Mercedes-Benz Greener Manufacturing

The Personal Case Study

1.Business Problem

1.1 Description

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include, for example, the passenger safety cell with 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 car makers. Daimler's Mercedes-Benz cars are leaders 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 each and every unique car configuration before they hit the road, Daimler's engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations is complex and time-consuming without a powerful algorithmic approach. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler's production lines.

1.2 Data Description:

This dataset contains an anonymized set of variables, each representing a custom feature in a Mercedes car. For example, a variable could be 4WD, added air suspension, or a head-up display.

The ground truth is labeled ‘y’ and represents the time (in seconds) that the car took to pass testing for each variable.

1.3 Objective

1. Reduce the time that cars spend on the test bench.
2. To speedier testing, resulting in lower carbon dioxide emissions without reducing Daimler’s standards.

1.4 Sources/Useful Links

Main Source: https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/overview/description (https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/overview/description)
Data: https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data (https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data)
Discussion: https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion (https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion)

2. Machine Learning Problem

2.1 Data

2.1.1 Data Overview

Get the data from : https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data (https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data)

- Data will be in a file Train.csv and Test.csv
- Size of Train.csv - 3.07MB
- Size of Train.csv - 3.04 MB
- Number of rows and columns in Train.csv = 4210 x 378
- Number of rows and columns in Test.csv = 4210 x 377

2.1.2 Example Data point

(CLOUMNS)ID y X0 X1 X2 X3 X4 X5 X6 X8 X10 X11 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 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49 X50 X51 X52 X53 X54 X55 X56 X57 X58 X59 X60 X61 X62 X63 X64 X65 X66 X67 X68 X69 X70 X71 X73 X74 X75 X76 X77 X78 X79 X80 X81 X82 X83 X84 X85 X86 X87 X88 X89 X90 X91 X92 X93 X94 X95 X96 X97 X98 X99 X100 X101 X102 X103 X104 X105 X106 X107 X108 X109 X110 X111 X112 X113 X114 X115 X116 X117 X118 X119 X120 X122 X123 X124 X125 X126 X127 X128 X129 X130 X131 X132 X133 X134 X135 X136 X137 X138 X139 X140 X141 X142 X143 X144 X145 X146 X147 X148 X150 X151 X152 X153 X154 X155 X156 X157 X158 X159 X160 X161 X162 X163 X164 X165 X166 X167 X168 X169 X170 X171 X172 X173 X174 X175 X176 X177 X178 X179 X180 X181 X182 X183 X184 X185 X186 X187 X189 X190 X191 X192 X194 X195 X196 X197 X198 X199 X200 X201 X202 X203 X204 X205 X206 X207 X208 X209 X210 X211 X212 X213 X214 X215 X216 X217 X218 X219 X220 X221 X222 X223 X224 X225 X226 X227 X228 X229 X230 X231 X232 X233 X234 X235 X236 X237 X238 X239 X240 X241 X242 X243 X244 X245 X246 X247 X248 X249 X250 X251 X252 X253 X254 X255 X256 X257 X258 X259 X260 X261 X262 X263 X264 X265 X266 X267 X268 X269 X270 X271 X272 X273 X274 X275 X276 X277 X278 X279 X280 X281 X282 X283 X284 X285 X286 X287 X288 X289 X290 X291 X292 X293 X294 X295 X296 X297 X298 X299 X300 X301 X302 X304 X305 X306 X307 X308 X309 X310 X311 X312 X313 X314 X315 X316 X317 X318 X319 X320 X321 X322 X323 X324 X325 X326 X327 X328 X329 X330 X331 X332 X333 X334 X335 X336 X337 X338 X339 X340 X341 X342 X343 X344 X345 X346 X347 X348 X349 X350 X351 X352 X353 X354 X355 X356 X357 X358 X359 X360 X361 X362 X363 X364 X365 X366 X367 X368 X369 X370 X371 X372 X373 X374 X375 X376 X377 X378 X379 X380 X382 X383 X384 X385

(1st ROW)0 130.81 k v at a d u j o 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

2.2 Mapping the real world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

We need to Reduce the time that cars spend on the test bench.

2.2.2 Performance Metric

link: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)

r2_score:

- sklearn.metrics.r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')
- R^2 (coefficient of determination) regression score function.

- Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

3.1 Importing important libraries

In [53]:
```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import xgboost as xgb
color = sns.color_palette()
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score
%matplotlib inline
from sklearn.svm import SVR, LinearSVC
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
import scipy.stats as stats
from sklearn.externals import joblib

from sklearn.pipeline import make_pipeline
from sklearn.linear_model import ElasticNetCV, LassoLarsCV
from sklearn.ensemble import ExtraTreesRegressor, GradientBoostingRegressor
from tpot.builtins import StackingEstimator
```

3.1 Reading data and basic stats

In [2]:
```
# Load the Drive helper and mount
#from google.colab import drive
# This will prompt for authorization.
#drive.mount('/content/drive')
```

In [3]:
```
#loading data from google drive
#train_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Personal Case STudy/
train.csv')
#print("Train shape : ", train_df.shape)
#test_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Personal Case STudy/t
est.csv')
#print("Train shape : ", test_df.shape)
```

In [2]:
```
#loading data from HDD
train_df = pd.read_csv('train.csv')
print("Train shape : ", train_df.shape)
test_df = pd.read_csv('test.csv')
print("Train shape : ", test_df.shape)
```

```
Train shape :  (4209, 378)
Train shape :  (4209, 377)
```

In [3]:
```
train_df.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 [4]:
```
train_df.head()
```

Out[4]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X38
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	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
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0
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

5 rows × 378 columns

Target Variable:

- "y" is the variable we need to predict. So let us do some analysis on this variable first.
- Varible y is of type float
- X0,X1,X2,X3,X4,X5,X6,X8 are of type object
- Rest of the columns are int type
- We will convert [X0,X1,X2,X3,X4,X5,X6,X8] to categorical types and plot to see the distribution of values.

3.2 Checking for missing values

In [6]:
```
def check_missing_values(df):

    if df.isnull().any().any():
        print("There are missing values in the data")
    else:
        print("There are no missing values in the data")
```

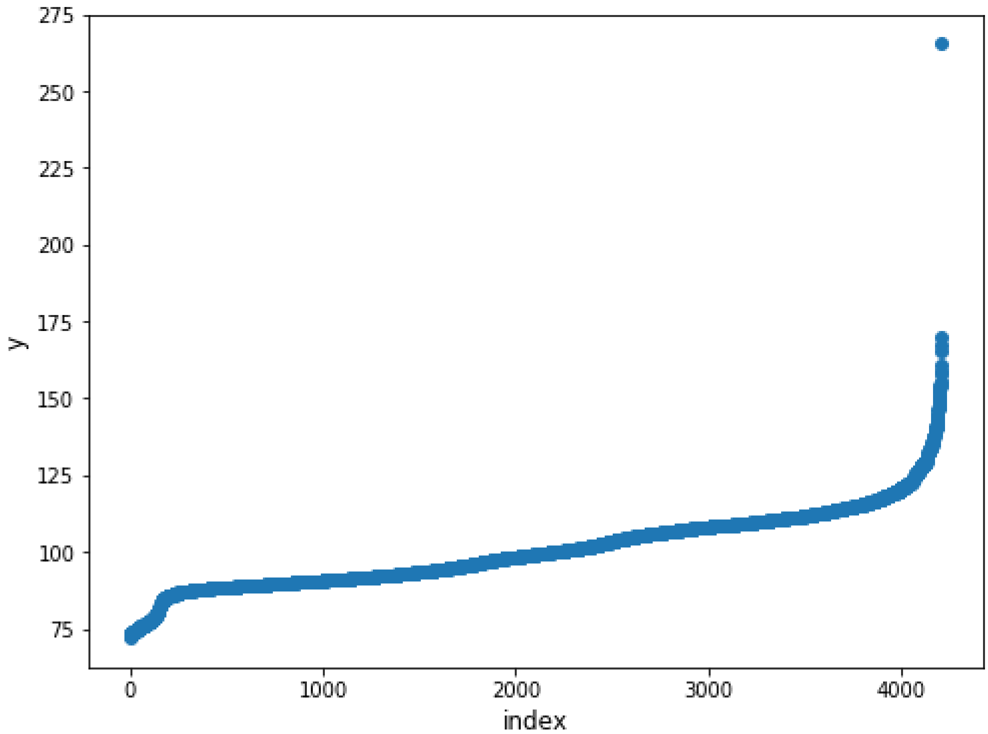
In [7]:
```
#calling functions to check missing values on training and test datasets
check_missing_values(train_df)
check_missing_values(test_df)
```

```
There are no missing values in the data
There are no missing values in the data
```

3.3 Ploting

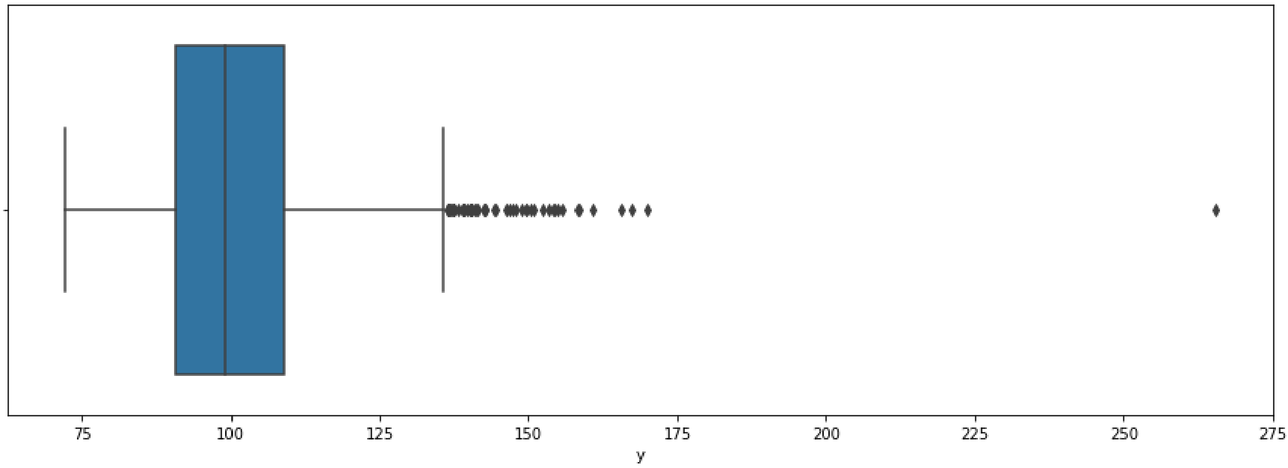
3.3.1 Ploting y values

In [8]:
```
#we are checking 'y' column
plt.figure(figsize=(8,6))
plt.scatter(range(train_df.shape[0]), np.sort(train_df.y.values))
plt.xlabel('index', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.show()

"""here we have observed 1 outlier at apporx 260"""
```



Out[8]: 'here we have observed 1 outlier at apporx 260'

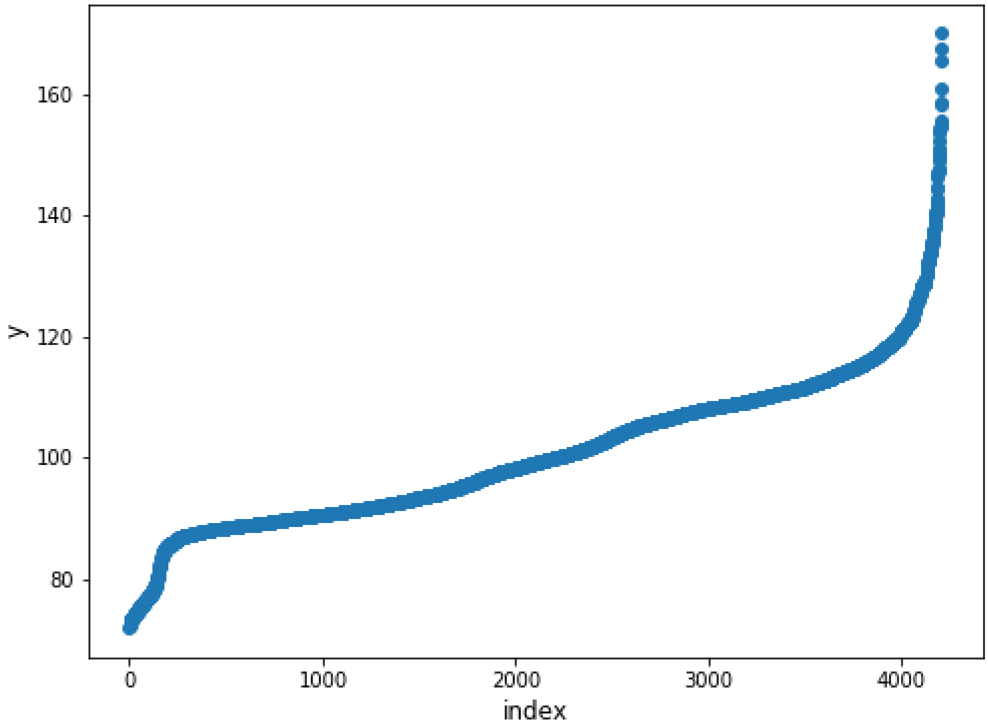
In [9]:
```
# we again check by visualising in BoxPlot

plt.figure(figsize=(15,5))
sns.boxplot(train_df.loc[:,'y'])
plt.show()
```



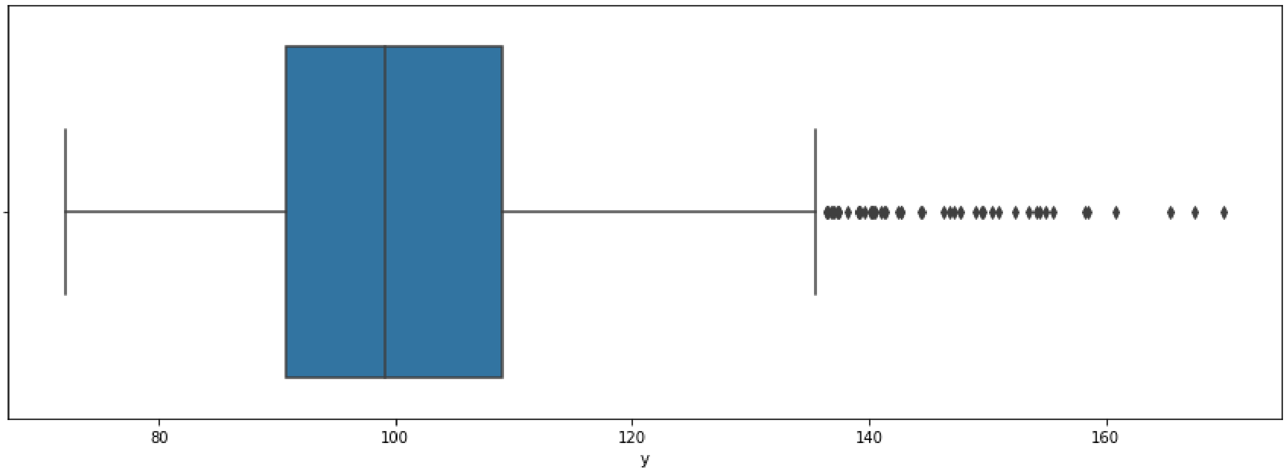
here we have observed 1 outlier

In [10]:
```
# we need to remove that outlier
# i have used zscore method and set threshold 10 acc to our data
# https://www.geeksforgeeks.org/scipy-stats-zscore-function-python/

train_df['x'] = np.abs(stats.zscore(train_df.loc[:,'y']))

outlier_ids = train_df[train_df['x']>10].ID

train_df_final = train_df[~train_df['ID'].isin(list(outlier_ids))]
```

In [11]:
```
#now plotting again without outlier
#we are checking 'y' column
plt.figure(figsize=(8,6))
plt.scatter(range(train_df_final.shape[0]), np.sort(train_df_final.y.values))
plt.xlabel('index', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.show()
```

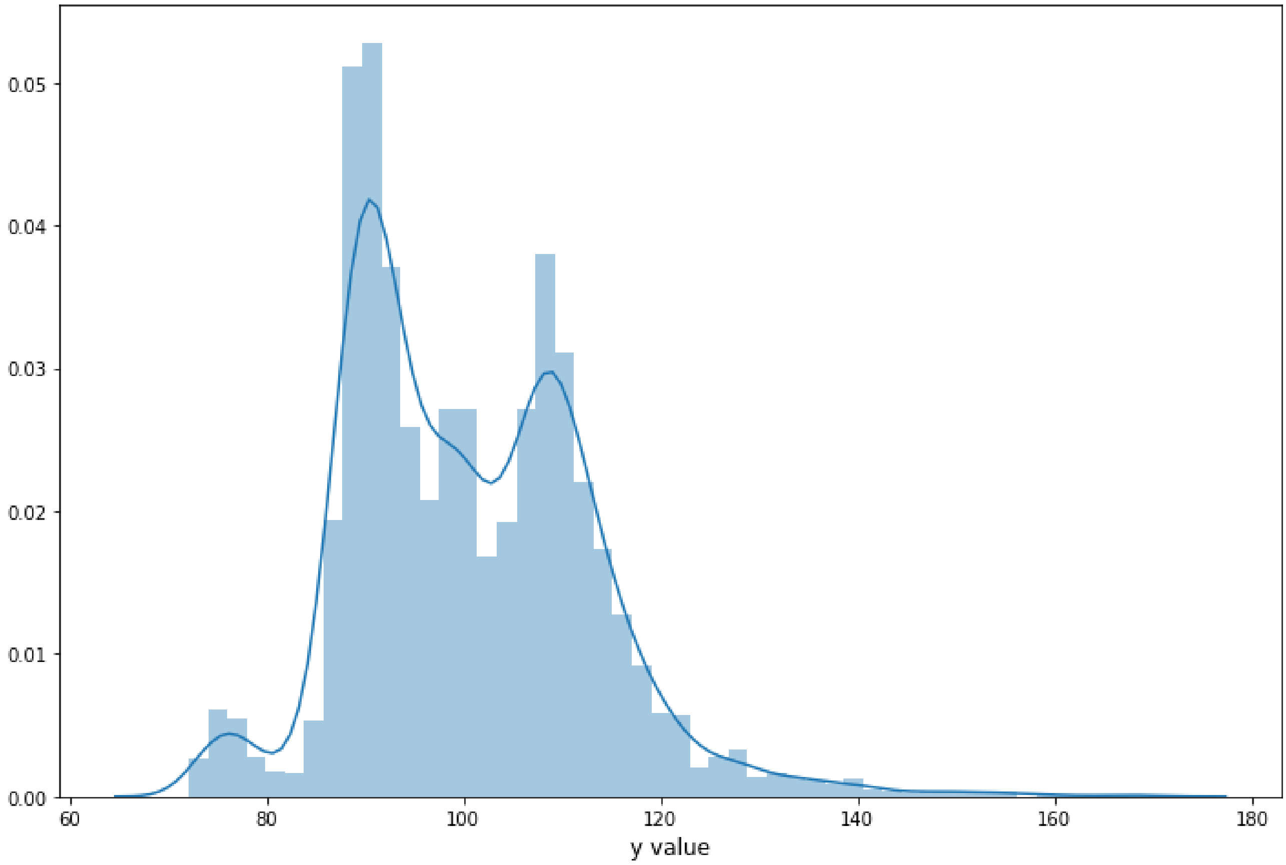


In [12]:
```
# we again check by visualising in BoxPlot

plt.figure(figsize=(15,5))
sns.boxplot(train_df_final.loc[:,'y'])
plt.show()
```



3.3.2 Plotting y distribution graph.

In [13]:
```
ulimit = 180# we have taken 180 data points
train_df_final['y'].ix[train_df_final['y']>ulimit] = ulimit

plt.figure(figsize=(12,8))#plot size
sns.distplot(train_df_final.y.values, bins=50, kde=True)
plt.xlabel('y value', fontsize=12)
plt.show()
```



In [14]:
```
#removing that x helper row for outlier from main row

train_df_final = train_df_final.drop(["x"], axis=1)
```

3.3.3 Data type of all the variables present in the dataset.

In [15]:
```
dtype_df = train_df_final.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
dtype_df.groupby("Column Type").aggregate('count').reset_index()
```

Out[15]:

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

Maximum of the columns are integers.
8 categorical columns.
1 float column (target variable) i.e. 'y'

In [16]:
```
#here we can see their types
dtype_df.ix[:15,:]
```

Out[16]:

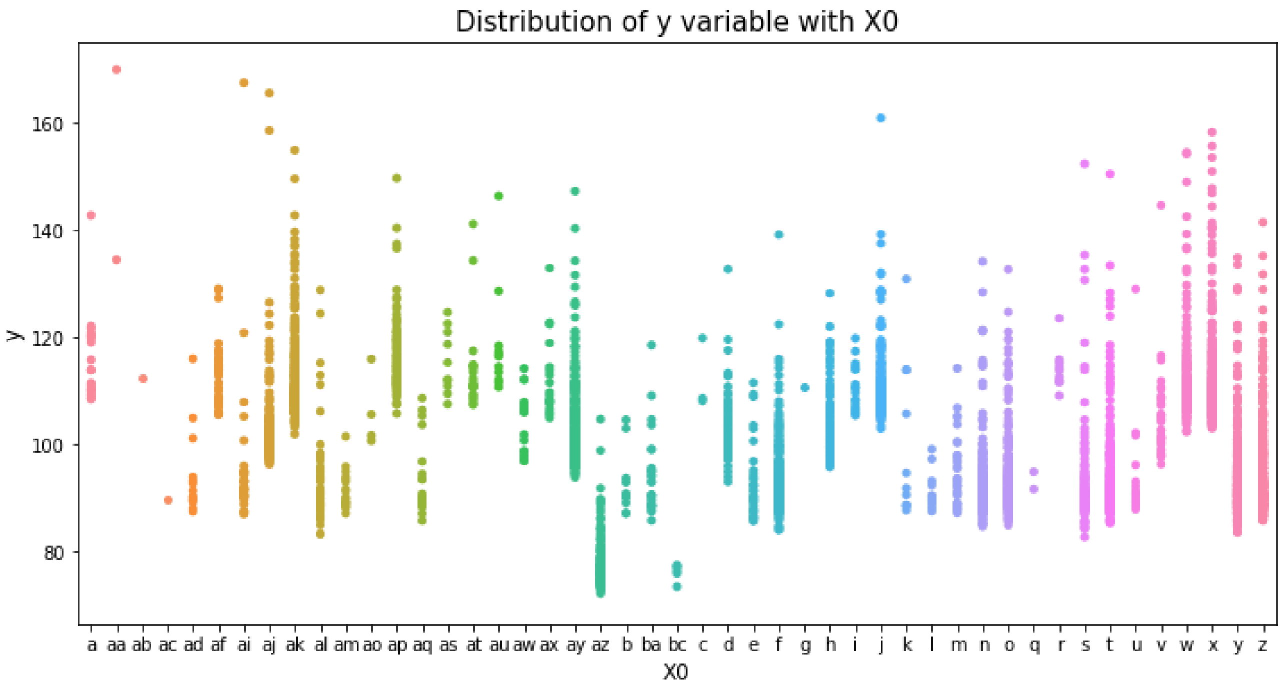
	Count	Column Type
0	ID	int64
1	y	float64
2	X0	object
3	X1	object
4	X2	object
5	X3	object
6	X4	object
7	X5	object
8	X6	object
9	X8	object
10	X10	int64
11	X11	int64
12	X12	int64
13	X13	int64
14	X14	int64
15	X15	int64

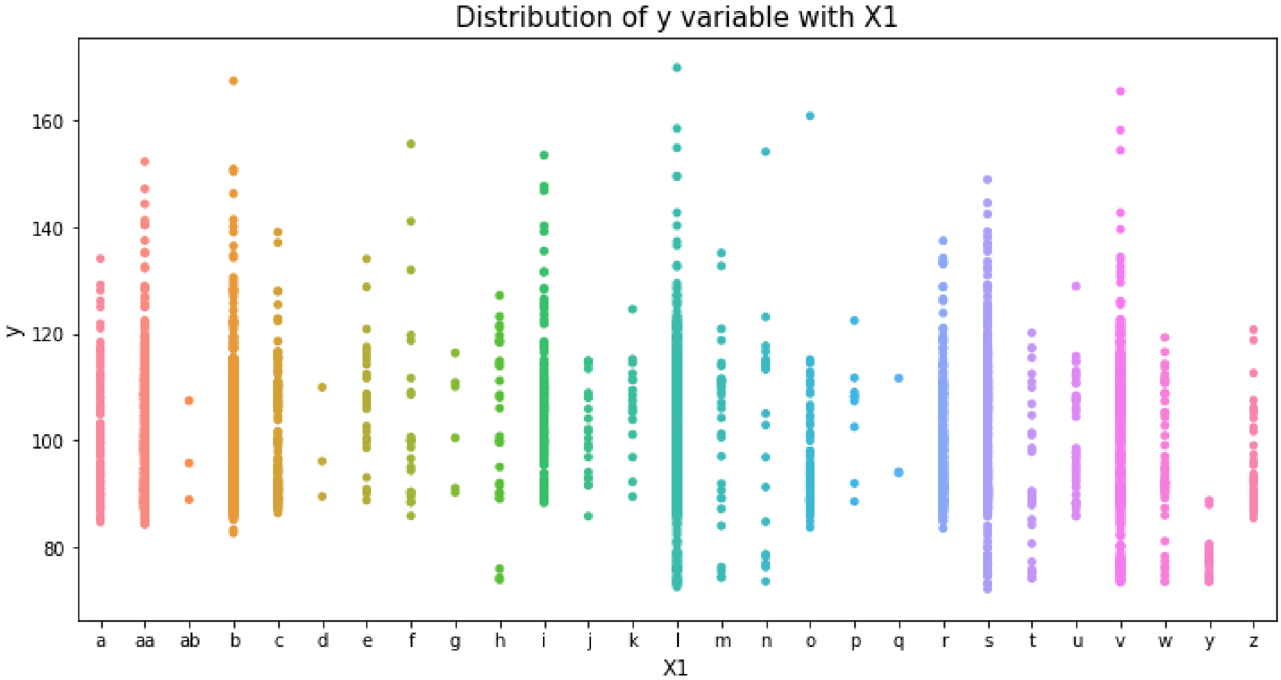
X0 to X8 are the categorical columns

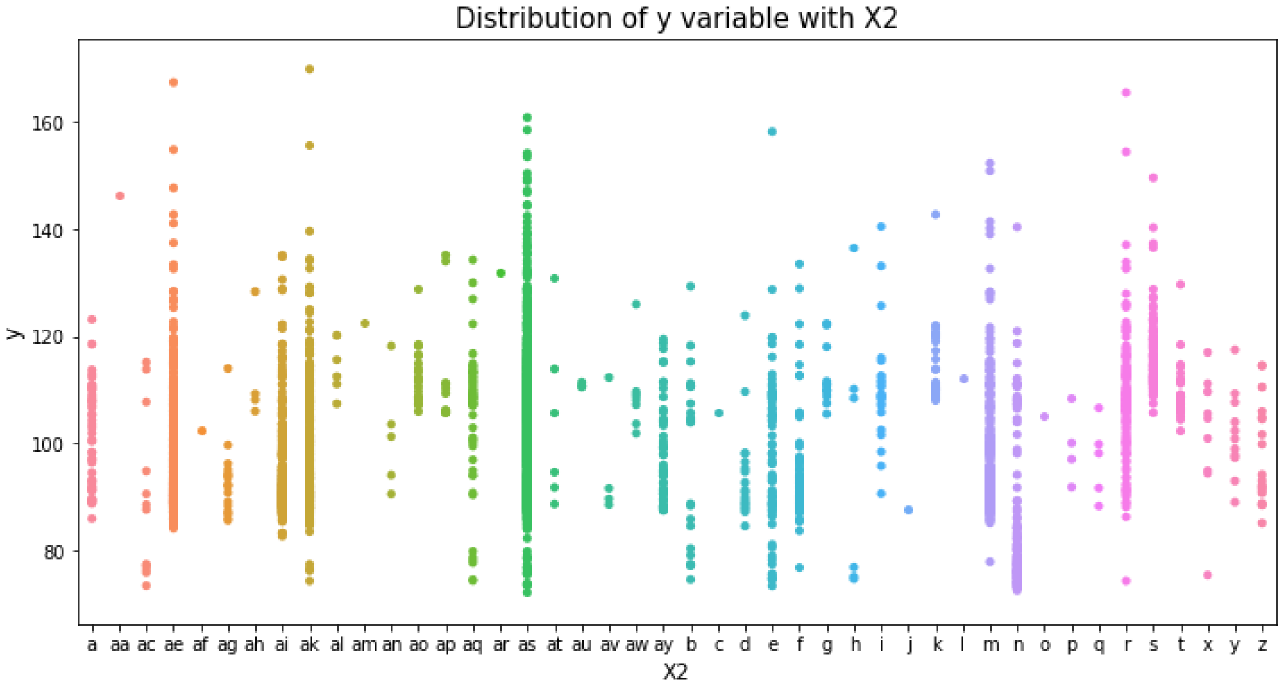
3.4 Plotting these categorical Values

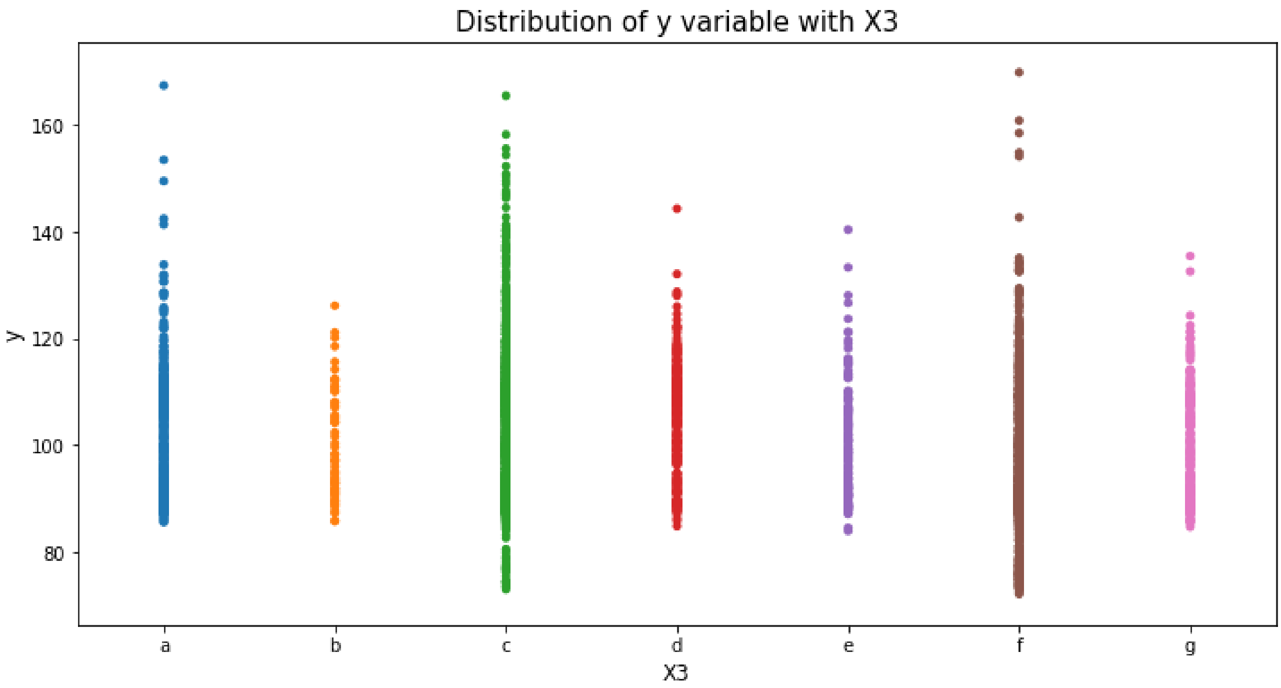
In [17]:

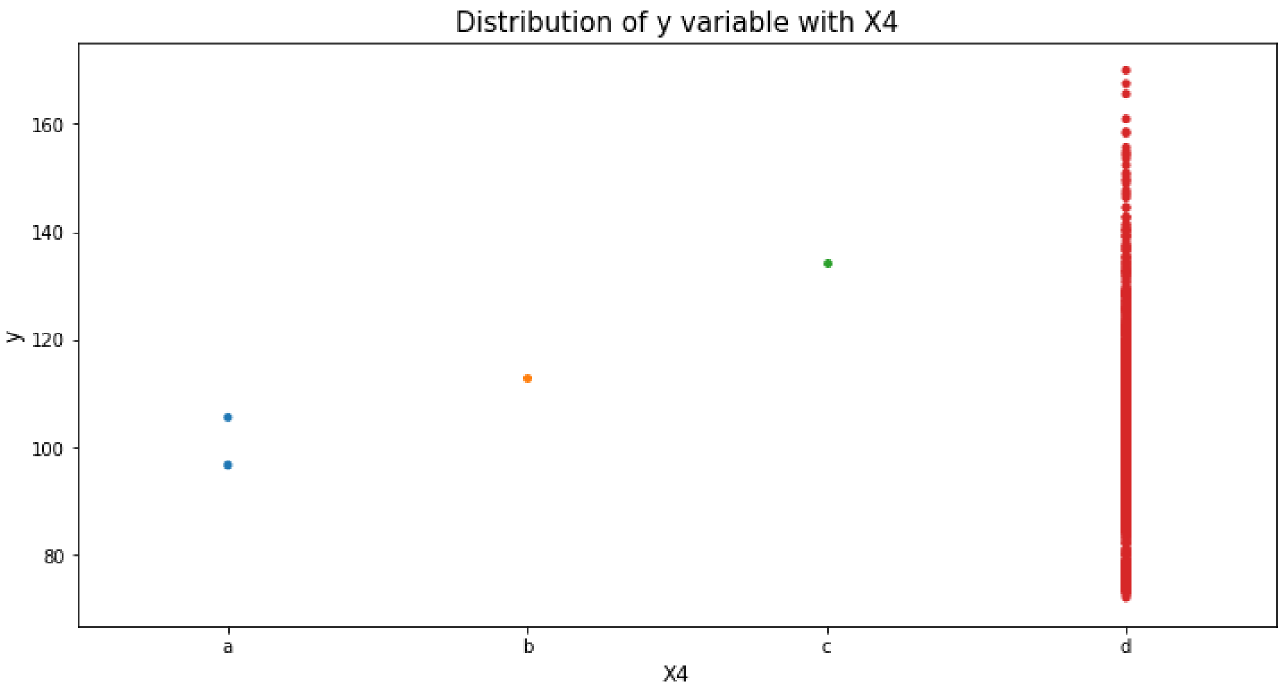
```
#https://www.kaggle.com/sudalairajkumar/simple-exploration-notebook-mercedes
var_name = ['X0','X1','X2','X3','X4','X5','X6','X8']
for val in var_name:
    col_order = np.sort(train_df_final[val].unique()).tolist()
    plt.figure(figsize=(12,6))
    sns.stripplot(x=val, y='y', data=train_df_final, order=col_order)
    plt.xlabel(val, fontsize=12)
    plt.ylabel('y', fontsize=12)
    plt.title("Distribution of y variable with "+val, fontsize=15)
    plt.show()
```

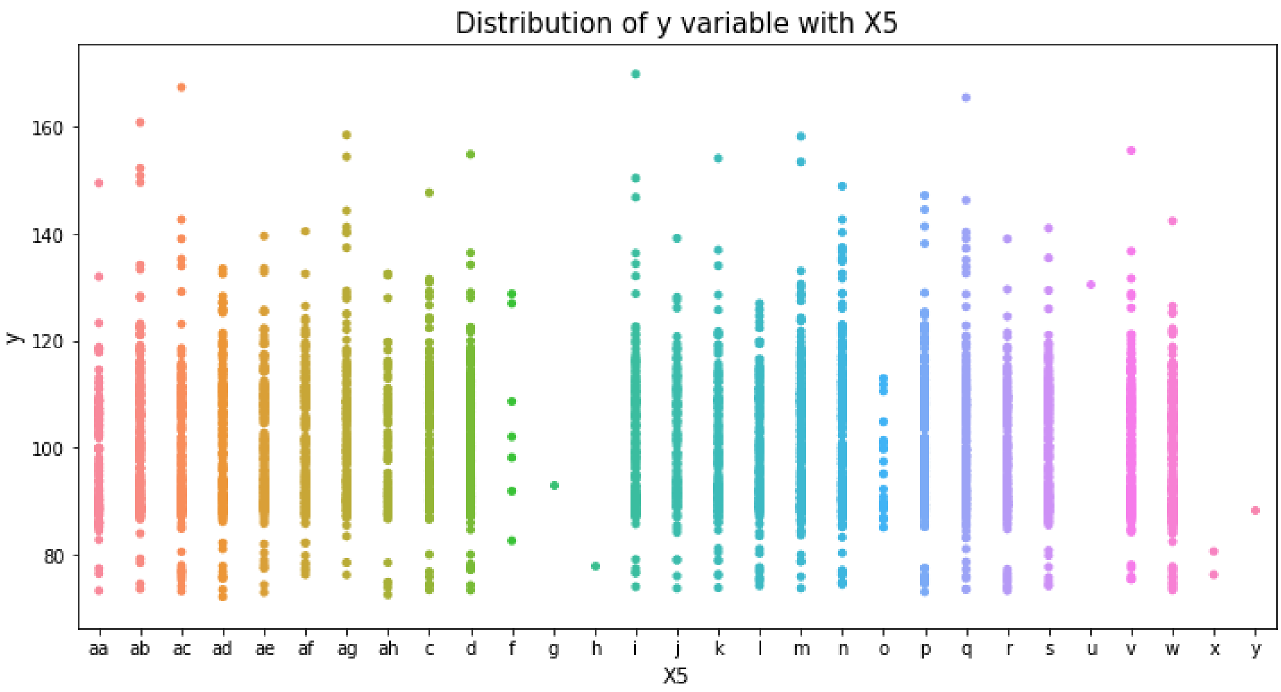

Distribution of y variable with X0
y
X0

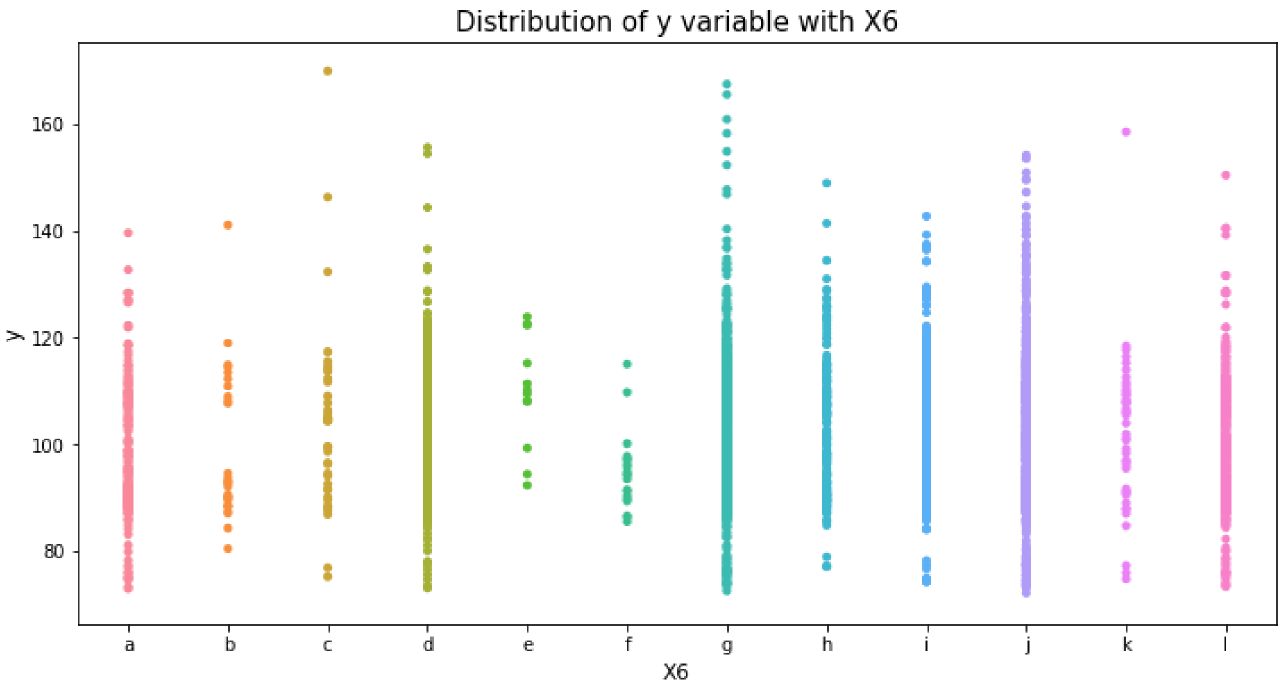

Distribution of y variable with X1
y
X1

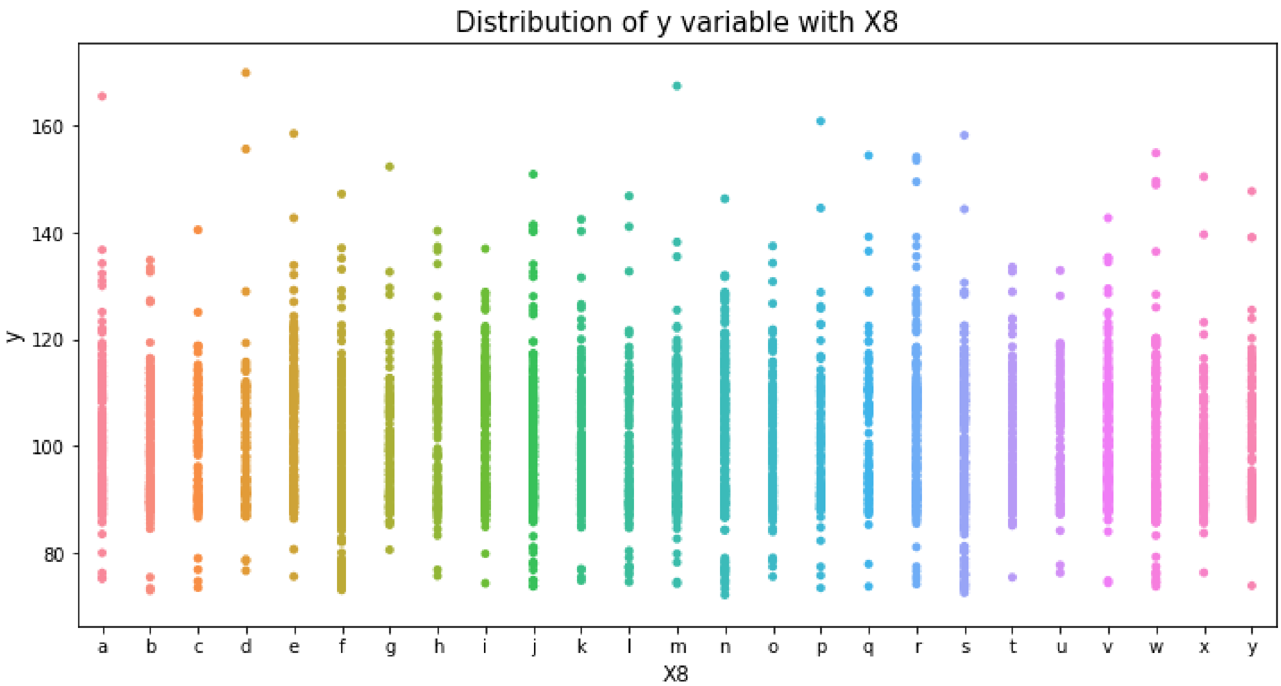

Distribution of y variable with X2
y
X2


Distribution of y variable with X3
y
160
140
120
100
80
a
b
c
d
e
f
g
X3


Distribution of y variable with X4
y
160
140
120
100
80
a
b
c
d
X4


Distribution of y variable with X5
y
160
140
120
100
80
aa ab ac ad ae af ag ah c d f g h i j k l m n o p q r s u v w x y
X5





Observation:

- we have observed that X0, X1, X2, X5, X6 and X8 have larger data point.
- X4 and X3 have lesser data point.

4. Machine Learning Models

4.1 Data preparation

In [182]:
```
train_df = pd.read_csv('train.csv')
print("Train shape : ", train_df.shape)
test_df = pd.read_csv('test.csv')
print("Train shape : ", test_df.shape)

y_train = train_df['y'].values
id_test = test_df['ID'].values

usable_columns = list(set(train_df.columns) - set(['ID', 'y']))#taking only important
coloumns
print(len(usable_columns))

x_train_final = train_df[usable_columns]
x_test_final = test_df[usable_columns]
```

```
Train shape :  (4209, 378)
Train shape :  (4209, 377)
376
```

In [183]:
```
# Converting training dataset object categorical values to numerical categorical type
s
#taken help from link: https://www.kaggle.com/anokas/mercedes-eda-xgboost-starter-0-5
5

for column in usable_columns:
    cardinality = len(np.unique(x_train_final[column]))

    if cardinality == 1:
        x_train_final.drop(column, axis=1) # Column with only one value is useless so
we drop it.
        x_test_final.drop(column, axis=1)

    if cardinality > 2: # Column is categorical.
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train_final[column] = x_train_final[column].apply(mapper)
        x_test_final[column] = x_test_final[column].apply(mapper)
```

In [184]:
```
# spiltting it into 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(x_train_final, y_train, test_size
=0.3, random_state=42)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2946, 376)
(1263, 376)
(2946,)
(1263,)
```

XGBoost

In [185]:
```
#taken help from kaggle discussion and kernels for xgboost
#setting up xtrain and xtrain

y_mean = y_train.mean()

d_train = xgb.DMatrix(X_train, label=y_train)
d_cvalid  = xgb.DMatrix(X_test, label=y_test)
d_test = xgb.DMatrix(x_test_final)#x_test_final
```

In [187]:
```
%%time
#xgb parameters
#just cross validation our model

params = {

    'eta': 0.005,
    'learning_rate': 0.008,
    'max_depth': 4,
    'subsample': 0.9,
    'objective': 'reg:linear',
    'n_estimators': 687,
    'eval_metric': 'rmse',
    'base_score': y_mean, # base prediction = mean(target)
    'silent': 1
}

num_boost_round=2000

#Cross Validation of XGBoost
cv_result = xgb.cv(params,
                   d_train,
                   num_boost_round,
                   nfold = 3,
                   early_stopping_rounds=50,
                   feval=r2_score_metric,#here we have used our metric method
                   verbose_eval=100,
                   show_stdv=False
                  )
```

```
[0]     train-r2:0.00950233     train-rmse:12.3021      test-r2:0.00846467      test
-rmse:12.3007
[100]   train-r2:0.502522       train-rmse:8.71869      test-r2:0.478873        test
-rmse:8.9194
[200]   train-r2:0.61252        train-rmse:7.69449      test-r2:0.571961        test
-rmse:8.08328
[300]   train-r2:0.647986       train-rmse:7.33373      test-r2:0.588661        test
-rmse:7.92327
[400]   train-r2:0.66697        train-rmse:7.13309      test-r2:0.591183        test
-rmse:7.89845
Wall time: 36.1 s
```

best cv_result is: train-r2:0.66697 test-r2:50.91183

In [97]:
```
%%time
#Training the model
#taken help from link: https://www.kaggle.com/anokas/mercedes-eda-xgboost-starter-0-5
5

#model = joblib.load('model_xgb.pkl')#from load

watchlist = [(d_train, 'train'), (d_cvalid, 'valid')]

model = xgb.train(params, d_train , num_boost_round, watchlist, early_stopping_rounds
=60,
                  feval=r2_score_metric, maximize=True, verbose_eval=10)

#joblib.dump(model, 'model_xgb.pkl')#to load
```

```
[0]	train-rmse:12.3026	valid-rmse:13.333	train-r2:0.009819	vali
d-r2:0.005644
Multiple eval metrics have been passed: 'valid-r2' will be used for early stopping.

Will train until valid-r2 hasn't improved in 60 rounds.
[10]	train-rmse:11.7348	valid-rmse:12.8304	train-r2:0.099115	vali
d-r2:0.079199
[20]	train-rmse:11.2266	valid-rmse:12.3849	train-r2:0.17545	vali
d-r2:0.142034
[30]	train-rmse:10.7723	valid-rmse:11.9932	train-r2:0.240832	vali
d-r2:0.195436
[40]	train-rmse:10.3694	valid-rmse:11.6509	train-r2:0.29656	vali
d-r2:0.240714
[50]	train-rmse:10.0114	valid-rmse:11.3498	train-r2:0.344291	vali
d-r2:0.279445
[60]	train-rmse:9.69459	valid-rmse:11.0879	train-r2:0.385134	vali
d-r2:0.312319
[70]	train-rmse:9.41517	valid-rmse:10.8607	train-r2:0.420067	vali
d-r2:0.340208
[80]	train-rmse:9.16928	valid-rmse:10.6662	train-r2:0.449963	vali
d-r2:0.363631
[90]	train-rmse:8.95304	valid-rmse:10.4979	train-r2:0.4756 valid-r2:0.3
8356
[100]	train-rmse:8.76198	valid-rmse:10.3513	train-r2:0.497743	vali
d-r2:0.400651
[110]	train-rmse:8.59697	valid-rmse:10.2279	train-r2:0.516483	vali
d-r2:0.41486
[120]	train-rmse:8.45165	valid-rmse:10.1224	train-r2:0.532691	vali
d-r2:0.426872
[130]	train-rmse:8.32342	valid-rmse:10.0302	train-r2:0.546763	vali
d-r2:0.437258
[140]	train-rmse:8.21265	valid-rmse:9.95325	train-r2:0.558747	vali
d-r2:0.445863
[150]	train-rmse:8.11541	valid-rmse:9.88898	train-r2:0.569134	vali
d-r2:0.452995
[160]	train-rmse:8.03097	valid-rmse:9.83332	train-r2:0.578054	vali
d-r2:0.459136
[170]	train-rmse:7.95615	valid-rmse:9.78744	train-r2:0.585879	vali
d-r2:0.464171
[180]	train-rmse:7.89008	valid-rmse:9.74714	train-r2:0.592729	vali
d-r2:0.468575
[190]	train-rmse:7.83219	valid-rmse:9.71435	train-r2:0.598683	vali
d-r2:0.472144
[200]	train-rmse:7.77968	valid-rmse:9.68679	train-r2:0.604046	vali
d-r2:0.475135
[210]	train-rmse:7.73171	valid-rmse:9.66645	train-r2:0.608913	vali
d-r2:0.477336
[220]	train-rmse:7.6894	valid-rmse:9.64917	train-r2:0.613183	vali
d-r2:0.479204
[230]	train-rmse:7.65263	valid-rmse:9.63488	train-r2:0.616873	vali
d-r2:0.480745
[240]	train-rmse:7.6195	valid-rmse:9.62354	train-r2:0.620183	vali
d-r2:0.481967
[250]	train-rmse:7.58922	valid-rmse:9.61348	train-r2:0.623196	vali
d-r2:0.483049
[260]	train-rmse:7.56142	valid-rmse:9.60584	train-r2:0.625952	vali
d-r2:0.48387
[270]	train-rmse:7.5356	valid-rmse:9.60165	train-r2:0.628502	vali
d-r2:0.484321
[280]	train-rmse:7.50979	valid-rmse:9.59848	train-r2:0.631042	vali
d-r2:0.484661
[290]	train-rmse:7.48632	valid-rmse:9.59509	train-r2:0.633344	vali
d-r2:0.485025
[300]	train-rmse:7.4664	valid-rmse:9.59175	train-r2:0.635293	vali
d-r2:0.485383
```

```
[310]	train-rmse:7.44628	valid-rmse:9.59189	train-r2:0.637256	vali
d-r2:0.485369
[320]	train-rmse:7.42812	valid-rmse:9.5908	train-r2:0.639024	vali
d-r2:0.485486
[330]	train-rmse:7.41023	valid-rmse:9.59082	train-r2:0.64076	vali
d-r2:0.485483
[340]	train-rmse:7.39496	valid-rmse:9.59082	train-r2:0.642239	vali
d-r2:0.485483
[350]	train-rmse:7.38103	valid-rmse:9.59077	train-r2:0.643586	vali
d-r2:0.485488
[360]	train-rmse:7.36523	valid-rmse:9.59208	train-r2:0.64511	vali
d-r2:0.485348
[370]	train-rmse:7.35261	valid-rmse:9.59335	train-r2:0.646325	vali
d-r2:0.485212
[380]	train-rmse:7.33914	valid-rmse:9.59447	train-r2:0.64762	vali
d-r2:0.485092
Stopping. Best iteration:
[327]	train-rmse:7.41638	valid-rmse:9.59026	train-r2:0.640164	vali
d-r2:0.485543

Wall time: 12.5 s
```

In [93]:
```
# Predict on test

#y_pred = model.predict(d_test)
```

In [54]:
```
#creating stacked model
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.h
tml
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingR
egressor.html
#here we have used GradientBoostingRegressor form ensemble and cross validation Lasso
LarsCV.
stack_model= make_pipeline(

   StackingEstimator(estimator=LassoLarsCV(normalize=True)),
   StackingEstimator(estimator=GradientBoostingRegressor( learning_rate=0.001,
                                                         loss="huber",
                                                         max_depth=6,
                                                         max_features=0.6,
                                                         min_samples_leaf=18,
                                                         min_samples_split=16,
                                                         subsample=0.7)),
    LassoLarsCV()
)
```

In [161]:
```
%%time
stack_model.fit(X_train,y_train)
```

```
Wall time: 9.25 s
```

Out[161]:
```
Pipeline(memory=None,
     steps=[('stackingestimator-1', StackingEstimator(estimator=LassoLarsCV(copy_X=T
rue, cv=None, eps=2.220446049250313e-16,
      fit_intercept=True, max_iter=500, max_n_alphas=1000, n_jobs=1,
      normalize=True, positive=False, precompute='auto', verbose=False))), ('stackin
gestimator-2', StackingEsti...x_n_alphas=1000, n_jobs=1,
      normalize=True, positive=False, precompute='auto', verbose=False))])
```

In [175]:
```
#final_results=stack_model.predict(x_test_final)
```

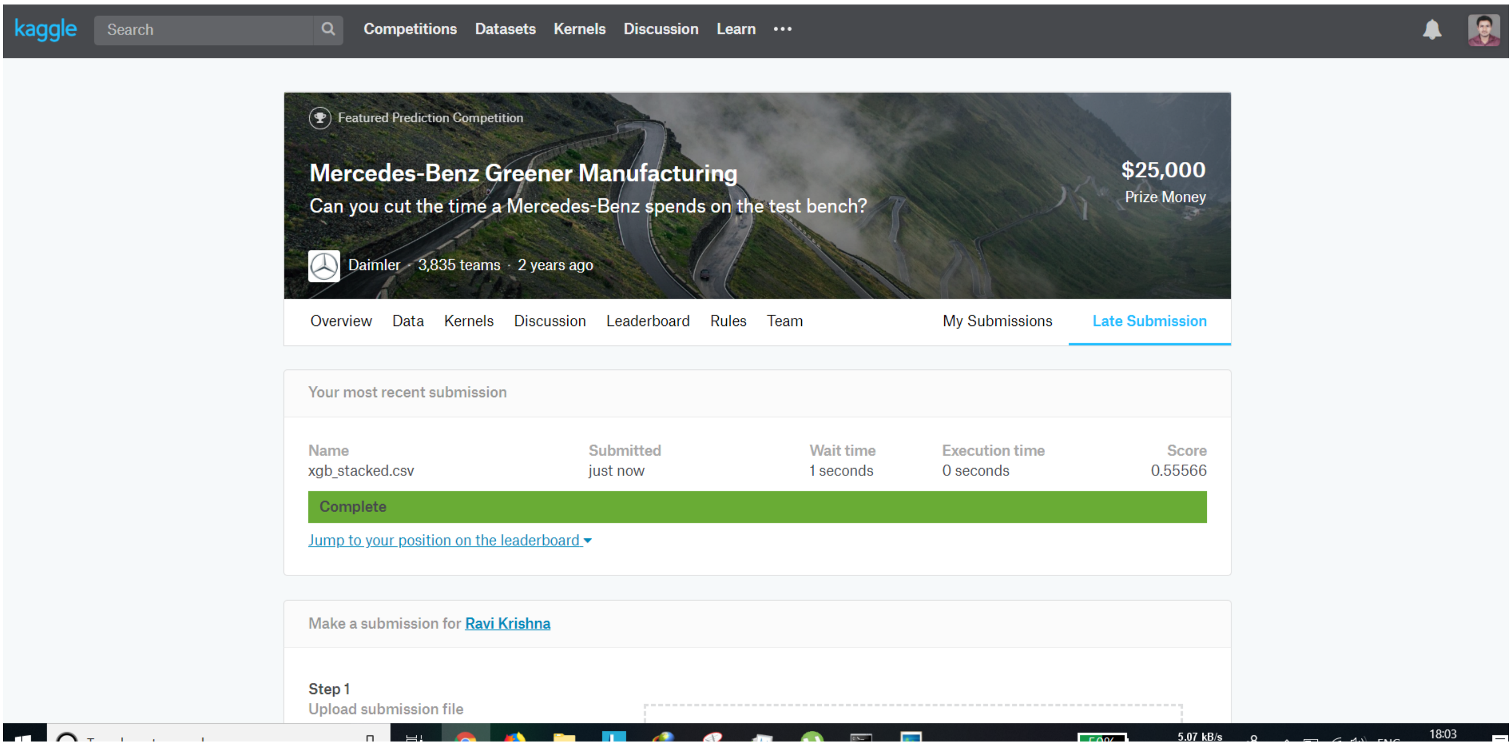
In [176]:
```
#final_results=model.predict(d_test)
```

In [177]:
```
# Predicting R2SCORE
#from sklearn.metrics import r2_score
#r2_score = r2_score(y_test, final_results)#taking r2score on traing data
##print('r2_score = ',r2_score)
#clf = RandomForestRegressor(n_estimators = 60 ,max_depth=5,oob_score=True)
```

In [ ]:
```
#predicting on test dataset

final_results=0.10 * model.predict(d_test) + 0.90 * stack_model.predict(x_test_final)
```

In [188]:
```
#exporting final results into csv file
test_df = pd.read_csv('test.csv')
csvfile = pd.DataFrame()
csvfile['ID'] = test_df['ID']
csvfile['y'] = final_results
csvfile.to_csv('xgb_stacked.csv', index=False)
```

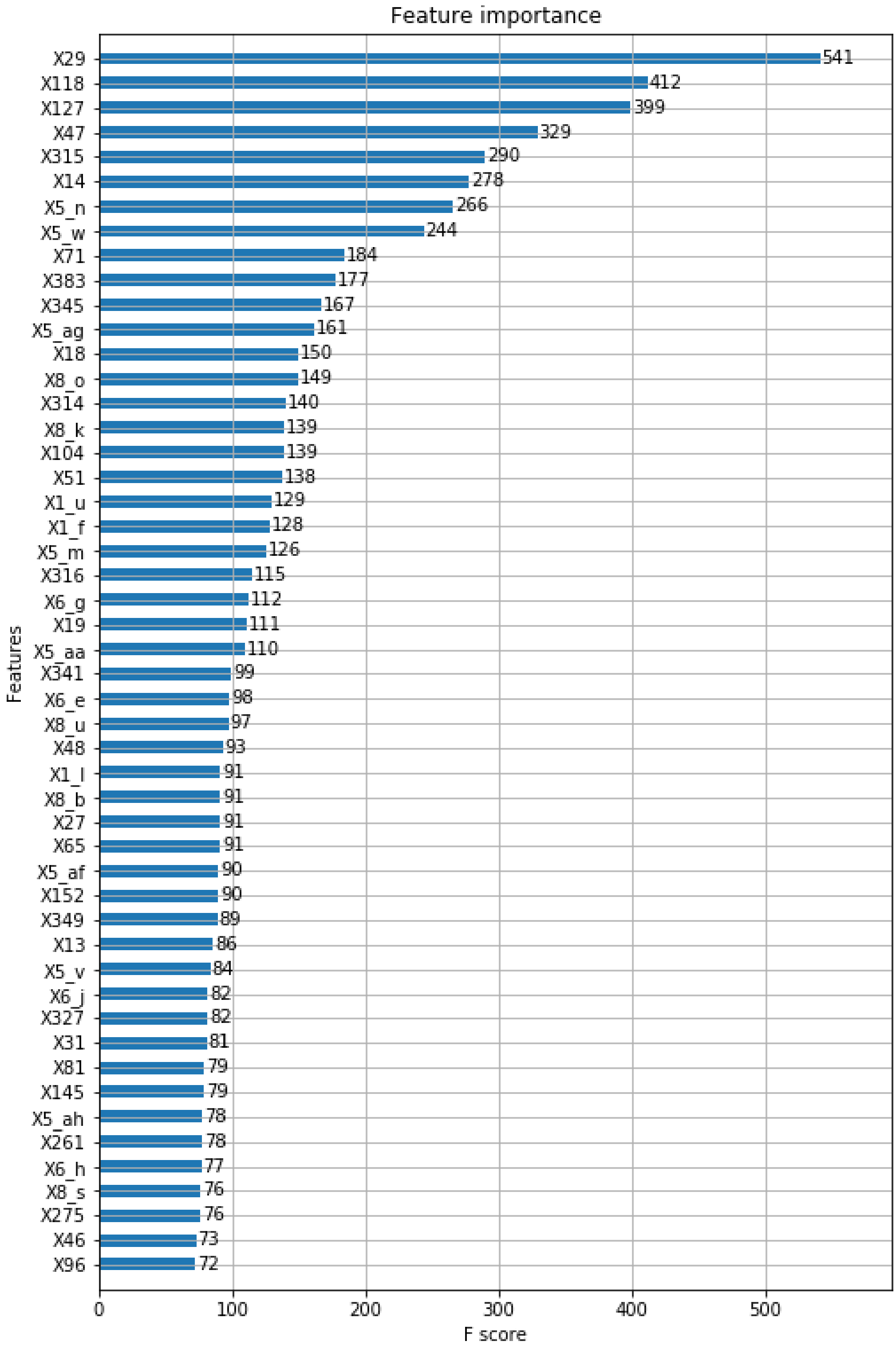
Final Test result given by kaggle is: 0.55566



Feature Importance

In [76]:
```
#https://www.kaggle.com/satadru5/mercedes-benz-xgb-modeling-lb-score-0-54472
fig, ax = plt.subplots(1, 1, figsize=(8, 13))
xgb.plot_importance(model, max_num_features=50, height=0.5, ax=ax)
```

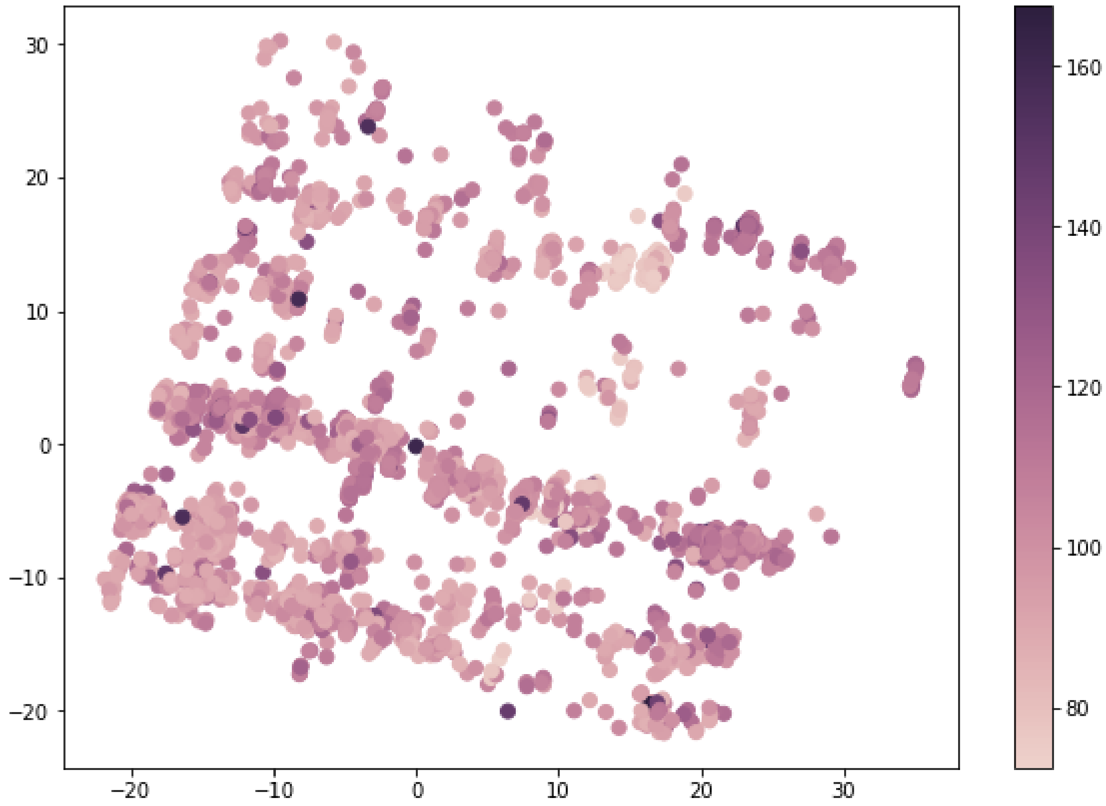
Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5e09445d30>



4.2 PCA - Principal component analysis

In [0]:
```
# PCA Implementation
pca = PCA(n_components=2)
pca_data = pca.fit_transform(X_train)
```

In [0]:
```
cmap = sns.cubehelix_palette(as_cmap=True)
f, ax = plt.subplots(figsize=(10,7))
points = ax.scatter(pca_data[:,0], pca_data[:,1], c=y_train, s=50, cmap=cmap)
f.colorbar(points)
plt.show()
```

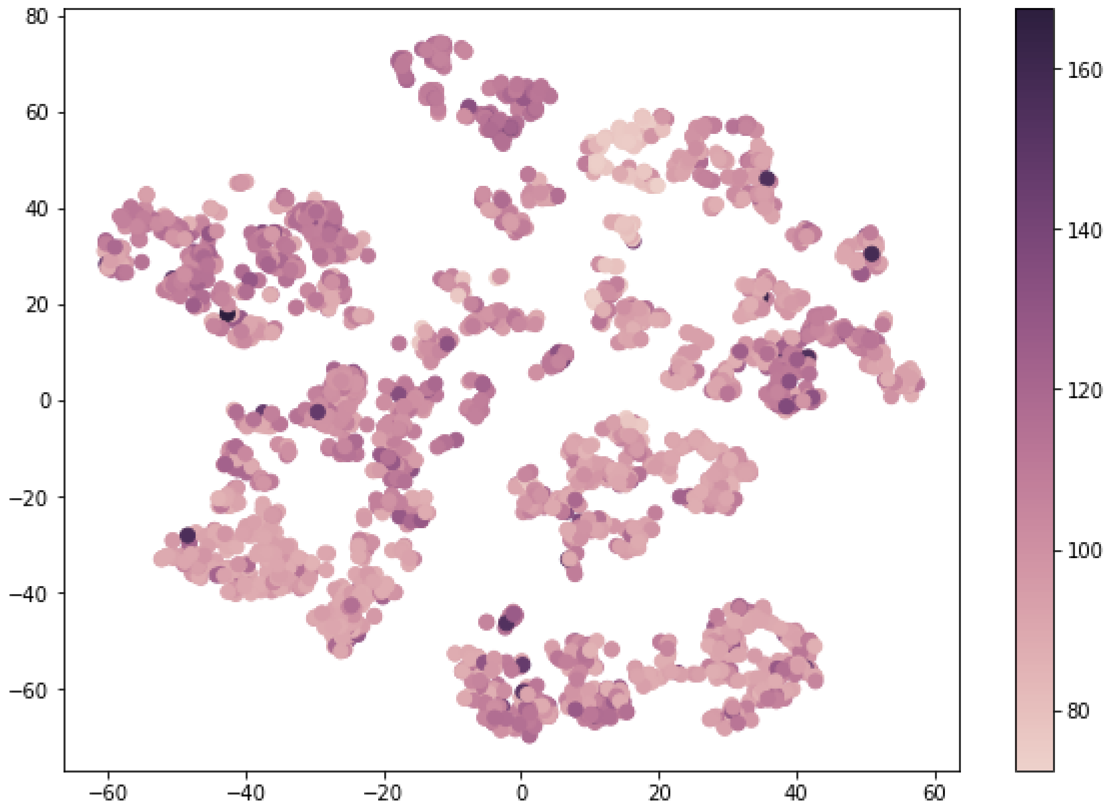


Observation: here we can see how PCA visualize all the data point far separated from each other, they are not forming tightly group.

4.3 T-SNE (t-distributed Stochastic Neighbor Embedding)

In [0]:
```
# TSNE Implementation
model = TSNE(n_components=2,random_state=0,perplexity=30)

tsne_data = model.fit_transform(X_train)
```

In [0]:
```
cmap = sns.cubehelix_palette(as_cmap=True)
f, ax = plt.subplots(figsize=(10,7))
points = ax.scatter(tsne_data[:,0], tsne_data[:,1], c=y_train, s=50, cmap=cmap)
f.colorbar(points)
plt.show()
```



Observation: here we can see how TSNE visualize all the data point closly attached from each other, They are well grouped .

4.4 K-Nearest Neighbors Regressor

In [0]:
```
#KNN implementation
#biulding model

knn = KNeighborsRegressor(n_neighbors=5)#k=5 gives best results

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

r2_score_knn = round(r2_score(y_test, y_pred),3)#taking r2score
accuracy = round(knn.score(X_train, y_train) *100,2)#taking accuracy

results = {'r2_score':r2_score_knn, 'accuracy':accuracy}
print (results)
```

```
{'r2_score': 0.414, 'accuracy': 65.57}
```

4.5 Support Vector Regressor

In [0]:

```
#SVR implementation
from sklearn.metrics import r2_score
clf = SVR()

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

r2_score = round(r2_score(y_test, y_pred),3)#taking r2score
accuracy = round(clf.score(X_train, y_train) * 100, 2)

results = {'r2_score':r2_score, 'accuracy':accuracy}
print(results)
```

```
{'r2_score': 0.384, 'accuracy': 44.76}
```

4.6 Random Forest Regressor

In [0]:

```
#RFR implementation
from sklearn.metrics import r2_score
clf = RandomForestRegressor(n_estimators = 60 ,max_depth=5,oob_score=True)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

r2_score = round(r2_score(y_test, y_pred),3)#taking r2score
accuracy = round(clf.score(X_train, y_train) * 100, 2)

results = {'r2_score':r2_score, 'accuracy':accuracy}
print (results)
```

```
{'r2_score': 0.533, 'accuracy': 65.66}
```

4.7 Linear Regression

In [143]:

```
#Linear Regression implementation
from sklearn.metrics import r2_score
clf = LinearRegression()

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

r2_score = round(r2_score(y_test, y_pred),3)#taking r2score
accuracy = round(clf.score(X_train, y_train) * 100, 2)

results = {'r2_score':r2_score, 'accuracy':accuracy}
print (results)
```

```
{'r2_score': 0.439, 'accuracy': 63.06}
```

5. Conclution

S.no	Model Algo	R2 Score	Accuracy
1.	K-Nearest Neighbors Regressor	0.414	65.57
2.	Support Vector Regressor	0.384	44.76
3.	Random Forest Regressor	0.533	65.66
3.	Linear Regression	0.439	63.06
4.	XGBoost-Stacked	0.55566	-

Here we can see from conclution that

1. XGBoost-Stacked perform best in this case by obtaining R2_SCORE =0.55566 ¶

2.Next Our RANDOM FOREST REGRESSOR also perform well with nearly R2_SCORE=0.533

3. After that Knn does better.

4. Linear Regression model not suit for this type of problem.

---xxx---