Mercedes-Benz Greener Manufacturing Case Study

Data Source: [https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data (https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data)](https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/data)

Problem Statment: Can you cut the time a Mercedes-Benz spends on the test bench?

1. 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.
2. 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.
3. In this competition, Daimler is challenging to reduce the time that cars spend on the test bench.

Data Description:

1. 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.
2. The ground truth is labeled 'y' and represents the time (in seconds) that the car took to pass testing for each variable.
3. File descriptions: Variables with letters are categorical. Variables with 0/1 are binary values.
4. train.csv - the training set
5. test.csv - the test set, you must predict the 'y' variable for the 'ID's in this file
6. sample_submission.csv - a sample submission file in the correct format

Data Analysis

In [1]:
```
# Importing all necessary modules.
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
from sklearn.base import BaseEstimator,TransformerMixin,ClassifierMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, FastICA
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import GaussianRandomProjection
from sklearn.random_projection import SparseRandomProjection
from sklearn.linear_model import ElasticNetCV, LassoLarsCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.pipeline import make_pipeline, make_union
from sklearn.utils import check_array
from sklearn.metrics import r2_score
# keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, BatchNormalization, Activation
from keras.wrappers.scikit_learn import KerasRegressor
from keras.callbacks import EarlyStopping, ModelCheckpoint

# model evaluation
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.feature_selection import SelectFromModel

# To make Results reproducible
seed = 40

import warnings
warnings.filterwarnings('ignore')
color = sns.color_palette()
%matplotlib inline
```

```
Using TensorFlow backend.
```

The number of rows are small with 388 columns. So We should be take care about not to overfit.

In [2]:
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
print("Train shape : ", train_df.shape)
print("Test shape : ", test_df.shape)
```

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

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

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	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
3	9	80.62	az	t	n	f	d	x	l	e	...	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

5 rows × 378 columns

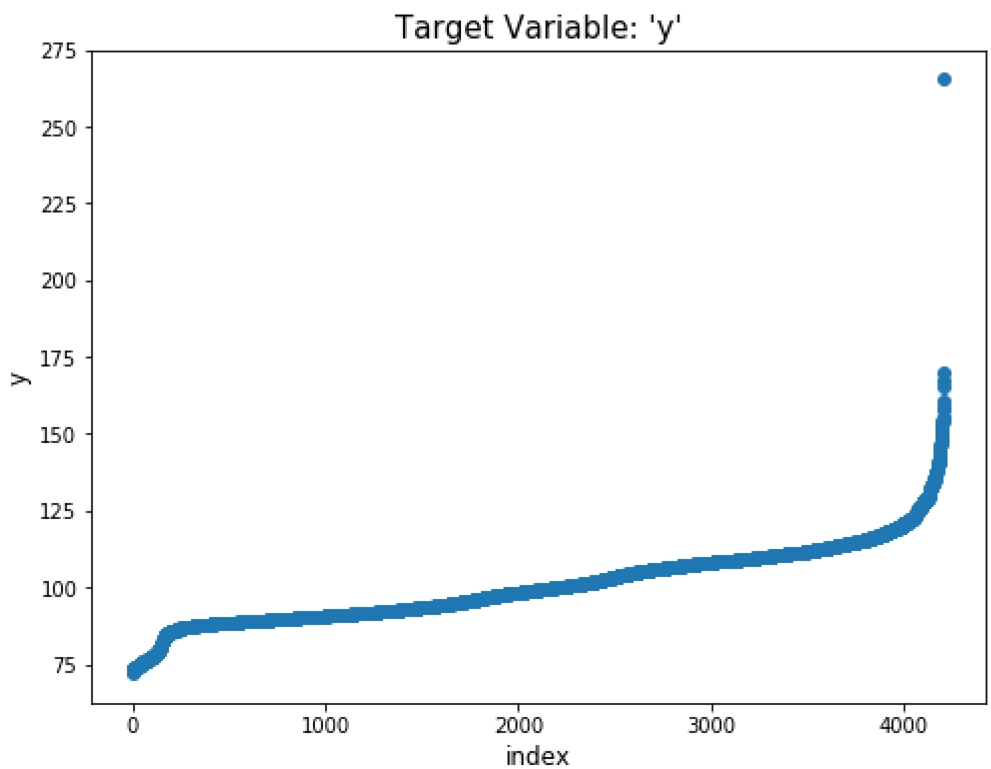
Features of Data:

1. **ID** : ID column of data.
2. **y** : Target Variable.
3. **X0-X385** : Data columns.

Target Variable(y):

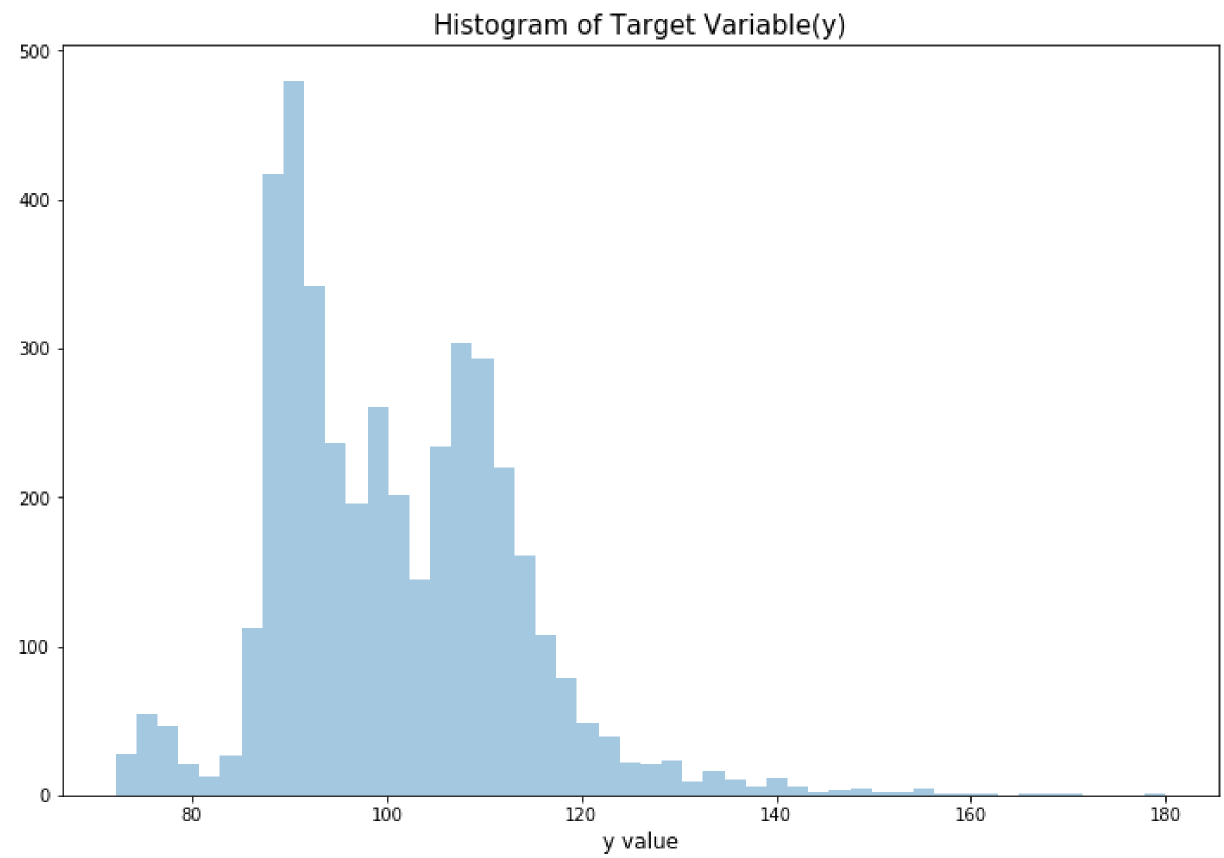
"y" is the variable we need to predict. So let us do perform analysis on this variable.

In [4]:
```
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.title("Target Variable: 'y'",fontsize=15)
plt.show()
```



Value of "y" is fairly spread across the range between 70-200

In [5]:
```
ulimit = 180
train_df['y'].ix[train_df['y']>ulimit] = ulimit

plt.figure(figsize=(12,8))
sns.distplot(train_df.y.values, bins=50, kde=False)
plt.xlabel('y value', fontsize=12)
plt.title("Histogram of Target Variable(y)",fontsize=15)
plt.show()
```



In [6]:
```
print('min: {} max: {} mean: {} std: {}'.format(min(train_df['y'].values), max(tr
print('Count of values above 180: {}'.format(np.sum(train_df['y'].values > 180)))
```

```
min: 72.11 max: 180.0 mean: 100.64904727963888 std: 12.481281731120474
Count of values above 180: 0
```

Observations of Target Variable y:

1. **We can observe that most of the values lies between 90-120. So avg production time is 90-120.**
2. So we have a pretty **standard distribution** here, which is centred around almost exactly 100.
3. The fact that ID is not equal to the row ID seems to suggest that the **train and test sets are randomly sampled**.

Variables/Feature Analysis.

In [7]:
```
dtype_df = train_df.dtypes.reset_index()
dtype_df.columns = ["Count", "Column Type"]
dtype_df.groupby("Column Type").aggregate('count').reset_index()
```

Out[7]:

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

So majority of the columns are integers with 8 categorical features and 1 float feature (target variable)

In [8]:
```
dtype_df.ix[:10,:]
```

Out[8]:

	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

X0 to X8 are the categorical columns.

Check for the missing values.

In [9]:
```
missing_df = train_df.isnull().sum(axis=0).reset_index()
missing_df.columns = ['column_name', 'missing_count']
missing_df = missing_df.ix[missing_df['missing_count']>0]
missing_df = missing_df.sort_values(by='missing_count')
missing_df
```

Out[9]:

column_name	missing_count

We don't have any missing values.

In [10]:
```
cols = [c for c in train_df.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))

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

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

Out[10]:
```
int64     368
object      8
dtype: int64
```

In [11]:
```
counts = [[], [], []]
for c in cols:
    typ = train_df[c].dtype
    uniq = len(np.unique(train_df[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'.form

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', 'X289', 'X29
0', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [12]:
```
unique_values_dict = {}
for col in train_df.columns:
    if col not in ["ID", "y", "X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        unique_value = str(np.sort(train_df[col].unique()).tolist())
        tlist = unique_values_dict.get(unique_value, [])
        tlist.append(col)
        unique_values_dict[unique_value] = tlist[:]
for unique_val, columns in unique_values_dict.items():
    print("Columns containing the unique values : ",unique_val)
    print(columns)
    print("--------------------------------------------------")
```

```
Columns containing the unique values :  [0, 1]
['X10', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20', 'X21',
'X22', 'X23', 'X24', 'X26', 'X27', 'X28', 'X29', 'X30', 'X31', 'X32', 'X33', 'X
34', 'X35', 'X36', 'X37', 'X38', 'X39', 'X40', 'X41', 'X42', 'X43', 'X44', 'X4
5', 'X46', 'X47', 'X48', 'X49', 'X50', 'X51', 'X52', 'X53', 'X54', 'X55', 'X5
6', 'X57', 'X58', 'X59', 'X60', 'X61', 'X62', 'X63', 'X64', 'X65', 'X66', 'X6
7', 'X68', 'X69', 'X70', 'X71', 'X73', 'X74', 'X75', 'X76', 'X77', 'X78', 'X7
9', 'X80', 'X81', 'X82', 'X83', 'X84', 'X85', 'X86', 'X87', 'X88', 'X89', 'X9
0', 'X91', 'X92', 'X94', 'X95', 'X96', 'X97', 'X98', 'X99', 'X100', 'X101', 'X1
02', 'X103', 'X104', 'X105', 'X106', 'X108', 'X109', 'X110', 'X111', 'X112', 'X
113', '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', 'X234', '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', 'X269',
'X270', 'X271', 'X272', 'X273', 'X274', 'X275', 'X276', 'X277', 'X278', 'X279',
'X280', 'X281', 'X282', 'X283', 'X284', 'X285', 'X286', 'X287', 'X288', 'X291',
'X292', 'X294', 'X295', 'X296', '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', 'X331', 'X332', 'X333', 'X334', 'X335',
'X336', 'X337', 'X338', 'X339', 'X340', 'X341', 'X342', 'X343', 'X344', 'X345',
'X346', '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']
--------------------------------------------------
Columns containing the unique values :  [0]
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297',
'X330', 'X347']
--------------------------------------------------
```

Categorical features.

In [13]:
```
# Let's print some categorical feature rows.
cat_feat = counts[2]
train_df[cat_feat].head()
```

Out[13]:

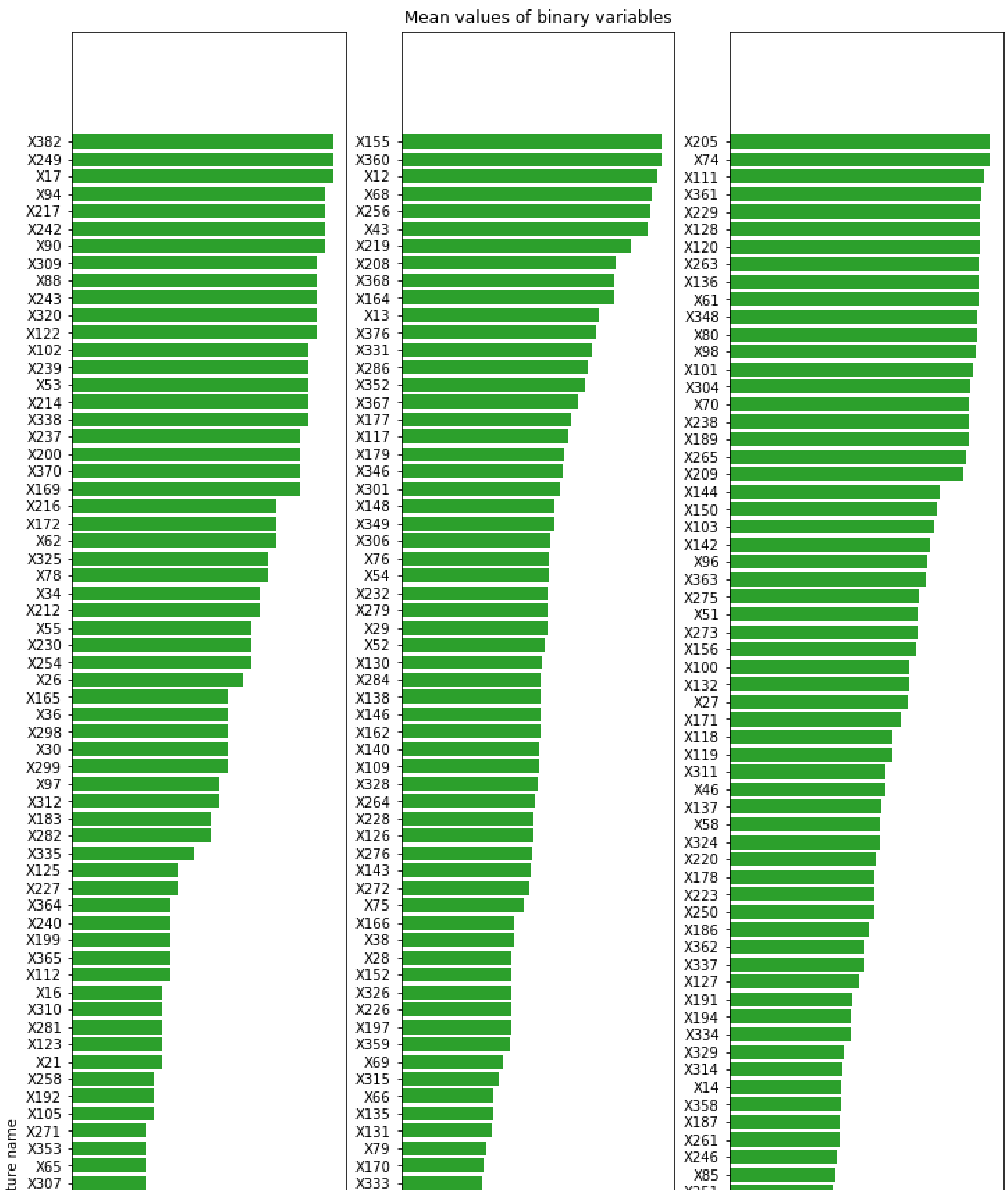
	X0	X1	X2	X3	X4	X5	X6	X8
0	k	v	at	a	d	u	j	o
1	k	t	av	e	d	y	l	o
2	az	w	n	c	d	x	j	x
3	az	t	n	f	d	x	l	e
4	az	v	n	f	d	h	d	n

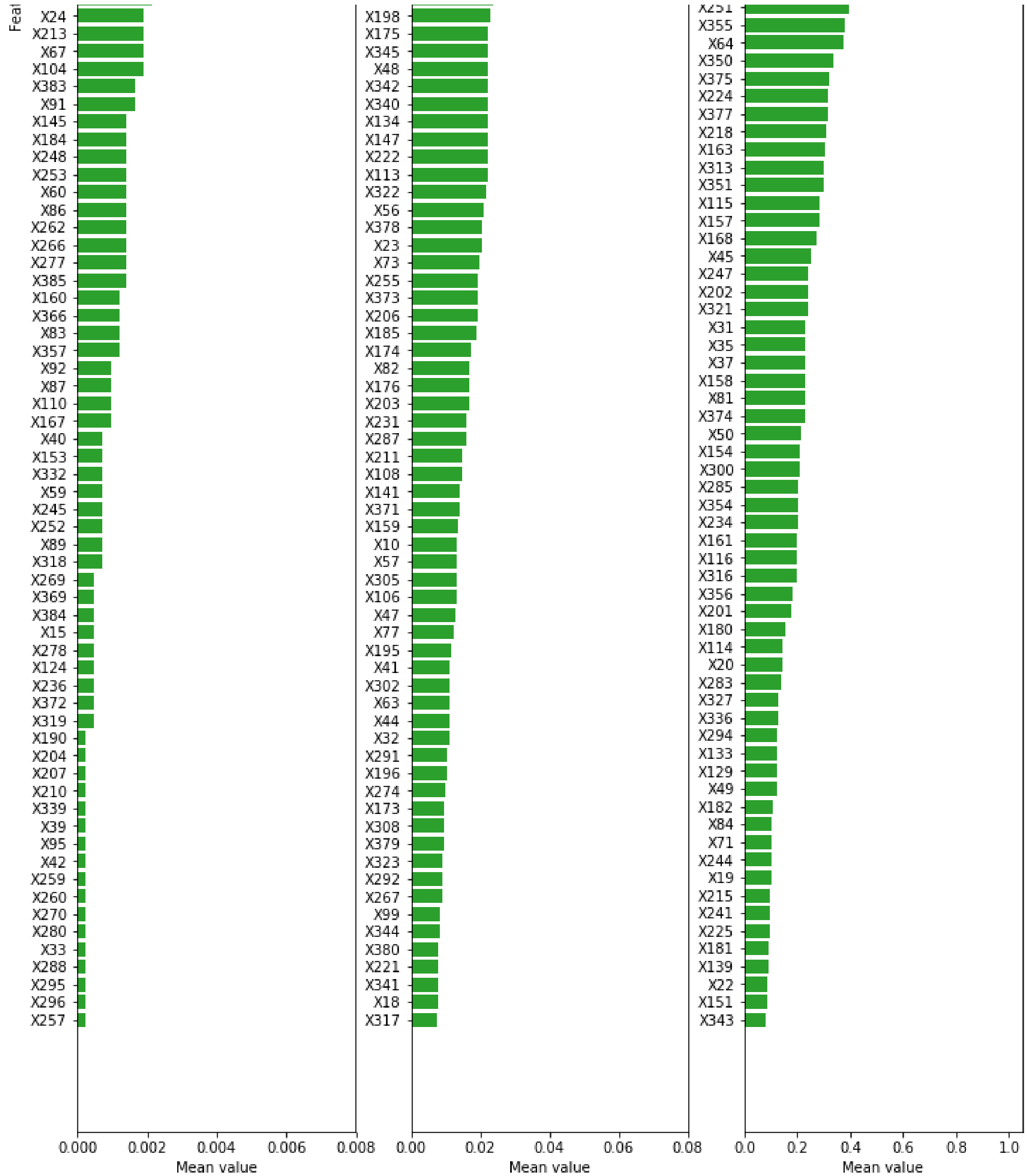
Features:

1. Constant features: 12
2. Binary features: 356
3. Categorical features: 8

We have 12 features which only have a single value in them - these are pretty useless for supervised algorithms, and should probably be dropped.

In [14]:
```
binary_means = [np.mean(train_df[c]) for c in counts[1]]
binary_names = np.array(counts[1])[np.argsort(binary_means)]
binary_means = np.sort(binary_means)

fig, ax = plt.subplots(1, 3, figsize=(12,30))
ax[0].set_ylabel('Feature name')
ax[1].set_title('Mean values of binary variables')
for i in range(3):
    names, means = binary_names[i*119:(i+1)*119], binary_means[i*119:(i+1)*119]
    ax[i].barh(range(len(means)), means, color=color[2])
    ax[i].set_xlabel('Mean value')
    ax[i].set_yticks(range(len(means)))
    ax[i].set_yticklabels(names, rotation='horizontal')
plt.show()
```





From above plot we can understand the general mean values of all Binary features.

Machine Learning Algorithms.

We can solve Regression problem to optimize the Production Time Feature (i.e. Target Variable = y).

Baseline Model 1: xgboost model

To analyse Important Variables

In [15]:
```
# Data Preprocessing.

# LabelEncoder: Used to Encode labels with value between 0 and n_classes-1.

for f in ["X0", "X1", "X2", "X3", "X4", "X5", "X6", "X8"]:
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train_df[f].values)) # Fit label encoder
        train_df[f] = lbl.transform(list(train_df[f].values)) # Transform labels 

# Dropping ID feature & creating saperate Input/Output training data.
train_y = train_df['y'].values
train_X = train_df.drop(["ID", "y"], axis=1)

# Reference : https://xgboost.readthedocs.io/en/latest/python/python_intro.html
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.values)
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=100, feval=
```

```
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
6 extra nodes, 0 pruned nodes, max_depth=2
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
6 extra nodes, 0 pruned nodes, max_depth=2
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
```

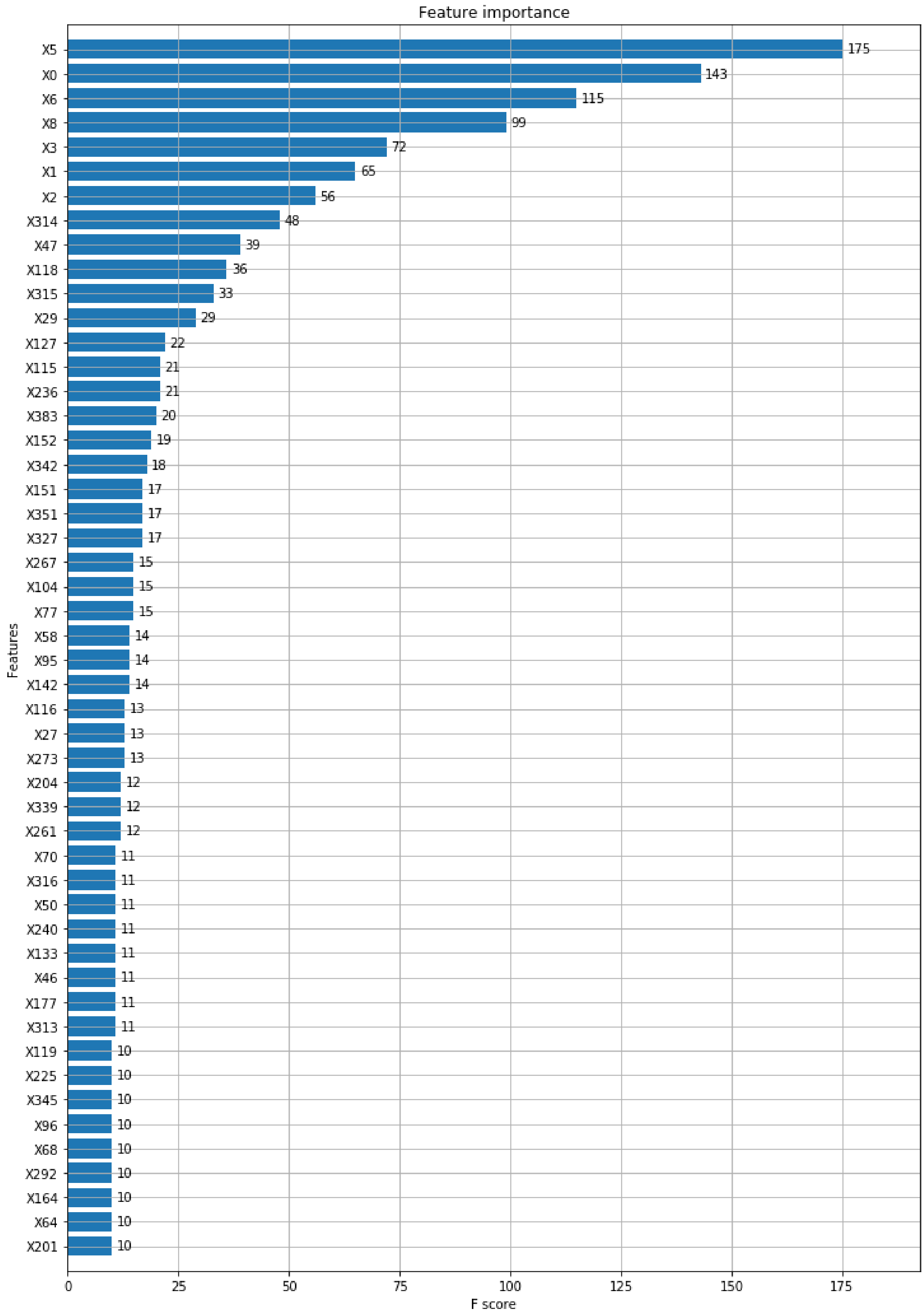
```
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
10 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
10 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
8 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
10 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
10 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
18 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
10 extra nodes, 0 pruned nodes, max_depth=3
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
12 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
20 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
20 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
16 extra nodes, 0 pruned nodes, max_depth=4
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
18 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
34 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
30 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
42 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
48 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
48 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
40 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
62 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
60 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
60 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
54 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
70 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
```

```
64 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
58 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
62 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
72 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
94 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
76 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
54 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
106 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
78 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
54 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
42 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
64 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
64 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
102 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
66 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
58 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
58 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
50 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
66 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
50 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
46 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
70 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
56 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
70 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
62 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
66 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
44 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
48 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
32 extra nodes, 0 pruned nodes, max_depth=6
```

```
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
38 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
36 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
78 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
30 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
28 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
44 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
46 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
66 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
32 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
44 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
48 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
64 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
34 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
60 extra nodes, 0 pruned nodes, max_depth=6

[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
88 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
20 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
46 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
36 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
24 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
46 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
72 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
44 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
42 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
86 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
62 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
82 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
90 extra nodes, 0 pruned nodes, max_depth=6
[10:41:50] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
30 extra nodes, 0 pruned nodes, max_depth=6
```

```
[10:41:51] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 roots,
50 extra nodes, 0 pruned nodes, max_depth=6
```

In [16]:
```
# 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()
```

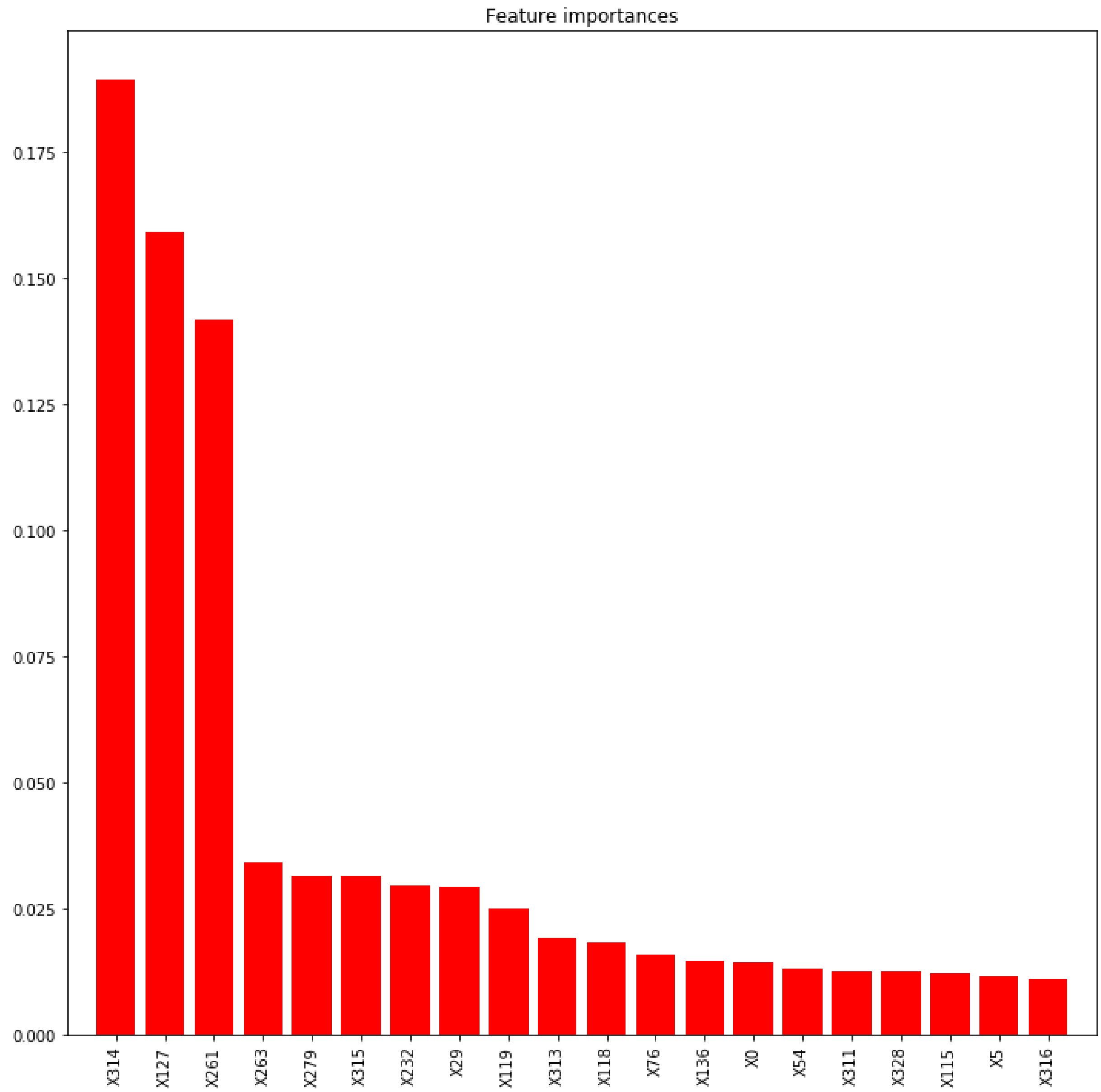


Observations:

1. **Categorical features X5,X0,X8,X6,X1,X2,X3 are highly important in prediction of XGBoost model.**
2. **Binary features are less important comparitively.**
3. **We have dropped ID feature as it is not important.**

Baseline Model 2: Random Forest model

In [17]:
```
from sklearn import ensemble
model = ensemble.RandomForestRegressor(n_estimators=200, max_depth=10, min_sample
model.fit(train_X, train_y)
feat_names = train_X.columns.values

## plot the importances ##
importances = model.feature_importances_
std = np.std([tree.feature_importances_ for tree in model.estimators_], axis=0)
indices = np.argsort(importances)[::-1][:20]

plt.figure(figsize=(12,12))
plt.title("Feature importances")
plt.bar(range(len(indices)), importances[indices], color="r", align="center")
plt.xticks(range(len(indices)), feat_names[indices], rotation='vertical')
plt.xlim([-1, len(indices)])
plt.show()
```



Observations:

1. **Binary features X314,X127,X261 are highly important in the prediction of Random forest model.**
2. **Categorical features are less important comparitively.**
3. **We have dropped ID feature as it is not important.**

XGboost Regression model.

Creating the components using various dimensionality reduction techniques.

In [19]:
```
# Reference : https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reducti

from sklearn.decomposition import PCA, FastICA, TruncatedSVD
# Dimensionality reduction techniques
n_comp = 15

# tSVD
tsvd = TruncatedSVD(n_components=n_comp, random_state=420)
tsvd_results_train = tsvd.fit_transform(train_X)
tsvd_results_test = tsvd.transform(test)
# PCA
pca = PCA(n_components=n_comp, random_state=42)
pca2_results_train = pca.fit_transform(train_X)
pca2_results_test = pca.transform(test)

# ICA
ica = FastICA(n_components=n_comp, random_state=42)
ica2_results_train = ica.fit_transform(train_X)
ica2_results_test = ica.transform(test)

# Append decomposition components to datasets
for i in range(1, n_comp+1):
    train['tsvd_' + str(i)] = tsvd_results_train[:,i-1]
    test['tsvd_' + str(i)] = tsvd_results_test[:, i-1]

    train['pca_' + str(i)] = pca2_results_train[:,i-1]
    test['pca_' + str(i)] = pca2_results_test[:, i-1]

    train['ica_' + str(i)] = ica2_results_train[:,i-1]
    test['ica_' + str(i)] = ica2_results_test[:, i-1]

y_mean = np.mean(y_train)
```

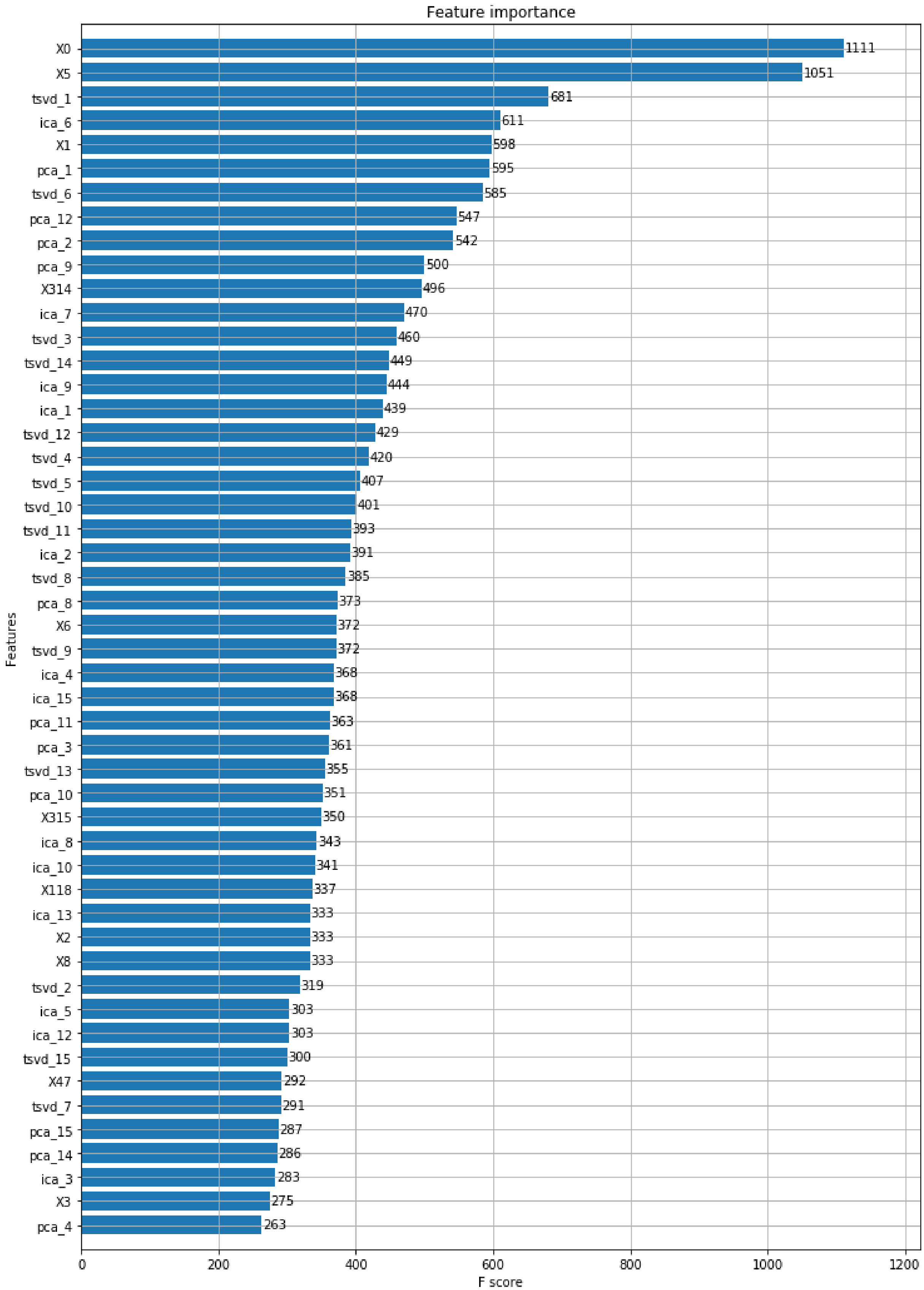
In [20]:
```
# Reference : https://www.analyticsvidhya.com/blog/2016/03/complete-guide-paramete

import xgboost as xgb

# Prepare dict of params for xgboost model.
xgb_params = {
    'n_trees': 500,
    'eta': 0.005,
    'max_depth':6,
    'subsample': 0.5,
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'base_score': y_mean, # base prediction = mean(target)
    'silent': 1}


# Creating DMatrices for Xgboost training
dtrain = xgb.DMatrix(train, y_train)
dtest = xgb.DMatrix(test)

# xgboost, cross-validation
cv_result = xgb.cv(xgb_params,dtrain,num_boost_round=700,verbose_eval=50,show_std

num_boost_rounds = len(cv_result)
print(num_boost_rounds)

# Train model
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=num_boost_r
```

```
[0]     train-rmse:12.3841      test-rmse:12.385
[50]    train-rmse:10.7518      test-rmse:10.8754
[100]   train-rmse:9.57748      test-rmse:9.84481
[150]   train-rmse:8.73635      test-rmse:9.16087
[200]   train-rmse:8.13691      test-rmse:8.72088
[250]   train-rmse:7.70472      test-rmse:8.44323
[300]   train-rmse:7.3866       test-rmse:8.27705
[350]   train-rmse:7.14044      test-rmse:8.18174
[400]   train-rmse:6.93932      test-rmse:8.12804
[450]   train-rmse:6.77325      test-rmse:8.09886
[500]   train-rmse:6.63014      test-rmse:8.084
[550]   train-rmse:6.51027      test-rmse:8.07597
[600]   train-rmse:6.40507      test-rmse:8.07439
[650]   train-rmse:6.30951      test-rmse:8.07828
[699]   train-rmse:6.22381      test-rmse:8.08397
700
[04:52:04] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 64 extra nodes, 0 pruned nodes, max_depth=6
[04:52:04] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
```

Observations:

1. **After 700 num_boost_round: 1. train-rmse:6.22381 2. test-rmse:8.08397**
2. We have pretty decent values of Train & Test error parameter(RMSE).
3. Model is performing nicely & not overfitting.

In [21]:
```
# 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()
```



Observations:

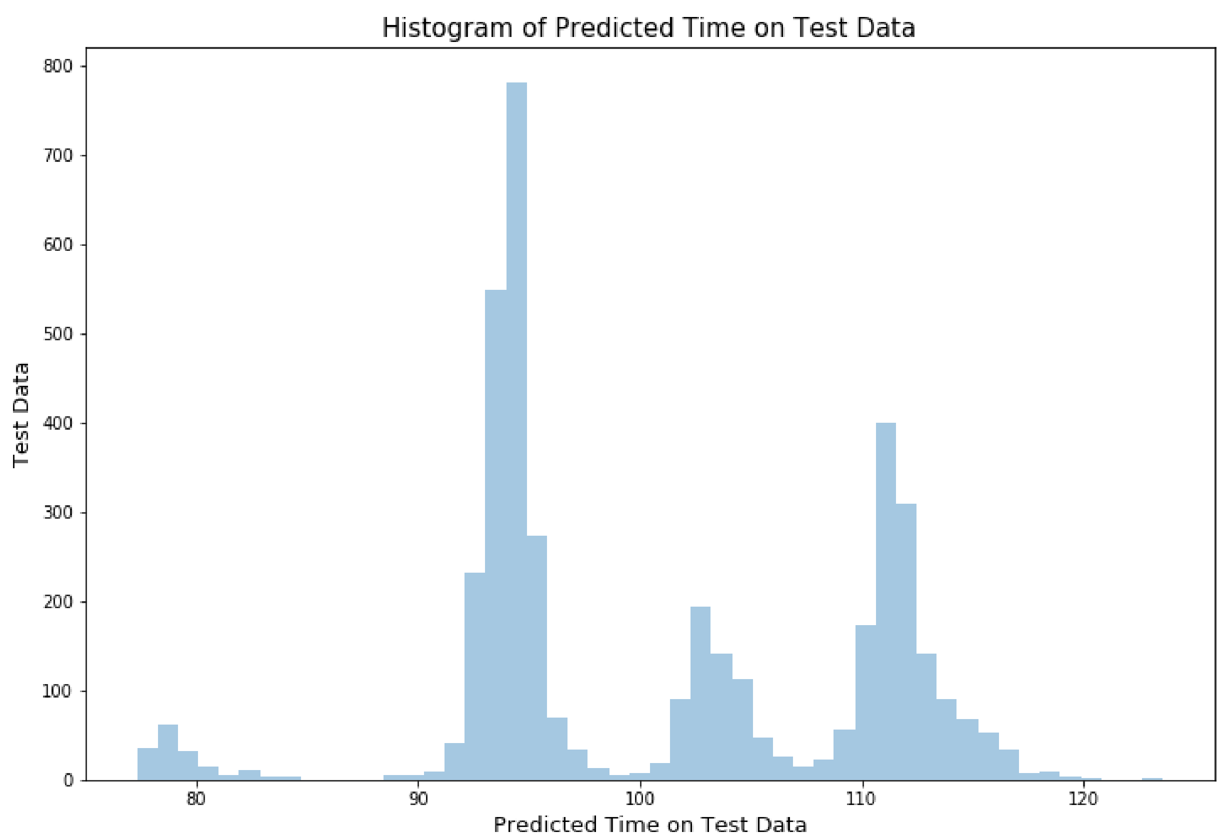
1. Categorical features **X0 & X5 are highly important** in the prediction of our XGBoost model.
2. **TSVD,PCA & ICA generated features are also contributing effectively in the prediction.**
3. We can also **drop the features that are less important** to increase the model effectivity/time to predict target Variable.

In [22]:
```
# check f2-score (to get higher score - increase num_boost_round in previous cell
from sklearn.metrics import r2_score

# now fixed, correct calculation
print(r2_score(dtrain.get_label(), model.predict(dtrain)))
```

```
0.718001643662283
```

r2_score = 0.718

In [23]:
```
test = pd.read_csv('test.csv')
y_pred = model.predict(dtest)
output = pd.DataFrame({'id': test['ID'].astype(np.int32), 'y': y_pred})
output.to_csv('XGB_test_results.csv', index=False)
output = pd.DataFrame({'id': test['ID'].astype(np.int32), 'y': y_pred})
plt.figure(figsize=(12,8))
sns.distplot(output.y.values, bins=50, kde=False)
plt.xlabel('Predicted Time on Test Data', fontsize=13)
plt.ylabel('Test Data', fontsize=13)
plt.title('Histogram of Predicted Time on Test Data', fontsize=15)
plt.show()
```



Stacked Regression model.

In [16]:
```
# Reference: https://github.com/nilaysen/Mercedes-Benz-Greener-Manufacturing-Kagg

class StackingEstimator(BaseEstimator, TransformerMixin):

    def __init__(self, estimator):
        self.estimator = estimator


    def fit(self, X, y=None, **fit_params):
        self.estimator.fit(X, y, **fit_params)
        return self

    def transform(self, X):
        X = check_array(X)
        X_transformed = np.copy(X)
        # add class probabilities as a synthetic feature
        if issubclass(self.estimator.__class__, ClassifierMixin) and hasattr(self
            X_transformed = np.hstack((self.estimator.predict_proba(X), X))

        # add class prodiction as a synthetic feature
        X_transformed = np.hstack((np.reshape(self.estimator.predict(X), (-1, 1))

        return X_transformed
```

In [17]:
```
# Import the data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

print("Train shape : ", train.shape)
print("Test shape : ", test.shape)
```

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

In [18]:
```
# Data Preprocessing.

# LabelEncoder: Used to Encode labels with value between 0 and n_classes-1.

for c in train.columns:
    if train[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train[c].values) + list(test[c].values))# Fit label encoder
        train[c] = lbl.transform(list(train[c].values))# Transform labels to norm
        test[c] = lbl.transform(list(test[c].values))# Transform labels to normal

# Dropping ID feature & creating saperate Input/Output training data.
train_y = train['y'].values
y_mean = np.mean(train_y)
id_test = test['ID'].values
train = train.drop(["ID"], axis=1)
test = test.drop(["ID"], axis=1)
# Reference : https://xgboost.readthedocs.io/en/latest/python/python_intro.html
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

# Save columns list before adding the decomposition components

usable_columns = list(set(train.columns) - set(['y']))
```

Creating the components using various dimensionality reduction techniques.

In [19]:
```
# Dimensionality reduction techniques
n_comp = 15

# tSVD
tsvd = TruncatedSVD(n_components=n_comp, random_state=420)
tsvd_results_train = tsvd.fit_transform(train.drop(["y"], axis=1))
tsvd_results_test = tsvd.transform(test)
# PCA
pca = PCA(n_components=n_comp, random_state=42)
pca2_results_train = pca.fit_transform(train.drop(["y"], axis=1))
pca2_results_test = pca.transform(test)

# ICA
ica = FastICA(n_components=n_comp, random_state=42)
ica2_results_train = ica.fit_transform(train.drop(["y"], axis=1))
ica2_results_test = ica.transform(test)

# GRP
grp = GaussianRandomProjection(n_components=n_comp, eps=0.1, random_state=42)
grp_results_train = grp.fit_transform(train.drop(["y"], axis=1))
grp_results_test = grp.transform(test)

# SRP
srp = SparseRandomProjection(n_components=n_comp, dense_output=True, random_state
srp_results_train = srp.fit_transform(train.drop(["y"], axis=1))
srp_results_test = srp.transform(test)

# Append decomposition components to datasets
for i in range(1, n_comp+1):

    train['tsvd_' + str(i)] = tsvd_results_train[:,i-1]
    test['tsvd_' + str(i)] = tsvd_results_test[:, i-1]

    train['pca_' + str(i)] = pca2_results_train[:,i-1]
    test['pca_' + str(i)] = pca2_results_test[:, i-1]

    train['ica_' + str(i)] = ica2_results_train[:,i-1]
    test['ica_' + str(i)] = ica2_results_test[:, i-1]

    train['grp_' + str(i)] = grp_results_train[:, i-1]
    test['grp_' + str(i)] = grp_results_test[:, i-1]

    train['srp_' + str(i)] = srp_results_train[:, i-1]
    test['srp_' + str(i)] = srp_results_test[:,i-1]
```

In [20]:
```
# final_train and final_test are data to be used only the stacked model (does not

final_train = train[usable_columns].values
final_test = test[usable_columns].values

# Reference : https://www.analyticsvidhya.com/blog/2016/03/complete-guide-paramete

import xgboost as xgb

# Prepare dict of params for xgboost model.
xgb_params = {
    'n_trees': 500,
    'eta': 0.005,
    'max_depth':5,
    'subsample': 0.9,
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'base_score': y_mean, # base prediction = mean(target)
    'silent': 1}


# Creating DMatrices for Xgboost training
dtrain = xgb.DMatrix(train.drop(["y"], axis=1), train_y)
dtest = xgb.DMatrix(test)

# xgboost, cross-validation
cv_result = xgb.cv(xgb_params,dtrain,num_boost_round=1000,verbose_eval=50,show_st

num_boost_rounds = len(cv_result)
print(num_boost_rounds)

# Train model
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=num_boost_r
y_pred = model.predict(dtest)
```

```
[550]   train-rmse:7.05194      test-rmse:8.51768
[600]   train-rmse:6.94264      test-rmse:8.52118
[650]   train-rmse:6.84518      test-rmse:8.52708
[700]   train-rmse:6.74954      test-rmse:8.5383
[750]   train-rmse:6.6645       test-rmse:8.54567
[800]   train-rmse:6.57573      test-rmse:8.55308
[850]   train-rmse:6.49414      test-rmse:8.5616
[900]   train-rmse:6.41803      test-rmse:8.56904
[950]   train-rmse:6.34769      test-rmse:8.57765
[999]   train-rmse:6.28061      test-rmse:8.58585
1000
[11:09:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 46 extra nodes, 0 pruned nodes, max_depth=5
[11:09:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 48 extra nodes, 0 pruned nodes, max_depth=5
[11:09:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 48 extra nodes, 0 pruned nodes, max_depth=5
[11:09:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 48 extra nodes, 0 pruned nodes, max_depth=5
[11:09:48] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
```

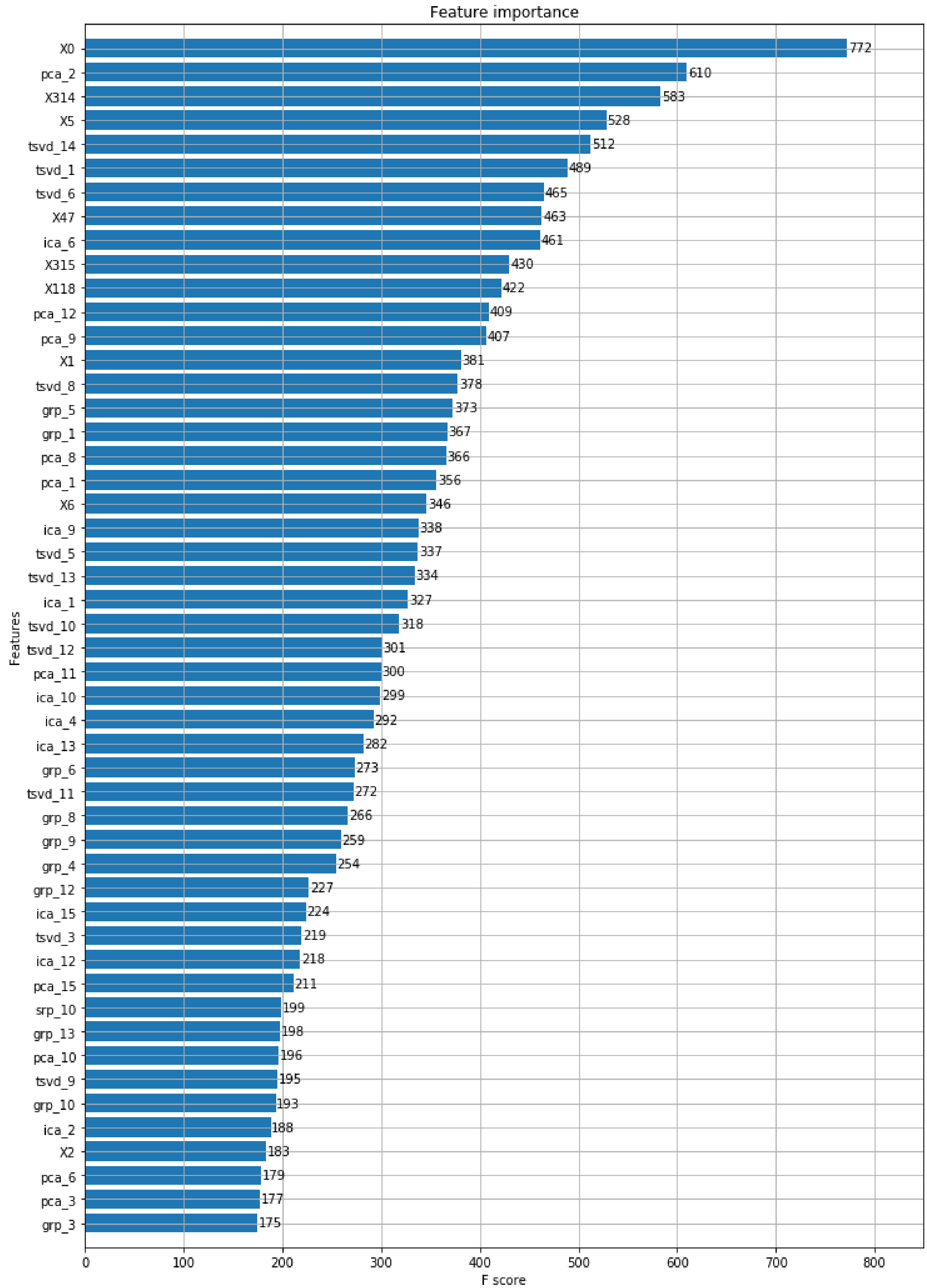
In [21]:
```
# Train the stacked models then predict the test data !!
from sklearn.pipeline import make_pipeline, make_union
Stacked_pipeline = make_pipeline(
    StackingEstimator(estimator=LassoLarsCV(normalize=True)),
    StackingEstimator(estimator=GradientBoostingRegressor(learning_rate=0.001,los
                                                          min_samples_leaf=18,
                                                          min_samples_split=14, s
    LassoLarsCV())



Stacked_pipeline.fit(final_train, train_y)
predictions = Stacked_pipeline.predict(final_test)

# R2 Score on the entire Train data when averaging

print('R2 score on train data:')
print(r2_score(train_y,Stacked_pipeline.predict(final_train)*0.2855 + model.predi

# Average the preditionon test data  of both models then save it on a csv file.

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = y_pred*0.75 + predictions*0.25
sub.to_csv('stacked_model_pred.csv', index=False)
```

```
R2 score on train data:
0.6805083932610694
```

Observations:

1. **After 1000 num_boost_round: 1. train-rmse:6.28061 2. test-rmse:8.58585**
2. We have pretty decent values of Train & Test error parameter(RMSE).
3. Model is performing nicely & not overfitting.
4. **R2 score on train data: 0.6805083932610694**

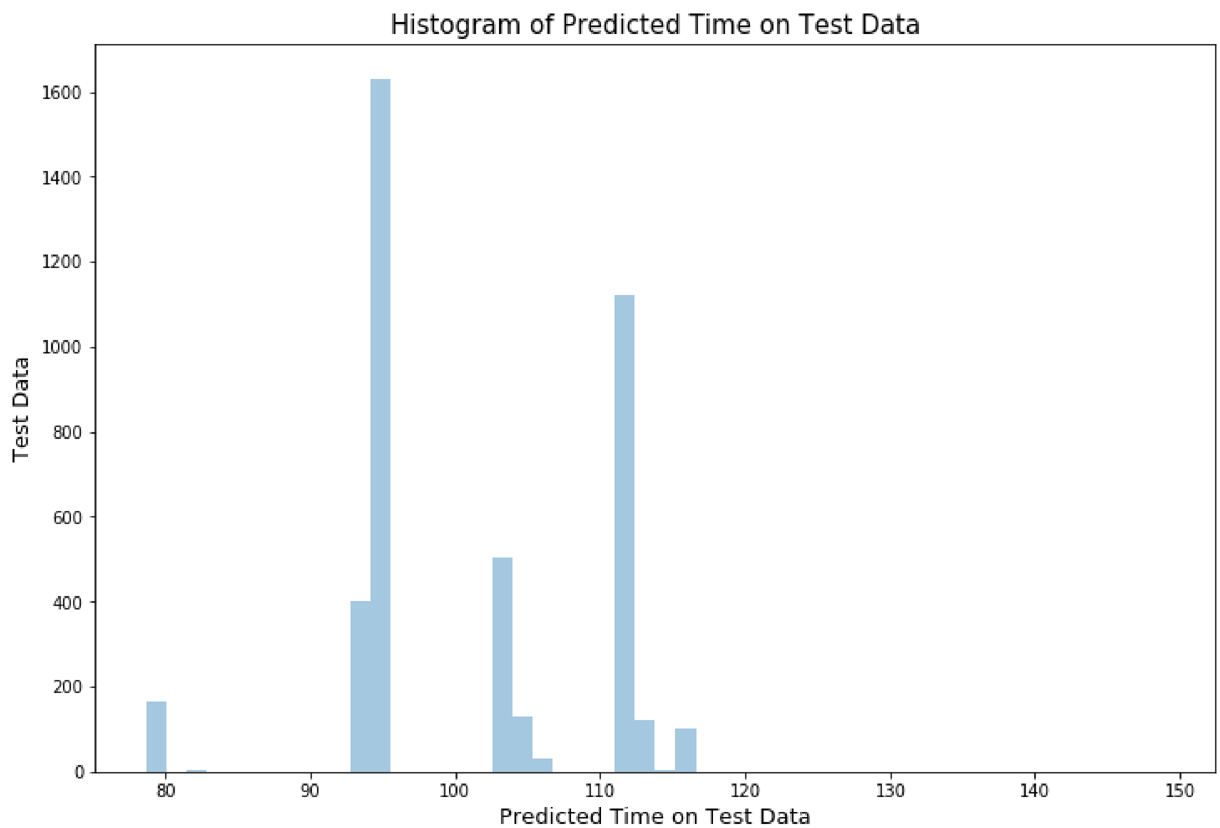
In [22]:
```
# Plot the important features #
fig, ax = plt.subplots(figsize=(12,18))
xgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
plt.show()
```



Observations:

1. Categorical features **X0 & X5 are highly important** in the prediction of our XGBoost model.
2. **TSVD,PCA & ICA generated features are also contributing effectively in the prediction.**
3. We can also **drop the features that are less important** to increase the model effectivity/time to predict target Variable.

In [23]:
```
test = pd.read_csv('test.csv')

output = pd.DataFrame({'id': test['ID'].astype(np.int32), 'y': predictions})

plt.figure(figsize=(12,8))
sns.distplot(output.y.values, bins=50, kde=False)
plt.xlabel('Predicted Time on Test Data', fontsize=13)
plt.ylabel('Test Data', fontsize=13)
plt.title('Histogram of Predicted Time on Test Data', fontsize=15)
plt.show()
```



Deep Learning Algorithms.

We can solve Regression problem to optimize the Production Time Feature (i.e. Target Variable = y).

Baseline Model: MLP using Keras

In [19]:
```
# Data preprocessing.

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# removing the outliers.
train = train.loc[train['y'] < 200, :]

# seperating label and features.
y_train = train['y']

train = train.drop(["ID", "y"], axis=1)

test = test.drop(["ID"], axis=1)

# label encoding the categorical features for dimension reduction.
for c in train.columns:
    if train[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train[c].values) + list(test[c].values))
        train[c] = lbl.transform(list(train[c].values))
        test[c] = lbl.transform(list(test[c].values))
```

In [20]:
```
print(train.shape)
print(test.shape)
```

```
(4208, 376)
(4209, 376)
```

In [21]:
```
from keras import optimizers

adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0

# Define custom R2 metrics for Keras backend.
from keras import backend as K

def r2_keras(y_true, y_pred):
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res / (SS_tot + K.epsilon()))

# Reference:https://stackoverflow.com/questions/45250100/kerasregressor-coefficie
```

In [22]:
```
# Initialize input dimensions variable.
input_dims = train.shape[1]
```

In [23]:
```
# Architecture of MLP.

def nn_model():
    model = Sequential()
    # Input layer.
    model.add(Dense(input_dims, input_dim=input_dims))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer1
    model.add(Dense(input_dims))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer2
    model.add(Dense(input_dims//2))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer3
    model.add(Dense(input_dims//4, activation='relu'))

    # Output layer (y_pred)
    model.add(Dense(1, activation='linear'))

    return model


model = nn_model()
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 376)               141752
_________________________________________________________________
batch_normalization_1 (Batch (None, 376)               1504
_________________________________________________________________
activation_1 (Activation)    (None, 376)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 376)               0
_________________________________________________________________
dense_2 (Dense)              (None, 376)               141752
_________________________________________________________________
batch_normalization_2 (Batch (None, 376)               1504
_________________________________________________________________
activation_2 (Activation)    (None, 376)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 376)               0
_________________________________________________________________
dense_3 (Dense)              (None, 188)               70876
_________________________________________________________________
batch_normalization_3 (Batch (None, 188)               752
_________________________________________________________________
activation_3 (Activation)    (None, 188)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 188)               0
```

```
_________________________________________________________________
dense_4 (Dense)              (None, 94)                17766     
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 95        
=================================================================
Total params: 376,001
Trainable params: 374,121
Non-trainable params: 1,880
_________________________________________________________________
None
```

In [26]:
```
filepath="weights_baseline_mlp.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_r2_keras', verbose=1, save_be
callbacks_list = [checkpoint]
```

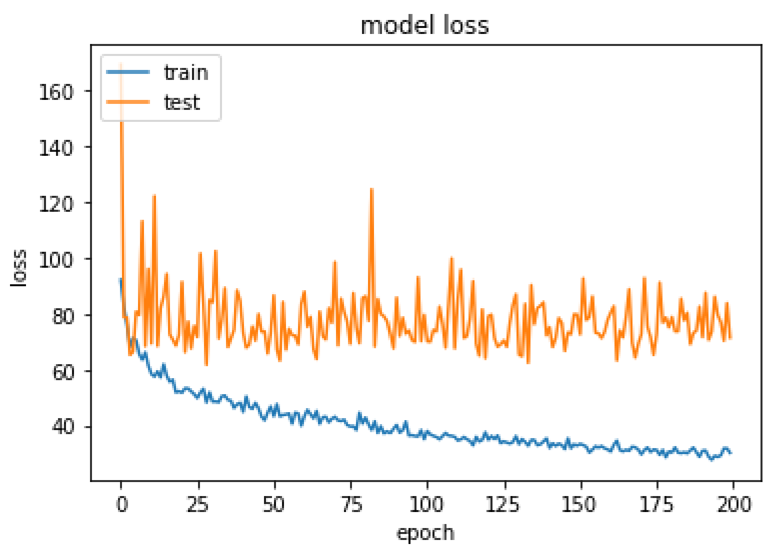
In [27]:
```
# Fitting the model on the training data.
model.compile(loss='mean_squared_error', optimizer=adam, metrics=[r2_keras])
history = model.fit(train, y_train, nb_epoch = 200, batch_size=50, shuffle=True,v
                    validation_split=0.3,callbacks=callbacks_list)
```

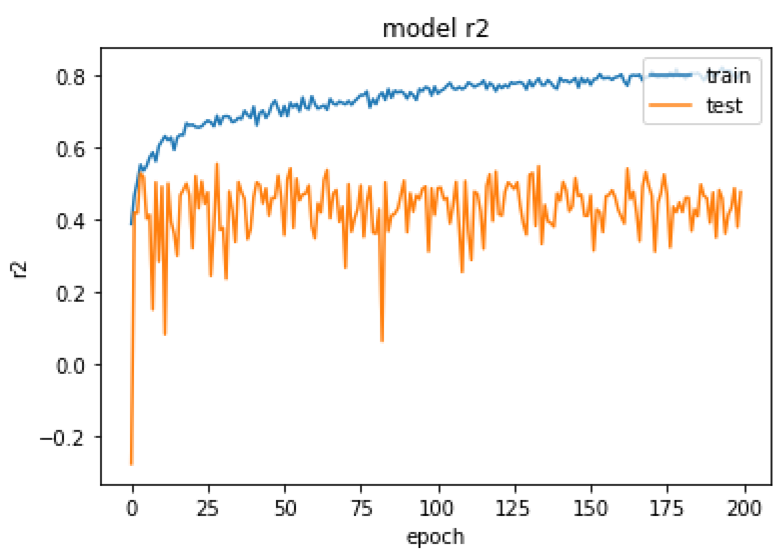
```
Train on 2945 samples, validate on 1263 samples
Epoch 1/200
2945/2945 [==============================] - 1s 485us/step - loss: 92.0901 -
r2_keras: 0.3881 - val_loss: 169.4072 - val_r2_keras: -0.2759

Epoch 00001: val_r2_keras improved from -inf to -0.27591, saving model to wei
ghts_baseline_mlp.best.hdf5
Epoch 2/200
2945/2945 [==============================] - 1s 178us/step - loss: 82.6006 -
r2_keras: 0.4670 - val_loss: 78.7804 - val_r2_keras: 0.4198

Epoch 00002: val_r2_keras improved from -0.27591 to 0.41983, saving model to
weights_baseline_mlp.best.hdf5
Epoch 3/200
2945/2945 [==============================] - 1s 178us/step - loss: 76.0682 -
r2_keras: 0.5060 - val_loss: 78.7540 - val_r2_keras: 0.4185

Epoch 00003: val_r2_keras did not improve from 0.41983
Epoch 4/200
```

After 29 epochs:

1. **Training Data:**
   A. **mean_squared_error loss: 48.2294**
   B. **r2 metric: 0.6881**
2. **Validation Data:**
   A. **mean_squared_error loss: 61.74**
   B. **r2 metric: 0.5546**

In [28]:
```
# Plot Loss & R2 metric.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r2')
plt.ylabel('r2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```





Observations:

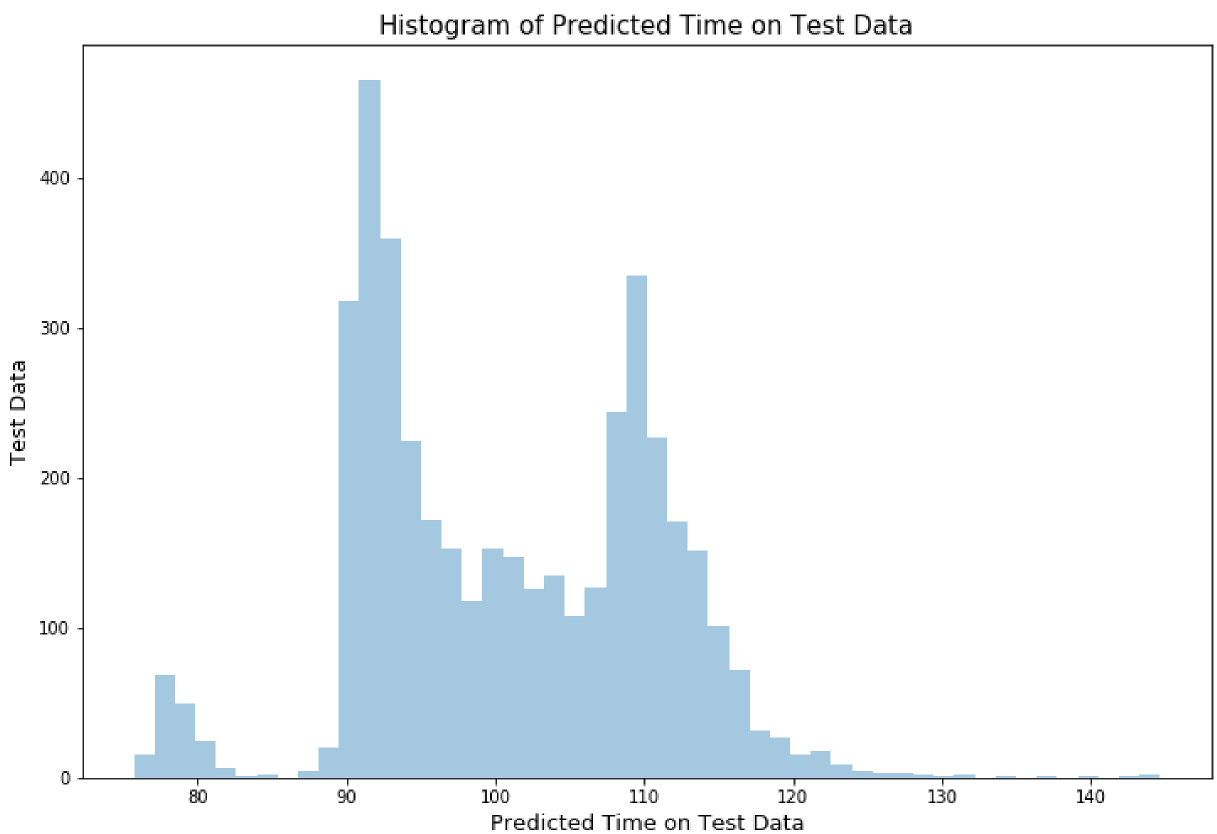
1. We can easily see that model is predicting the Target variable brillianlty.
2. The Loss & R2 metric graphs converge after few epochs.

3. **No Overfitting.**

In [29]:
```
# Passing the Test data through the trained model & storing the results on disk.

Dtest = pd.read_csv('test.csv')
# predict results
res = model.predict(test).ravel()
print(res)

# create df and convert it to csv
output = pd.DataFrame({'id': Dtest["ID"], 'y': res})
output.to_csv('keras-baseline.csv', index=False)
```

```
[ 85.117355  94.5353    79.48812  ...  96.592606 107.8587    95.03621 ]
```

In [30]:
```
plt.figure(figsize=(12,8))
sns.distplot(output.y.values, bins=50, kde=False)
plt.xlabel('Predicted Time on Test Data', fontsize=13)
plt.ylabel('Test Data', fontsize=13)
plt.title('Histogram of Predicted Time on Test Data', fontsize=15)
plt.show()
```



Observations:

1. The distribution of the Predicted Time (y) on Test Data is almost similar to the input data distribution.

Final Model: MLP using Keras.

In [26]:
```
# Data preprocessing.

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# removing the outliers.
train = train.loc[train['y'] < 200, :]

# seperating label and features
y_train = train['y']
train = train.drop(["ID", "y"], axis=1)
test = test.drop(["ID"], axis=1)
y_mean = np.mean(y_train)
# label encoding the categorical features for dimension reduction
for c in train.columns:
    if train[c].dtype == 'object':
        lbl = LabelEncoder()
        lbl.fit(list(train[c].values) + list(test[c].values))
        train[c] = lbl.transform(list(train[c].values))
        test[c] = lbl.transform(list(test[c].values))
```

In [27]:
```
# Dimensionality reduction techniques

n_comp = 15

# tSVD
tsvd = TruncatedSVD(n_components=n_comp, random_state=42)
tsvd_results_train = tsvd.fit_transform(train)
tsvd_results_test = tsvd.transform(test)
# PCA
pca = PCA(n_components=n_comp, random_state=42)
pca2_results_train = pca.fit_transform(train)
pca2_results_test = pca.transform(test)

# ICA
ica = FastICA(n_components=n_comp, random_state=42)
ica2_results_train = ica.fit_transform(train)
ica2_results_test = ica.transform(test)

# GRP
grp = GaussianRandomProjection(n_components=n_comp, eps=0.1, random_state=42)
grp_results_train = grp.fit_transform(train)
grp_results_test = grp.transform(test)

# SRP
srp = SparseRandomProjection(n_components=n_comp, dense_output=True, random_state
srp_results_train = srp.fit_transform(train)
srp_results_test = srp.transform(test)

# Append decomposition components to datasets
for i in range(1, n_comp+1):
    train['tsvd_' + str(i)] = tsvd_results_train[:,i-1]
    test['tsvd_' + str(i)] = tsvd_results_test[:, i-1]

    train['pca_' + str(i)] = pca2_results_train[:,i-1]
    test['pca_' + str(i)] = pca2_results_test[:, i-1]

    train['ica_' + str(i)] = ica2_results_train[:,i-1]
    test['ica_' + str(i)] = ica2_results_test[:, i-1]

    train['grp_' + str(i)] = grp_results_train[:,i-1]
    test['grp_' + str(i)] = grp_results_test[:, i-1]

    train['srp_' + str(i)] = srp_results_train[:,i-1]
    test['srp_' + str(i)] = srp_results_test[:, i-1]
```

In [28]:
```
# Define custom R2 metrics for Keras backend
from keras import backend as K

def r2_keras(y_true, y_pred):
    SS_res =  K.sum(K.square( y_true - y_pred ))
    SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )

# Reference:https://stackoverflow.com/questions/45250100/kerasregressor-coefficie
```

In [29]:
```
# Model architecture definition.
def model():
    model = Sequential()
    #input layer
    model.add(Dense(input_dims, input_dim=input_dims))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer1
    model.add(Dense(input_dims))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer2
    model.add(Dense(input_dims//2))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.4))
    # hidden layer3
    model.add(Dense(input_dims//4, activation='relu'))

    # output layer (y_pred)
    model.add(Dense(1, activation='linear'))

    # compile this model
    model.compile(loss='mean_squared_error',optimizer='adam',metrics=[r2_keras])

    # Visualize NN architecture
    print(model.summary())
    return model
```

In [30]:
```
# Initialize input dimension
input_dims = train.shape[1]

# To make Results reproducible
np.random.seed(seed)

# Initialize estimator, wrap model in KerasRegressor.
# Reference : https://stackoverflow.com/questions/44132652/keras-how-to-perform-a

estimator = KerasRegressor(build_fn=model,nb_epoch=150,batch_size=20,verbose=1)
```

In [31]:
```
filepath="weights_final_mlp.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_r2_keras', verbose=1, save_be
callbacks_list = [checkpoint]
```

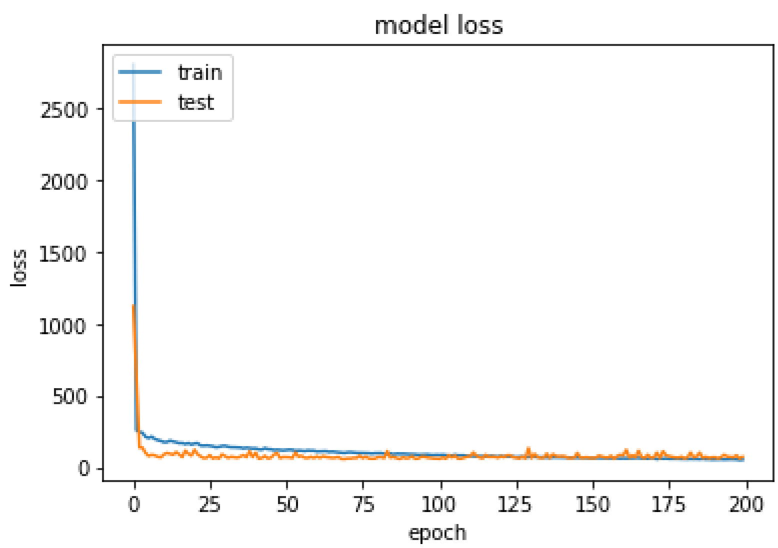
In [32]:
```
# Fit the estimator.
history = estimator.fit(train,y_train,epochs=200,validation_split=0.3,verbose=2,s
```

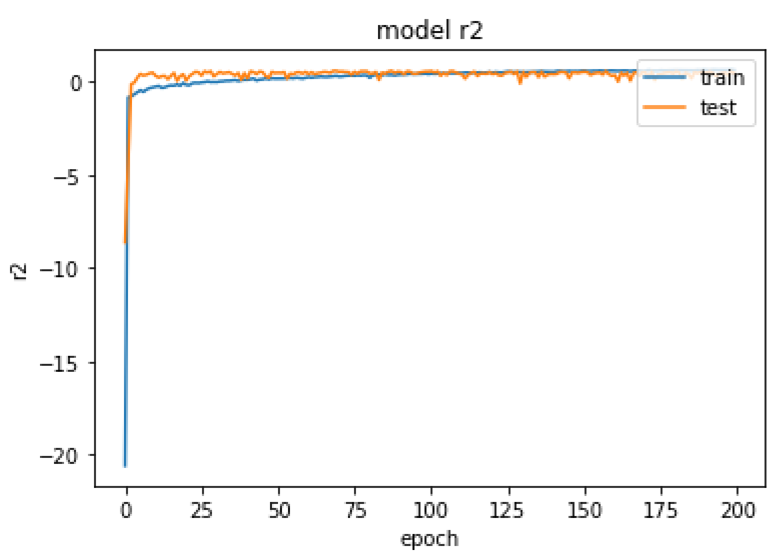
```
 - 1s - loss: 92.7324 - r2_keras: 0.3655 - val_loss: 57.1178 - val_r2_keras:
0.5780

Epoch 00088: val_r2_keras improved from 0.57648 to 0.57795, saving model to w
eights_final_mlp.best.hdf5
Epoch 89/200
```

After 88 epochs:

1. **Training Data:**
   A. **mean_squared_error loss: 92.732**
   B. **r2 metric: 0.3655**
2. **Validation Data:**
   A. **mean_squared_error loss: 57.118**
   B. **r2 metric: 0.57795**

In [33]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r2')
plt.ylabel('r2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```





Observations:

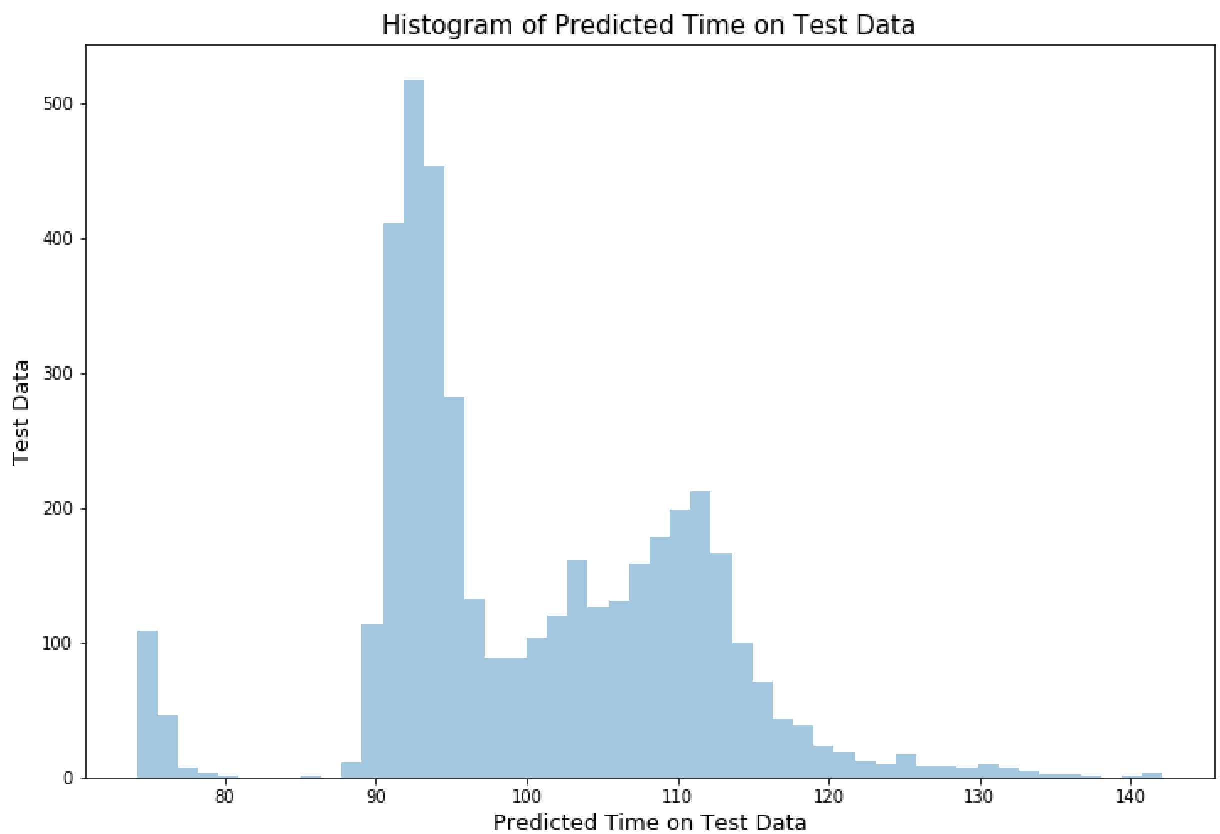
1. We can easily see that model is predicting the Target variable brillianlty.
2. The Loss & R2 metric graphs converge after few epochs.

3. **No Overfitting.**

In [34]:
```
Dtest = pd.read_csv('test.csv')
# predict results
res = estimator.predict(test).ravel()
print(res)

# create df and convert it to csv
output = pd.DataFrame({'id': Dtest["ID"], 'y': res})
output.to_csv('Keras_final.csv', index=False)
```

```
4209/4209 [==============================] - 0s 84us/step
[ 76.85151   92.717636  74.61126  ...  93.58735  108.86175   92.010956]
```

In [35]:
```
plt.figure(figsize=(12,8))
sns.distplot(output.y.values, bins=50, kde=False)
plt.xlabel('Predicted Time on Test Data', fontsize=13)
plt.ylabel('Test Data', fontsize=13)
plt.title('Histogram of Predicted Time on Test Data', fontsize=15)
plt.show()
```



Observations:

1. The distribution of the Predicted Time (y) on Test Data is shifted left compared to the input data distribution.

Models Performance Table

Mercedes-Benz Greener Manufacturing Case Study(Regression)				
Sr. No.	Model	Train Error/Loss (RMSE)	Test Error/Loss (RMSE)	R2 metric.
1	XGBoost model	6.22	8.08	0.72
2	Stacking Model	6.28	8.58	0.68
3	Baseline MLP Model	6.94	7.85	0.55
4	Final MLP Model	9.62	7.55	0.58

¶

Conclusion:

1. **We have taken the Mercedes-Benz Greener Manufacturing data.**
2. **We have trained ML & DL models on the data.**
3. **Stacking model gives best performence.**