 37 11 42 38 43 30 29 25 17 0 26 44 35 7 3 15 33 49 16 40 23 28 32 41 21 31 2 6 12 4 8 13 46 20 47 34 24 48 10 14 22 45

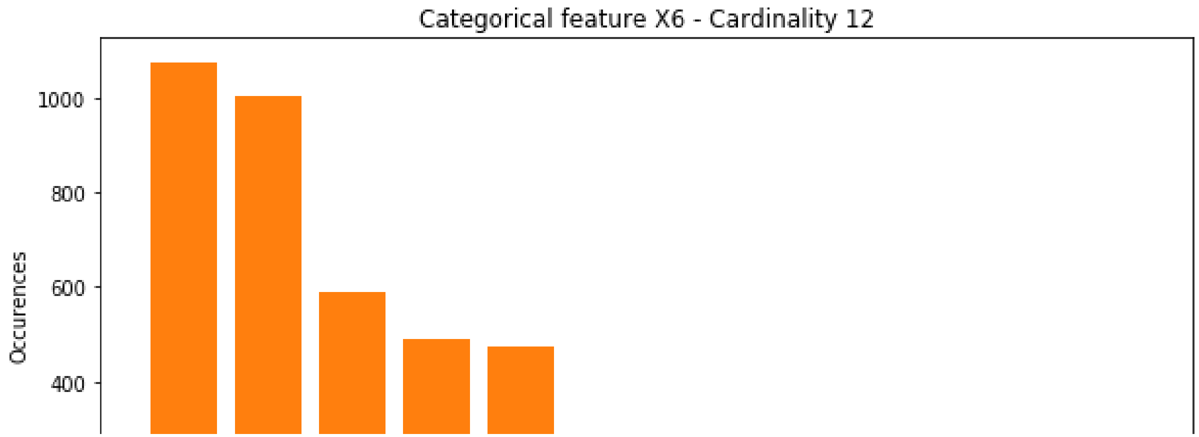
Feature value

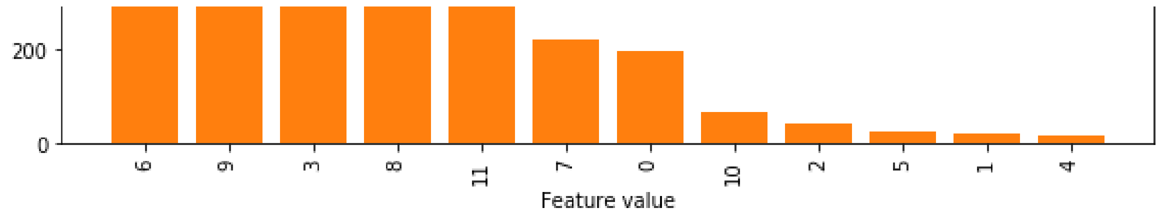


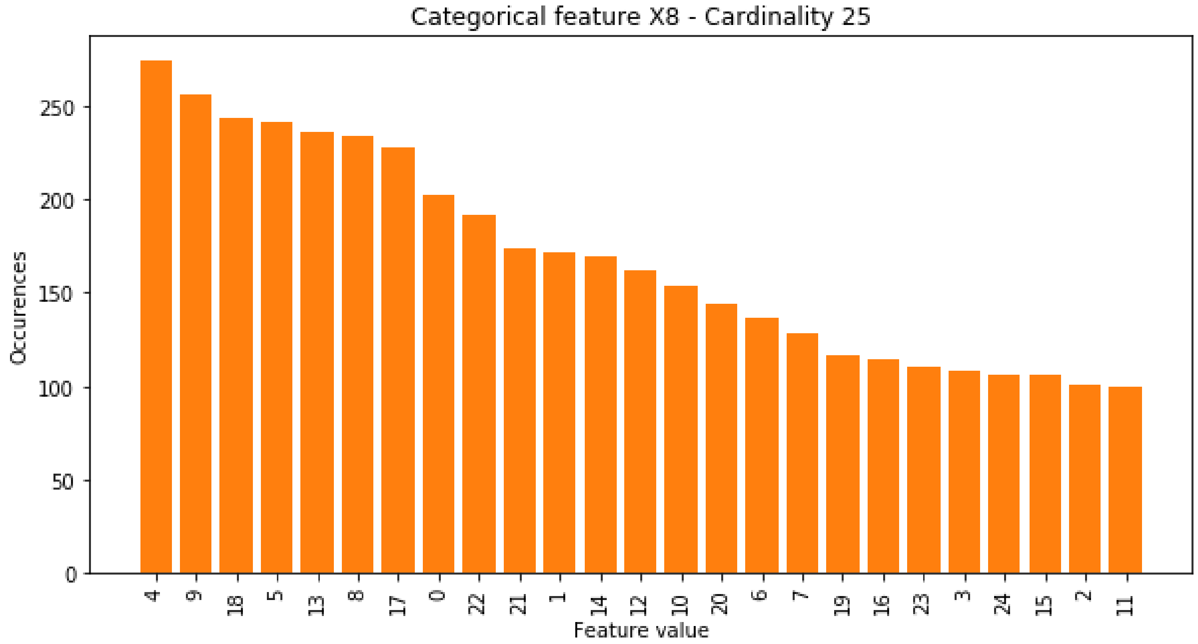



0
3
1
2
0
Feature value


Categorical feature X5 - Cardinality 32
250
200
150
100
50
0
Occurences
28 24 22 29 6 4 3 20 18 25 7 19 23 5 17 11 15 2 16 10 1 8 21 13 12 14 30 31 32 9 26 0
Feature value


Categorical feature X6 - Cardinality 12
1000
800
600
400
Occurences


200
0
6
9
3
8
11
7
0
10
2
5
1
4
Feature value

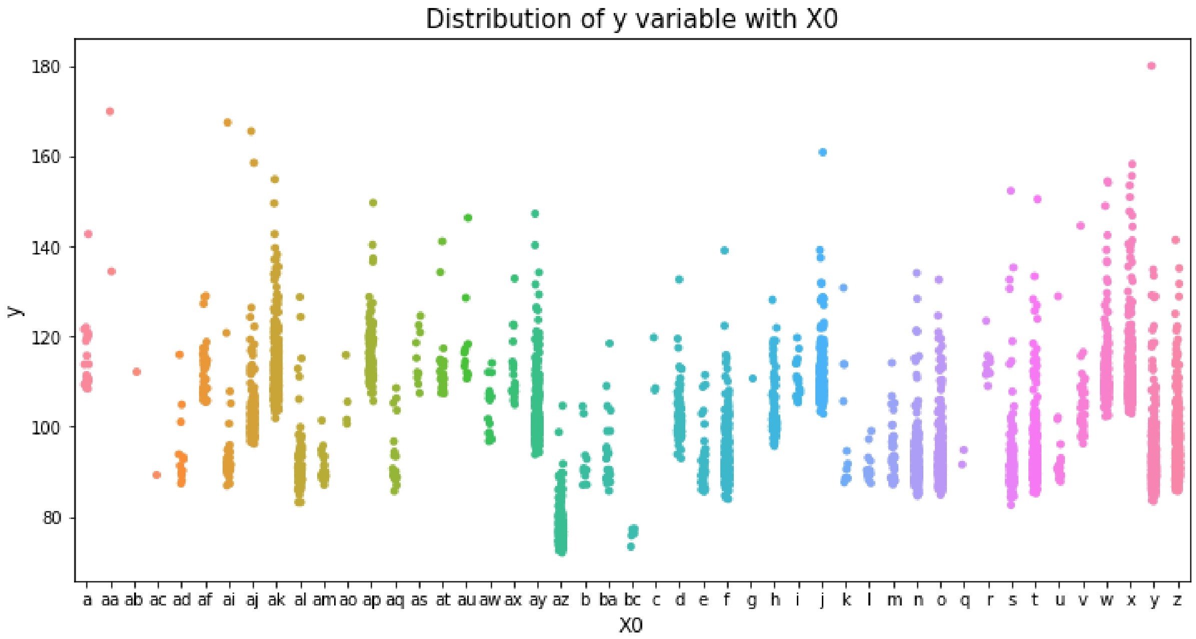

Categorical feature X8 - Cardinality 25
Occurences
250
200
150
100
50
0
4
9
18
5
13
8
17
0
22
21
1
14
12
10
20
6
7
19
16
23
3
24
15
2
11
Feature value

In [51]:
```
# checking for missing values
missing_df = df_train.isnull().sum(axis=0).reset_index()
missing_df.columns = ['column_name', 'missing_count']
missing_df = missing_df.loc[missing_df['missing_count']>0]
missing_df = missing_df.sort_values(by='missing_count')
missing_df
```

Out[51]:

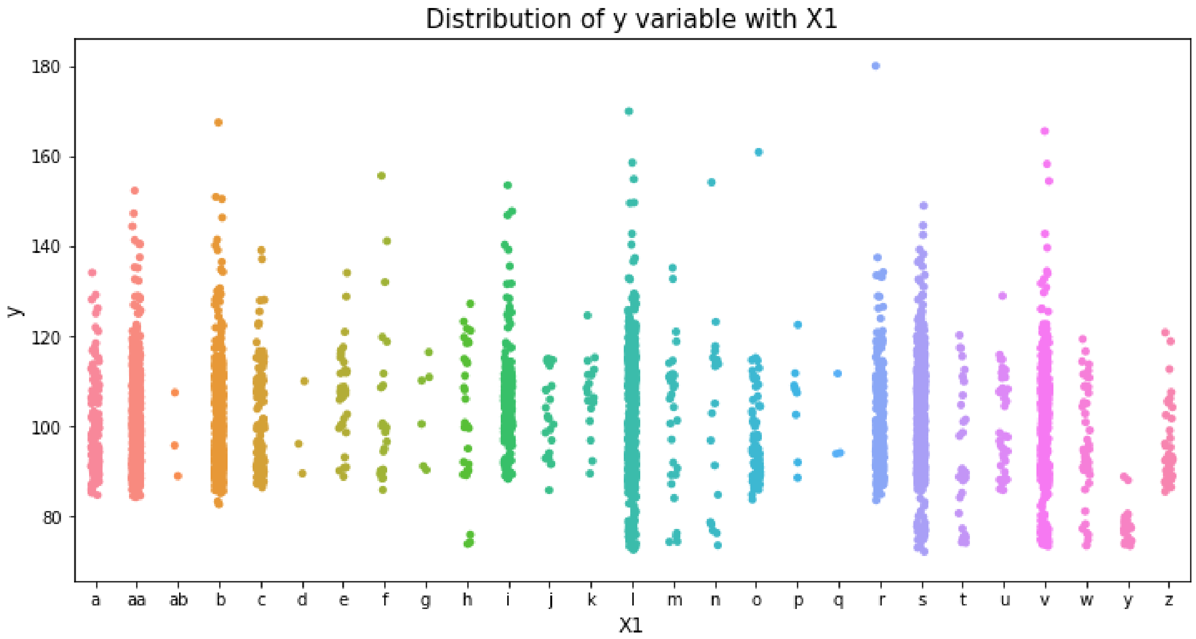
column_name	missing_count

In [52]:
```
# distribution with X0
var_name = "X0"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.stripplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```

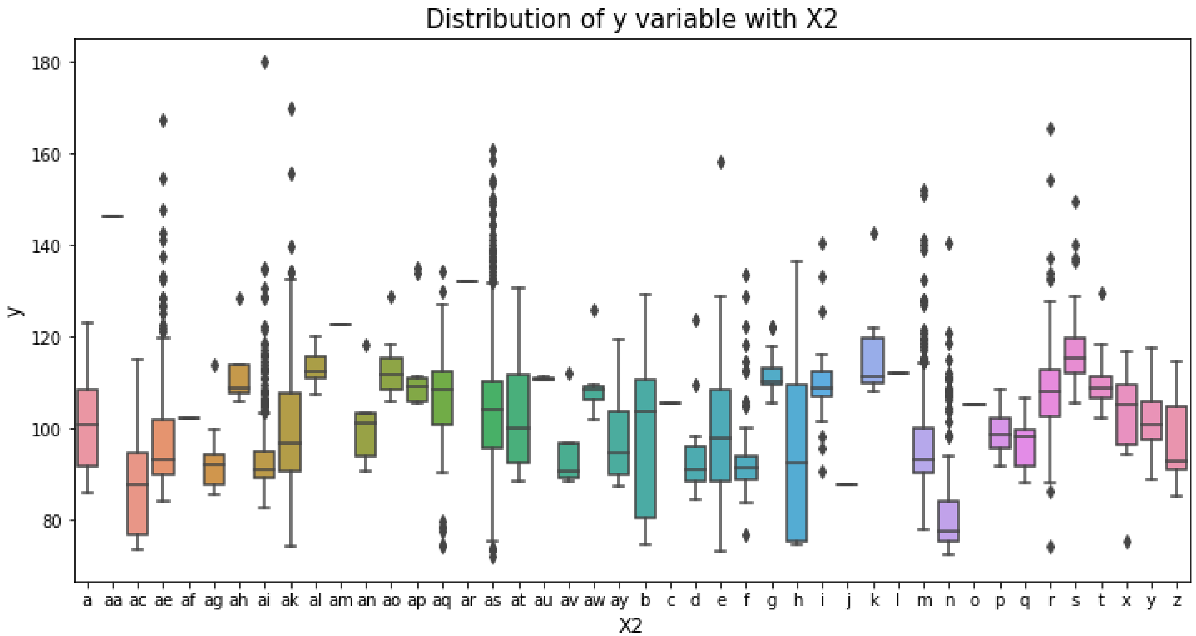


In [53]:

```
# distribution with X1
var_name = "X1"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.stripplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```

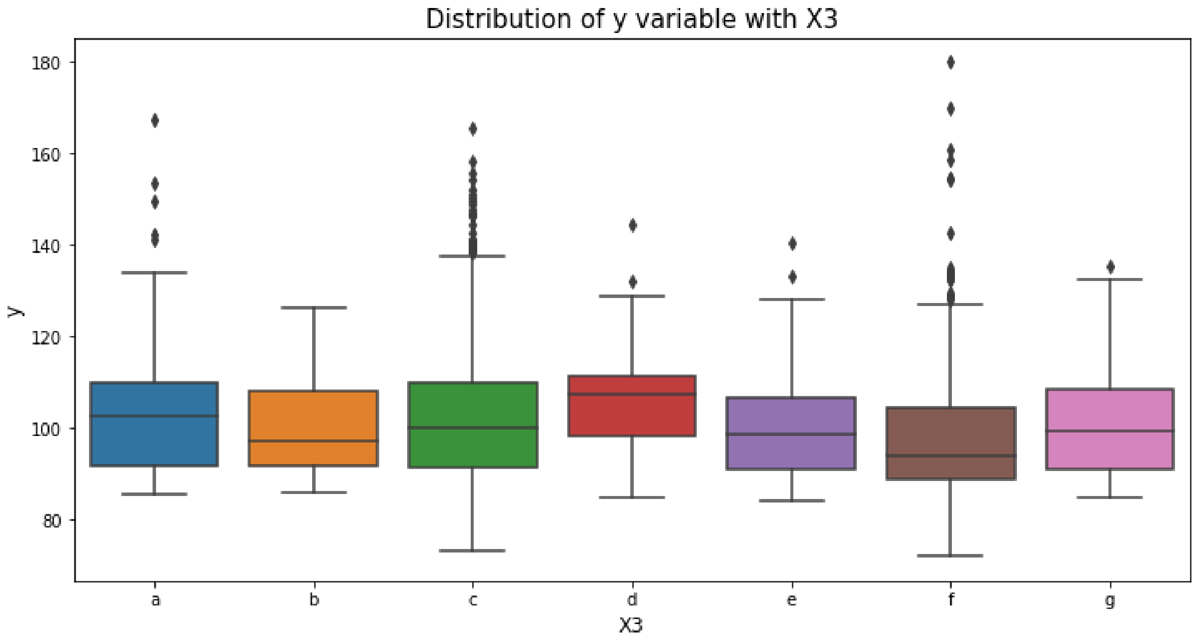


In [54]:
```
# distribution with X2
var_name = "X2"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```

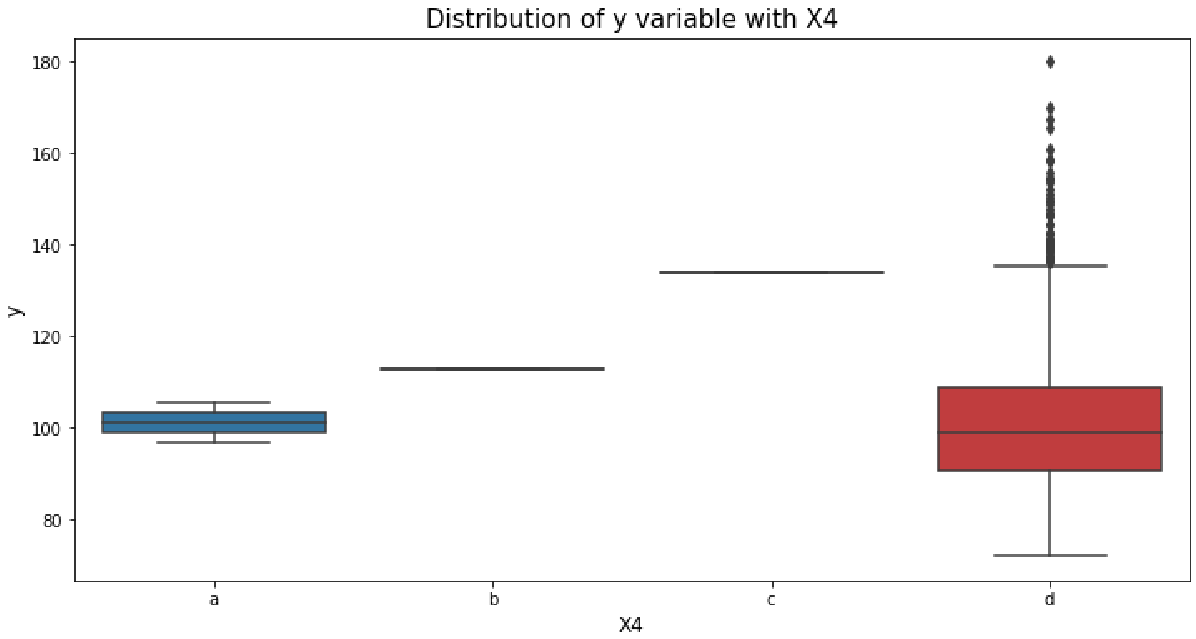


In [55]:

```
# distribution with X3
var_name = "X3"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```

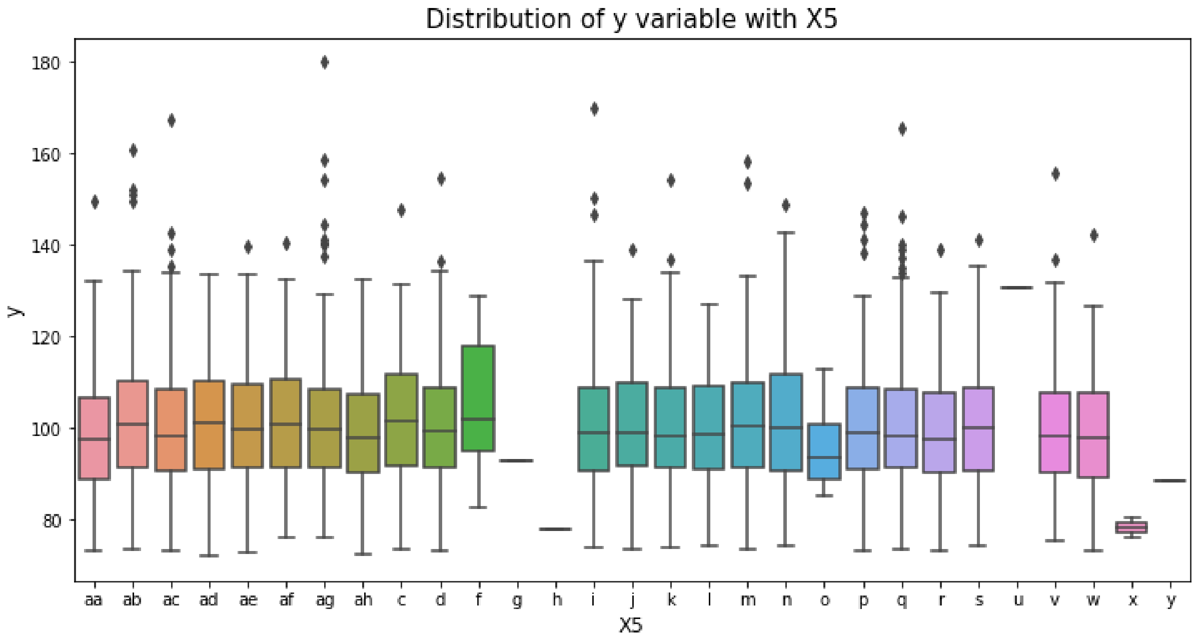


In [56]:
```
# distribution with X4
var_name = "X4"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```

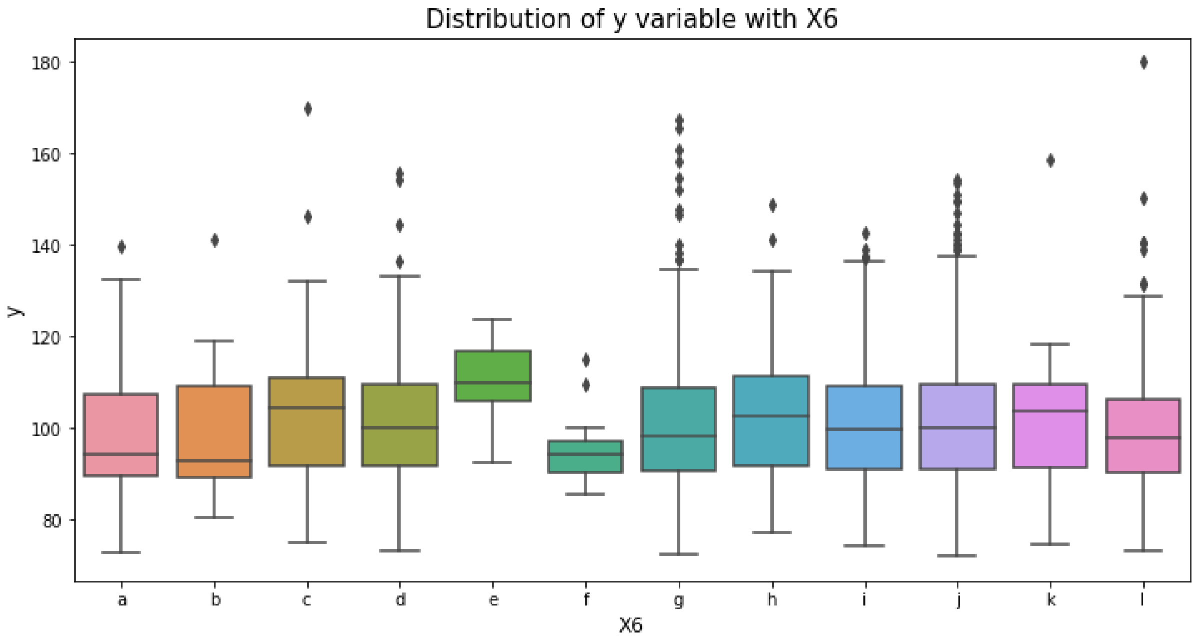


In [57]:

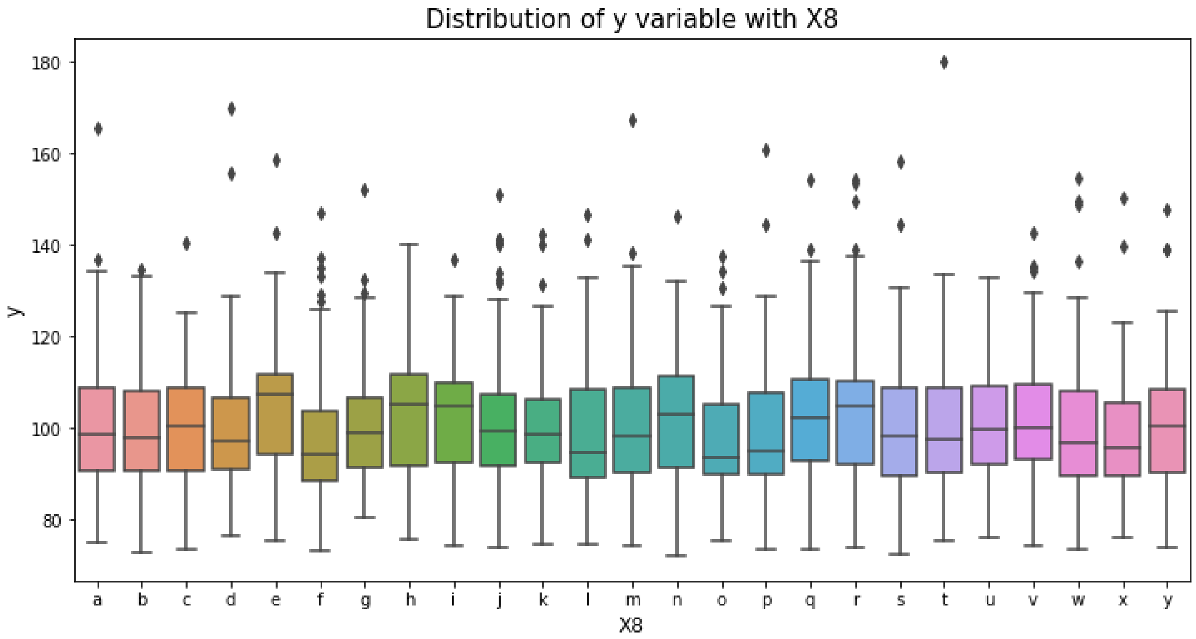
```
# distribution with X5
var_name = "X5"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```



In [58]:
```
# distribution with X6
var_name = "X6"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```



In [59]:
```
# distribution with X8
var_name = "X8"
col_order = np.sort(df_train[var_name].unique()).tolist()
plt.figure(figsize=(12,6))
sns.boxplot(x=var_name, y='y', data=df_train, order=col_order)
plt.xlabel(var_name, fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+var_name, fontsize=15)
plt.show()
```



In [60]:

```
# Step6: Read the test.csv data

df_test = pd.read_csv('test.csv')
df_test.head()
```

Out[60]:

	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 [283]:

```
#removing constant columns from data set
df_test = df_test.drop(['ID','X11', 'X93', 'X107', 'X233', 'X235', 'X26
8', 'X289', 'X290', 'X293', 'X297', 'X330', 'X347'], axis=1)
```

In [21]:

```
# remove columns ID and Y from the data as they are not used for learni
ng
usable_columns = list(set(df_train.columns) - set(['ID', 'y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
print('Train:',y_train)

#'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293',
 'X297', 'X330', 'X347'
```

```
Train: [130.81  88.53  76.26 ... 109.22  87.48 110.85]
```

In [22]:

```
x_train.info
```

Out[22]:

```
<bound method DataFrame.info of       X159  X32  X156  X100  X366  X109
  X13  X294  X227  X346  ...  X27  X35  \
0        0    0     1     0     0     0    1     0     0     0  ...
0    1
1        0    0     1     1     0     0    0     0     0     0  ...
1    1
2        0    0     0     0     0     0    0     0     0     0  ...
1    1
3        0    0     0     0     0     0    0     0     0     0  ...
1    1
4        0    0     0     0     0     0    0     0     0     0  ...
1    1
...    ...  ...   ...   ...   ...   ...  ...   ...   ...   ...  ...
...  ...
4204     0    0     0     1     0     0    0     0     0     0  ...
1    0
4205     0    0     1     0     0     0    0     1     0     0  ...
0    0
```

```
4206      0    0     1     1     0     0    1     0     0     0  ...
1    0
4207      0    0     1     0     0     0    0     0     0     0  ...
0    0
4208      0    0     1     1     0     0    0     0     0     0  ...
0    0

      X327  X225  X350  X310  X78  X170  X151  X299
0        1     0     0     0    0     1     0     0
1        0     0     0     0    0     0     0     0
2        0     0     1     0    0     0     0     0
3        0     0     1     0    0     0     0     0
4        0     0     1     0    0     0     0     0
...    ...   ...   ...   ...  ...   ...   ...   ...
4204     0     1     0     0    0     0     0     0
4205     0     0     0     0    0     0     0     0
4206     0     0     0     0    0     0     0     0
4207     0     0     0     0    0     0     0     0
4208     0     0     1     0    0     0     0     0

[4209 rows x 364 columns]>
```

In [24]:
```
df_train.head(10)
```

Out[24]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X38[illegible]
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	[illegible]
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	[illegible]
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	[illegible]
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	[illegible]
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	[illegible]
5	18	92.93	t	b	e	c	d	g	h	s	...	0	0	1	0	0	0	[illegible]
6	24	128.76	al	r	e	f	d	f	h	s	...	0	0	0	0	0	0	[illegible]
7	25	91.91	o	l	as	f	d	f	j	a	...	0	0	0	0	0	0	[illegible]

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X38
8	27	108.67	w	s	as	e	d	f	i	h	...	1	0	0	0	0	0	(
9	30	126.99	j	b	aq	c	d	f	a	e	...	0	0	1	0	0	0	(

10 rows × 366 columns

In [25]:
```
df_train_drop = df_train.drop(["ID", "y"], axis=1)
df_train_drop.head(5)
```

Out[25]:

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

5 rows × 364 columns

In [26]:
```
from sklearn import preprocessing
for f in ["X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(df_train_drop[f].values))
        df_train_drop[f] = lbl.transform(list(df_train_drop[f].values))
```

In [27]:
```
print('y' in df_test)
```

False

In [28]:
```
pd.DataFrame(y_train).head()
```

Out[28]:

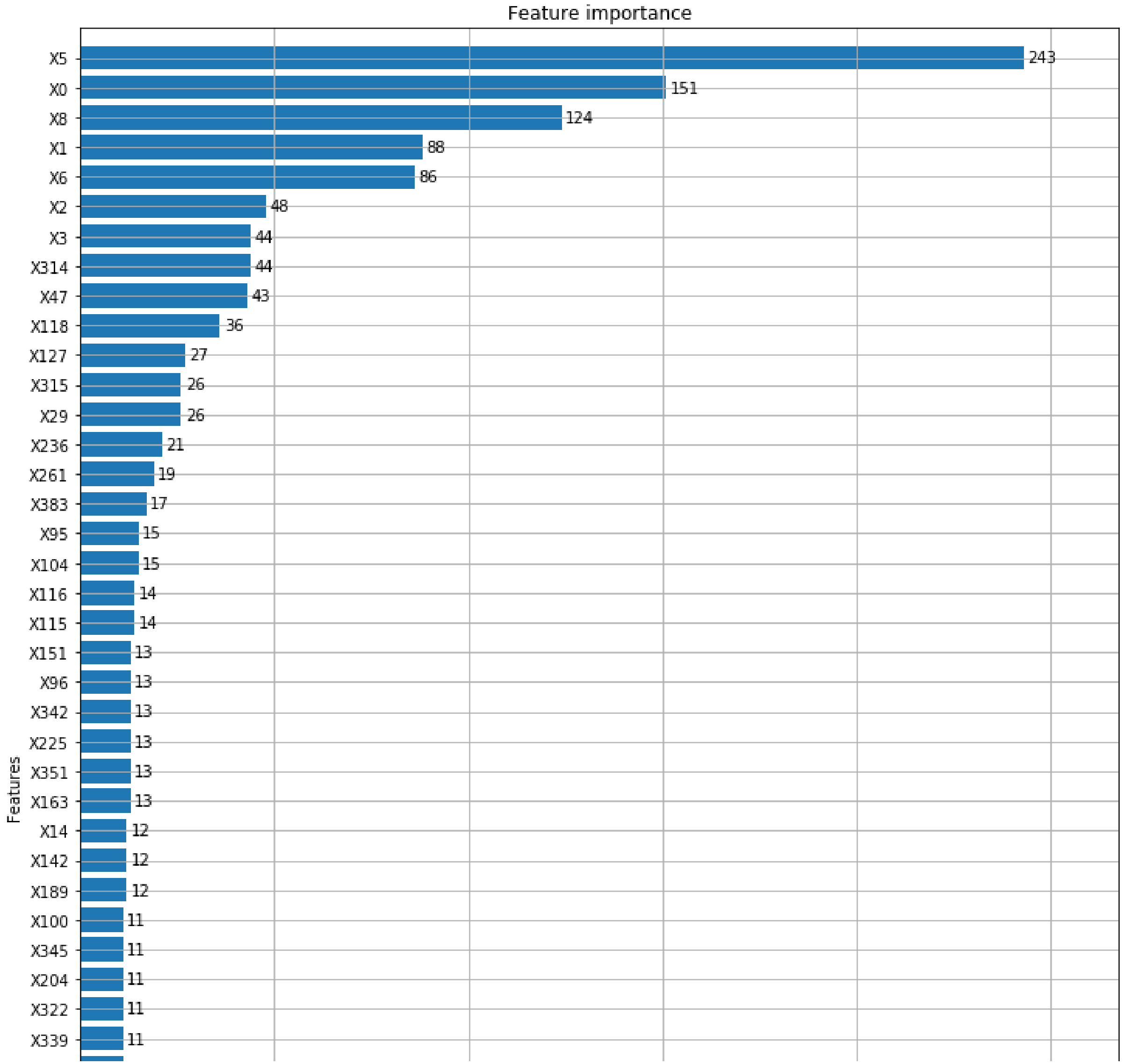
	0
0	130.81
1	88.53
2	76.26
3	80.62
4	78.02

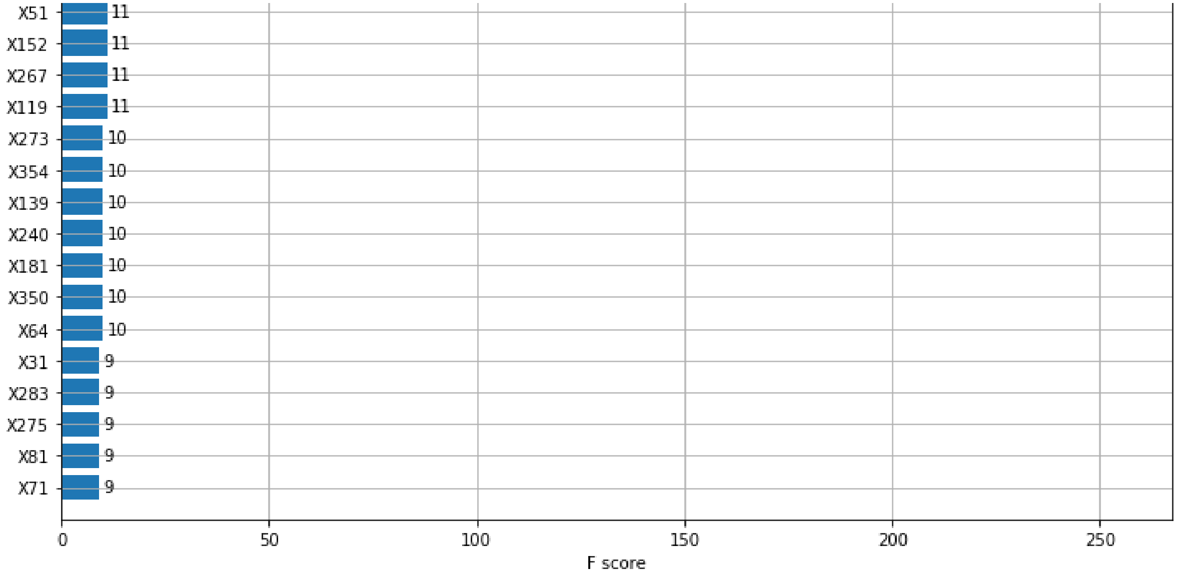
In [29]:

```
import xgboost as xgb
for f in ["X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(df_train[f].values))
        df_train[f] = lbl.transform(list(df_train[f].values))

train_y = df_train['y'].values
train_X = df_train.drop(["ID", "y"], axis=1)

# Thanks to anokas for this #
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

xgb_params = {
    'eta': 0.05,
    'max_depth': 6,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'objective': 'reg:linear',
    'silent': 1
}
dtrain = xgb.DMatrix(train_X, train_y, feature_names=train_X.columns.va
lues)

model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=1
00, feval=xgb_r2_score, maximize=True)

# plot the important features #
```

```
fig, ax = plt.subplots(figsize=(12,18))
xgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
plt.show()
```

```
[14:35:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederror.
```





In [61]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA, FastICA
from sklearn.metrics import r2_score
import random
```

In [62]:
```
random_seed = 0
random.seed(random_seed)
np.random.seed(random_seed)

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)
```

In [63]:
```
# Remove non-informative columns
cols_to_remove = []
for c in df_test.columns:
    if len(df_train[c].unique()) == 1:
        cols_to_remove.append(c)
print('Columns to remove: ' + str(cols_to_remove))
```

```
df_train = df_train.drop(cols_to_remove, axis=1)
df_test = df_test.drop(cols_to_remove, axis=1)
```

```
Columns to remove: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X28
9', 'X290', 'X293', 'X297', 'X330', 'X347']
```

In [66]:

```
# Process columns, apply LabelEncoder to categorical features
for c in df_train.columns:
    if df_train[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(df_train[c].values) + list(df_test[c].values))
        df_train[c] = lbl.transform(list(df_test[c].values))
        df_test[c] = lbl.transform(list(df_test[c].values))
```

In [67]:

```
import random
random_seed = 0
random.seed(random_seed)
np.random.seed(random_seed)

# Add decomposed components: PCA / ICA etc.
n_comp = 12

# PCA
pca = PCA(n_components=n_comp, random_state=random_seed)
pca2_results_train = pca.fit_transform(df_train.drop(["y"], axis=1))
pca2_results_test = pca.transform(df_test)
```

In [68]:

```
for i in range(1, n_comp+1):
    df_train['pca_' + str(i)] = pca2_results_train[:, i-1]
    df_test['pca_' + str(i)] = pca2_results_test[:, i-1]
```

In [69]:

```
# Prepare data
X = np.array(df_train.drop(['y'], axis=1))
y = df_train.y.values

y_mean = np.mean(y)

X_test = np.array(df_test)
```

```
ids_test = df_test.ID.values

print('X.shape = ' + str(X.shape) + ', y.shape = ' + str(y.shape))
print('X_test.shape = ' + str(X.shape))
```

```
X.shape = (4209, 377), y.shape = (4209,)
X_test.shape = (4209, 377)
```

In [75]:

```
#Run xgboost model
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import r2_score

params = {}
params['n_trees'] = 500
params['objective'] = 'reg:linear'
params['eta'] = 0.005
params['max_depth'] = 4
params['subsample'] = 0.95
params['base_score'] = y_mean
params['silent'] = 1

xgb_r2_buf = []
test_preds_buf = []
d_test = xgb.DMatrix(X_test)

cv = ShuffleSplit(n_splits=15, test_size=0.19, random_state=random_seed
)
fold_i = 0
for train_index, test_index in cv.split(X):
    print('Fold #' + str(fold_i))
    x_train, x_valid, y_train, y_valid = X[train_index], X[test_index],
 y[train_index], y[test_index]

    d_train = xgb.DMatrix(x_train, label=y_train)
    d_valid = xgb.DMatrix(x_valid, label=y_valid)

    print('XGB: Evaluating model')
    eval_set = [(x_train, y_train), (x_valid, y_valid)]
    watchlist = [(d_train, 'train'), (d_valid, 'valid')]
```

```
    model = xgb.train(params, d_train, 1000, watchlist, early_stopping_
rounds=50, \
        feval=xgb_r2_score, maximize=True, verbose_eval=100)

    p = model.predict(d_valid)
    r2 = r2_score(y_valid, p)
    xgb_r2_buf.append(r2)
    print('R2 = ' + str(r2))

    test_preds_buf.append(model.predict(d_test))

    fold_i += 1

print('XGB Mean R2 = ' + str(np.mean(xgb_r2_buf)) + ' +/- ' + str(np.st
d(xgb_r2_buf)))

print('XGB: Train on full dataset and predicting on test')
d_train = xgb.DMatrix(X, label=y)
watchlist = [(d_train, 'train')]
model = xgb.train(params, d_train, 700, watchlist, feval=xgb_r2_score,
 \
    maximize=True, verbose_eval=100)

p_test = model.predict(d_test)

test_preds_buf = np.array(test_preds_buf).T
test_preds_buf = np.concatenate((test_preds_buf, p_test.reshape((len(p_
test),1))), axis=1)
```

```
Fold #0
XGB: Evaluating model
[0]     train-rmse:12.3893      valid-rmse:12.6746      train-r2:0.0059
74      valid-r2:0.003946
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.68649      valid-rmse:10.1996      train-r2:0.3923
77      valid-r2:0.354967
```

```
[200]   train-rmse:8.4342       valid-rmse:9.12854      train-r2:0.5393
31      valid-r2:0.483327
[300]   train-rmse:7.88115      valid-rmse:8.70722      train-r2:0.5977
64      valid-r2:0.529919
[400]   train-rmse:7.61888      valid-rmse:8.55482      train-r2:0.6240
9       valid-r2:0.546231
[500]   train-rmse:7.46455      valid-rmse:8.50316      train-r2:0.6391
66      valid-r2:0.551694
[600]   train-rmse:7.3556       valid-rmse:8.4898       train-r2:0.6496
22      valid-r2:0.553102
[700]   train-rmse:7.26024      valid-rmse:8.48145      train-r2:0.6586
47      valid-r2:0.553981
[800]   train-rmse:7.16607      valid-rmse:8.48185      train-r2:0.6674
45      valid-r2:0.553939
Stopping. Best iteration:
[774]   train-rmse:7.18917      valid-rmse:8.48031      train-r2:0.6652
97      valid-r2:0.554101

R2 = 0.5537233795132259
Fold #1
XGB: Evaluating model
[0]     train-rmse:12.5163      valid-rmse:12.1318      train-r2:0.0056
46      valid-r2:0.002423
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.89364      valid-rmse:9.36452      train-r2:0.3787
01      valid-r2:0.405611
[200]   train-rmse:8.68618      valid-rmse:8.12224      train-r2:0.5210
98      valid-r2:0.552852
[300]   train-rmse:8.14671      valid-rmse:7.61282      train-r2:0.5787
37      valid-r2:0.607182
[400]   train-rmse:7.89033      valid-rmse:7.41632      train-r2:0.6048
35      valid-r2:0.627199
[500]   train-rmse:7.731        valid-rmse:7.34044      train-r2:0.6206
32      valid-r2:0.634789
[600]   train-rmse:7.60912      valid-rmse:7.30607      train-r2:0.6325
valid-r2:0.638202
[700]   train-rmse:7.50705      valid-rmse:7.28486      train-r2:0.6422
```

```
93      valid-r2:0.640299
[800]   train-rmse:7.42381      valid-rmse:7.27258      train-r2:0.6501
82      valid-r2:0.64151
[900]   train-rmse:7.34147      valid-rmse:7.26634      train-r2:0.6578

98      valid-r2:0.642125
Stopping. Best iteration:
[900]   train-rmse:7.34147      valid-rmse:7.26634      train-r2:0.6578
98      valid-r2:0.642125

R2 = 0.642065141374909
Fold #2
XGB: Evaluating model
[0]     train-rmse:12.611       valid-rmse:11.7045      train-r2:0.0058
33      valid-r2:0.00462
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.96173      valid-rmse:9.0238       train-r2:0.3796
63      valid-r2:0.408351
[200]   train-rmse:8.73734      valid-rmse:7.82195      train-r2:0.5227
83      valid-r2:0.555455
[300]   train-rmse:8.18853      valid-rmse:7.33984      train-r2:0.5808
5       valid-r2:0.608566
[400]   train-rmse:7.92848      valid-rmse:7.15359      train-r2:0.6070
5       valid-r2:0.628179
[500]   train-rmse:7.7684       valid-rmse:7.08954      train-r2:0.6227
57      valid-r2:0.634808
[600]   train-rmse:7.64809      valid-rmse:7.06735      train-r2:0.6343
52      valid-r2:0.63709
[700]   train-rmse:7.55044      valid-rmse:7.06164      train-r2:0.6436
29      valid-r2:0.637676
Stopping. Best iteration:
[694]   train-rmse:7.55724      valid-rmse:7.061        train-r2:0.6429
87      valid-r2:0.637742

R2 = 0.63740791872011
Fold #3
XGB: Evaluating model
```

```
[0]     train-rmse:12.3273      valid-rmse:12.9319      train-r2:0.0059
5       valid-r2:0.005651
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.


Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.71689      valid-rmse:10.3525      train-r2:0.3823
69      valid-r2:0.362763
[200]   train-rmse:8.51385      valid-rmse:9.18078      train-r2:0.5258
38      valid-r2:0.498843
[300]   train-rmse:7.98282      valid-rmse:8.67824      train-r2:0.5831
44      valid-r2:0.552206
[400]   train-rmse:7.72597      valid-rmse:8.47138      train-r2:0.6095
37      valid-r2:0.5733
[500]   train-rmse:7.55644      valid-rmse:8.3903       train-r2:0.6264
84      valid-r2:0.581429
[600]   train-rmse:7.43851      valid-rmse:8.35078      train-r2:0.6380
52      valid-r2:0.585362
[700]   train-rmse:7.33148      valid-rmse:8.33737      train-r2:0.6483
93      valid-r2:0.586694
Stopping. Best iteration:
[713]   train-rmse:7.32138      valid-rmse:8.33675      train-r2:0.6493
61      valid-r2:0.586755

R2 = 0.5865368093446632
Fold #4
XGB: Evaluating model
[0]     train-rmse:12.5453      valid-rmse:12.003       train-r2:0.0058
74      valid-r2:0.002024
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.81186      valid-rmse:9.69455      train-r2:0.3918
87      valid-r2:0.348974
[200]   train-rmse:8.54357      valid-rmse:8.69406      train-r2:0.5389
37      valid-r2:0.476413
[300]   train-rmse:7.98113      valid-rmse:8.29937      train-r2:0.5976
44      valid-r2:0.522874
```

```
[400]	train-rmse:7.71103	valid-rmse:8.15703	train-r2:0.6244
17	valid-r2:0.539099
[500]	train-rmse:7.54166	valid-rmse:8.11851	train-r2:0.6407
35	valid-r2:0.543442

[600]	train-rmse:7.42421	valid-rmse:8.11016	train-r2:0.6518
38	valid-r2:0.544381
Stopping. Best iteration:
[632]	train-rmse:7.39148	valid-rmse:8.10956	train-r2:0.6549
valid-r2:0.544448

R2 = 0.5440721550892719
Fold #5
XGB: Evaluating model
[0]	train-rmse:12.3326	valid-rmse:12.9067	train-r2:0.0059
58	valid-r2:0.004045
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]	train-rmse:9.66722	valid-rmse:10.3438	train-r2:0.3892
02	valid-r2:0.360311
[200]	train-rmse:8.42585	valid-rmse:9.24805	train-r2:0.5359
96	valid-r2:0.488662
[300]	train-rmse:7.8771	valid-rmse:8.83728	train-r2:0.5944
67	valid-r2:0.533077
[400]	train-rmse:7.61836	valid-rmse:8.70461	train-r2:0.6206
7	valid-r2:0.546992
[500]	train-rmse:7.46834	valid-rmse:8.68005	train-r2:0.6354
63	valid-r2:0.549545
Stopping. Best iteration:
[505]	train-rmse:7.46078	valid-rmse:8.6793	train-r2:0.6362
valid-r2:0.549622

R2 = 0.5495068935342887
Fold #6
XGB: Evaluating model
[0]	train-rmse:12.4728	valid-rmse:12.3215	train-r2:0.0058
93	valid-r2:0.00557
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
```

```
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]	train-rmse:9.84196	valid-rmse:9.69324	train-r2:0.3810

29	valid-r2:0.384557
[200]	train-rmse:8.62225	valid-rmse:8.53553	train-r2:0.5249
4	valid-r2:0.522789
[300]	train-rmse:8.07829	valid-rmse:8.07553	train-r2:0.5829
91	valid-r2:0.572838
[400]	train-rmse:7.81707	valid-rmse:7.89781	train-r2:0.6095
23	valid-r2:0.591433
[500]	train-rmse:7.65346	valid-rmse:7.83943	train-r2:0.6256
98	valid-r2:0.597451
[600]	train-rmse:7.53747	valid-rmse:7.82011	train-r2:0.6369
57	valid-r2:0.599433
[700]	train-rmse:7.43811	valid-rmse:7.80745	train-r2:0.6464
65	valid-r2:0.600729
[800]	train-rmse:7.34846	valid-rmse:7.80371	train-r2:0.6549
36	valid-r2:0.601111
Stopping. Best iteration:
[806]	train-rmse:7.34181	valid-rmse:7.8032	train-r2:0.6555
6	valid-r2:0.601163

R2 = 0.6009450118429852
Fold #7
XGB: Evaluating model
[0]	train-rmse:12.5173	valid-rmse:12.1279	train-r2:0.0059
05	valid-r2:0.005953
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]	train-rmse:9.86444	valid-rmse:9.54975	train-r2:0.3826
26	valid-r2:0.383665
[200]	train-rmse:8.63426	valid-rmse:8.43408	train-r2:0.5270
08	valid-r2:0.519263
[300]	train-rmse:8.08754	valid-rmse:8.00282	train-r2:0.5850
12	valid-r2:0.567169
[400]	train-rmse:7.81877	valid-rmse:7.86416	train-r2:0.6121
```

```
35      valid-r2:0.582037
[500]   train-rmse:7.65236      valid-rmse:7.81952      train-r2:0.6284
69      valid-r2:0.58677
Stopping. Best iteration:

[528]   train-rmse:7.61326      valid-rmse:7.81738      train-r2:0.6322
57      valid-r2:0.586995

R2 = 0.5868282412433511
Fold #8
XGB: Evaluating model
[0]     train-rmse:12.4313      valid-rmse:12.4959      train-r2:0.0061
33      valid-r2:0.0056
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.73011      valid-rmse:10.0906      train-r2:0.3911
19      valid-r2:0.35158
[200]   train-rmse:8.47598      valid-rmse:9.05163      train-r2:0.5379
64      valid-r2:0.478229
[300]   train-rmse:7.917        valid-rmse:8.6347       train-r2:0.5968
96      valid-r2:0.525189
[400]   train-rmse:7.64708      valid-rmse:8.48455      train-r2:0.6239
14      valid-r2:0.541559
[500]   train-rmse:7.49702      valid-rmse:8.4283       train-r2:0.6385
29      valid-r2:0.547617
[600]   train-rmse:7.38044      valid-rmse:8.40873      train-r2:0.6496
83      valid-r2:0.549716
[700]   train-rmse:7.2822       valid-rmse:8.3975       train-r2:0.6589
47      valid-r2:0.550918
Stopping. Best iteration:
[711]   train-rmse:7.27118      valid-rmse:8.39641      train-r2:0.6599
79      valid-r2:0.551035

R2 = 0.5507580365436747
Fold #9
XGB: Evaluating model
[0]     train-rmse:12.4306      valid-rmse:12.4983      train-r2:0.0059
76      valid-r2:0.002139
```

```
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.

[100]   train-rmse:9.69792      valid-rmse:10.1247      train-r2:0.3949
79      valid-r2:0.345156
[200]   train-rmse:8.42371      valid-rmse:9.15457      train-r2:0.5435
23      valid-r2:0.464641
[300]   train-rmse:7.85827      valid-rmse:8.80293      train-r2:0.6027
48      valid-r2:0.504978
[400]   train-rmse:7.58807      valid-rmse:8.68955      train-r2:0.6295
97      valid-r2:0.517648
[500]   train-rmse:7.41735      valid-rmse:8.65634      train-r2:0.6460
76      valid-r2:0.521328
[600]   train-rmse:7.29266      valid-rmse:8.65122      train-r2:0.6578
75      valid-r2:0.521894
[700]   train-rmse:7.19189      valid-rmse:8.64839      train-r2:0.6672
64      valid-r2:0.522207
Stopping. Best iteration:
[660]   train-rmse:7.23124      valid-rmse:8.64787      train-r2:0.6636
14      valid-r2:0.522264

R2 = 0.5221948932738631
Fold #10
XGB: Evaluating model
[0]     train-rmse:12.4058      valid-rmse:12.6048      train-r2:0.0060
6       valid-r2:0.005236
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.72152      valid-rmse:10.1561      train-r2:0.3896
53      valid-r2:0.354189
[200]   train-rmse:8.47683      valid-rmse:9.10896      train-r2:0.5359
39      valid-r2:0.4805
[300]   train-rmse:7.92431      valid-rmse:8.69237      train-r2:0.5944
62      valid-r2:0.526932
[400]   train-rmse:7.65499      valid-rmse:8.54915      train-r2:0.6215
59      valid-r2:0.542392
```

```
[500]   train-rmse:7.48142      valid-rmse:8.5031       train-r2:0.6385
27      valid-r2:0.547309
[600]   train-rmse:7.35802      valid-rmse:8.48531      train-r2:0.6503
53      valid-r2:0.5492

[700]   train-rmse:7.25699      valid-rmse:8.48416      train-r2:0.6598
89      valid-r2:0.549322
Stopping. Best iteration:
[718]   train-rmse:7.24247      valid-rmse:8.48306      train-r2:0.6612
49      valid-r2:0.54944

R2 = 0.5490754808411635
Fold #11
XGB: Evaluating model
[0]     train-rmse:12.5633      valid-rmse:11.9202      train-r2:0.0058
85      valid-r2:0.004705
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.89804      valid-rmse:9.29071      train-r2:0.3829
45      valid-r2:0.395384
[200]   train-rmse:8.65884      valid-rmse:8.13491      train-r2:0.5277
79      valid-r2:0.53646
[300]   train-rmse:8.10945      valid-rmse:7.6857       train-r2:0.5858
02      valid-r2:0.58624
[400]   train-rmse:7.84778      valid-rmse:7.53281      train-r2:0.6121
01      valid-r2:0.602538
[500]   train-rmse:7.68815      valid-rmse:7.48332      train-r2:0.6277
2       valid-r2:0.607743
[600]   train-rmse:7.57515      valid-rmse:7.46642      train-r2:0.6385
84      valid-r2:0.609513
[700]   train-rmse:7.48261      valid-rmse:7.45905      train-r2:0.6473
6       valid-r2:0.610284
Stopping. Best iteration:
[703]   train-rmse:7.4803       valid-rmse:7.4589       train-r2:0.6475
78      valid-r2:0.610299

R2 = 0.610009593752336
Fold #12
```

```
XGB: Evaluating model
[0]     train-rmse:12.421       valid-rmse:12.5435      train-r2:0.0059
2       valid-r2:0.00569
Multiple eval metrics have been passed: 'valid-r2' will be used for ear

ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.77454      valid-rmse:9.95578      train-r2:0.3843
97      valid-r2:0.373627
[200]   train-rmse:8.54655      valid-rmse:8.78378      train-r2:0.5293
6       valid-r2:0.51242
[300]   train-rmse:8.00263      valid-rmse:8.30341      train-r2:0.5873
58      valid-r2:0.564292
[400]   train-rmse:7.74514      valid-rmse:8.11945      train-r2:0.6134
85      valid-r2:0.583384
[500]   train-rmse:7.58596      valid-rmse:8.06402      train-r2:0.6292
09      valid-r2:0.589053
[600]   train-rmse:7.46754      valid-rmse:8.0498       train-r2:0.6406
95      valid-r2:0.590501
[700]   train-rmse:7.3718       valid-rmse:8.04609      train-r2:0.6498
49      valid-r2:0.590878
Stopping. Best iteration:
[689]   train-rmse:7.38203      valid-rmse:8.04545      train-r2:0.6488
77      valid-r2:0.590944

R2 = 0.5907237381797608
Fold #13
XGB: Evaluating model
[0]     train-rmse:12.4059      valid-rmse:12.6037      train-r2:0.0060
9       valid-r2:0.00519
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.68562      valid-rmse:10.1648      train-r2:0.3941
8       valid-r2:0.352948
[200]   train-rmse:8.41829      valid-rmse:9.13707      train-r2:0.5423
46      valid-r2:0.477176
[300]   train-rmse:7.85494      valid-rmse:8.74143      train-r2:0.6015
```

```
49      valid-r2:0.521474
[400]   train-rmse:7.58776      valid-rmse:8.60402      train-r2:0.6281
94      valid-r2:0.5364
[500]   train-rmse:7.41576      valid-rmse:8.56644      train-r2:0.6448

6       valid-r2:0.540441
[600]   train-rmse:7.29476      valid-rmse:8.5566       train-r2:0.6563
54      valid-r2:0.541495
[700]   train-rmse:7.1973       valid-rmse:8.53711      train-r2:0.6654
75      valid-r2:0.543582
[800]   train-rmse:7.10876      valid-rmse:8.52682      train-r2:0.6736
55      valid-r2:0.544682
Stopping. Best iteration:
[838]   train-rmse:7.08148      valid-rmse:8.52313      train-r2:0.6761
55      valid-r2:0.545076

R2 = 0.544743176279407
Fold #14
XGB: Evaluating model
[0]     train-rmse:12.3692      valid-rmse:12.7592      train-r2:0.0059
83      valid-r2:0.005267
Multiple eval metrics have been passed: 'valid-r2' will be used for ear
ly stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[100]   train-rmse:9.70448      valid-rmse:10.2752      train-r2:0.3881
4       valid-r2:0.354886
[200]   train-rmse:8.47227      valid-rmse:9.15821      train-r2:0.5336
55      valid-r2:0.487517
[300]   train-rmse:7.92213      valid-rmse:8.68632      train-r2:0.5922
52      valid-r2:0.53897
[400]   train-rmse:7.66434      valid-rmse:8.48987      train-r2:0.6183
57      valid-r2:0.559587
[500]   train-rmse:7.49447      valid-rmse:8.40326      train-r2:0.6350
87      valid-r2:0.568527
[600]   train-rmse:7.36331      valid-rmse:8.36607      train-r2:0.6477
48      valid-r2:0.572338
[700]   train-rmse:7.25966      valid-rmse:8.35116      train-r2:0.6575
95      valid-r2:0.57386
[800]   train-rmse:7.16258      valid-rmse:8.3449       train-r2:0.6666
```

```
91      valid-r2:0.5745
[900]   train-rmse:7.07952      valid-rmse:8.34253      train-r2:0.6743
76      valid-r2:0.574741
Stopping. Best iteration:

[918]   train-rmse:7.06451      valid-rmse:8.34013      train-r2:0.6757
56      valid-r2:0.574985

R2 = 0.5746894040759057
XGB Mean R2 = 0.5762186582405945 +/- 0.03445972563874399
XGB: Train on full dataset and predicting on test
[0]     train-rmse:12.444       train-r2:0.00596
[100]   train-rmse:9.80354      train-r2:0.383054
[200]   train-rmse:8.59265      train-r2:0.526047
[300]   train-rmse:8.05873      train-r2:0.583117
[400]   train-rmse:7.81088      train-r2:0.608365
[500]   train-rmse:7.66162      train-r2:0.62319
[600]   train-rmse:7.54865      train-r2:0.63422
[699]   train-rmse:7.46002      train-r2:0.642759
```

In [76]:
```
p_test
```

Out[76]:
```
array([ 80.90558,  96.23905,  80.05758, ...,  92.76221, 109.78918,
        93.22348], dtype=float32)
```

In [77]:
```
test_preds_buf
```

Out[77]:
```
array([[ 83.03288 ,  82.82963 ,  82.8795  , ...,  81.98509 ,  84.61196
,
         80.90558 ],
       [101.50865 ,  96.266556, 102.15492 , ..., 102.70724 ,  96.80157
,
         96.23905 ],
       [ 80.24612 ,  79.27248 ,  80.28389 , ...,  80.21457 ,  79.37524
4,
         80.05758 ],
       ...,
       [ 93.42518 ,  92.8904  ,  92.31735 , ...,  92.715485,  93.73725
,
```

```
         92.76221 ],
        [111.294586, 111.494286, 109.97745 , ..., 110.63302 , 109.53612
,
         109.78918 ],
        [ 93.846   ,  94.91789 ,  93.52831 , ...,  93.68853 ,  93.23867
,
          93.22348 ]], dtype=float32)
```

In [ ]:

In [ ]:

In [ ]: