MERCEDES BENZ___BEMBERKAR SHASHANKSAI

June 15, 2020

[43]:

```
import pandas as pd
import numpy as np
# import plotting libraries
import matplotlib
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from matplotlib import style
%matplotlib inline

import seaborn as sns
sns.set(style="white", color_codes=True)
sns.set(font_scale=1.5)

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
from scipy.stats import normaltest
from scipy.stats import anderson

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate

# import libraries for metrics and reporting
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn import metrics
from statsmodels.tools.eval_measures import rmse
from sklearn.metrics import mean_squared_error,r2_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import learning_curve
```

```
[44]: location_train='train.csv'
```

```
[45]: df_train=pd.read_csv(location_train)
```

```
[46]: df_train.shape
```

```
[46]: (4209, 378)
```

```
[47]: df_train.head()
```

```
[47]:    ID       y  X0 X1  X2 X3 X4 X5 X6 X8  …  X375  X376  X377  X378  X379  \
      0   0  130.81   k  v  at  a  d  u  j  o  …     0     0     1     0     0
      1   6   88.53   k  t  av  e  d  y  l  o  …     1     0     0     0     0
      2   7   76.26  az  w   n  c  d  x  j  x  …     0     0     0     0     0
      3   9   80.62  az  t   n  f  d  x  l  e  …     0     0     0     0     0
      4  13   78.02  az  v   n  f  d  h  d  n  …     0     0     0     0     0

         X380  X382  X383  X384  X385
      0     0     0     0     0     0
      1     0     0     0     0     0
      2     0     1     0     0     0
      3     0     0     0     0     0
      4     0     0     0     0     0

      [5 rows x 378 columns]
```

```
[48]: df_train.dtypes
```

```
[48]: ID        int64
      y       float64
      X0       object
      X1       object
      X2       object
                …
      X380      int64
      X382      int64
      X383      int64
      X384      int64
      X385      int64
      Length: 378, dtype: object
```

```
[49]: df_train.columns[10:]
```

```
[49]: Index(['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
              …
              'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
              'X385'],
             dtype='object', length=368)
```

```
[50]: np.unique(df_train[df_train.columns[10:]])
```

```
[50]: array([0, 1])
```

```
[51]: df_train.isnull().sum()
```

```
[51]: ID      0
      y       0
      X0      0
      X1      0
      X2      0
             ..
      X380    0
      X382    0
      X383    0
      X384    0
      X385    0
      Length: 378, dtype: int64
```

```
[52]: num=['int16','int32','int64','float16','float32','float64']
      obj=['O']
```

```
[53]: df_train_num=df_train.select_dtypes(include=num)
      df_train_cat=df_train.select_dtypes(include=obj)
      print(df_train_num.columns)
      print(df_train_cat.columns)
```

```
Index(['ID', 'y', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=370)
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')
```

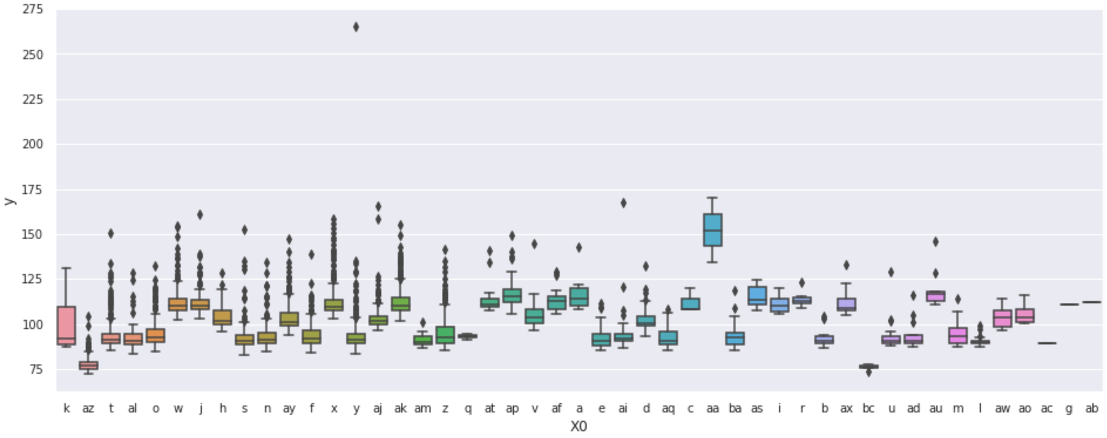
```
[54]: for col_name in df_train_cat.columns:
          print('unique values in'+col_name+'are',df_train_cat[col_name].nunique())
          print(df_train_cat[col_name].unique())
```

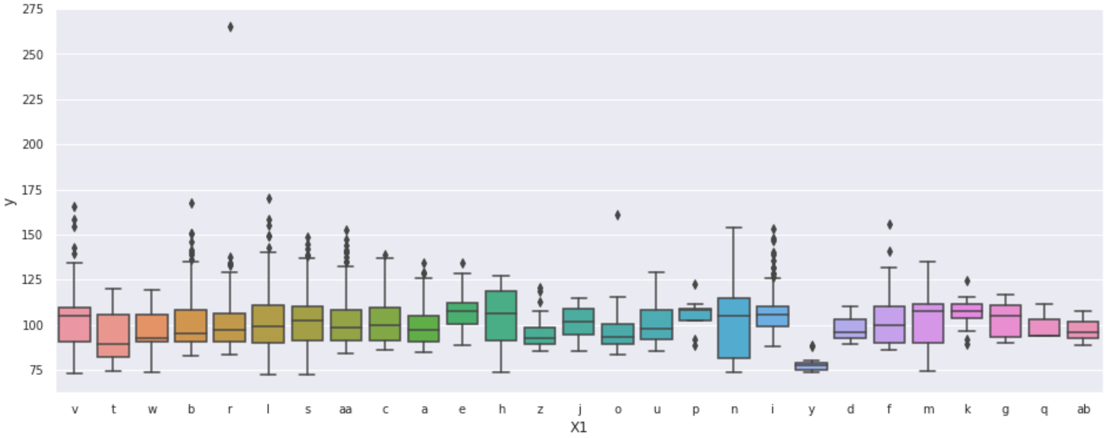
```
unique values inX0are 47
['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'ak' 'am'
 'z' 'q' 'at' 'ap' 'v' 'af' 'a' 'e' 'ai' 'd' 'aq' 'c' 'aa' 'ba' 'as' 'i'
 'r' 'b' 'ax' 'bc' 'u' 'ad' 'au' 'm' 'l' 'aw' 'ao' 'ac' 'g' 'ab']
```

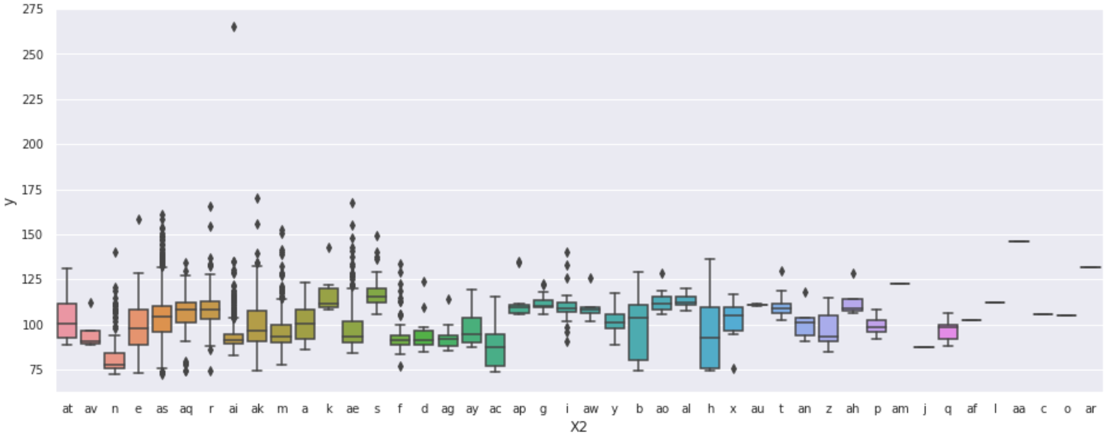
```
unique values inX1are 27
['v' 't' 'w' 'b' 'r' 'l' 's' 'aa' 'c' 'a' 'e' 'h' 'z' 'j' 'o' 'u' 'p' 'n'
 'i' 'y' 'd' 'f' 'm' 'k' 'g' 'q' 'ab']
unique values inX2are 44
['at' 'av' 'n' 'e' 'as' 'aq' 'r' 'ai' 'ak' 'm' 'a' 'k' 'ae' 's' 'f' 'd'
 'ag' 'ay' 'ac' 'ap' 'g' 'i' 'aw' 'y' 'b' 'ao' 'al' 'h' 'x' 'au' 't' 'an'
 'z' 'ah' 'p' 'am' 'j' 'q' 'af' 'l' 'aa' 'c' 'o' 'ar']
unique values inX3are 7
['a' 'e' 'c' 'f' 'd' 'b' 'g']
unique values inX4are 4
['d' 'b' 'c' 'a']
unique values inX5are 29
['u' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac' 'ad' 'ae'
 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
unique values inX6are 12
['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']
unique values inX8are 25
['o' 'x' 'e' 'n' 's' 'a' 'h' 'p' 'm' 'k' 'd' 'i' 'v' 'j' 'b' 'q' 'w' 'g'
 'y' 'l' 'f' 'u' 'r' 't' 'c']
```

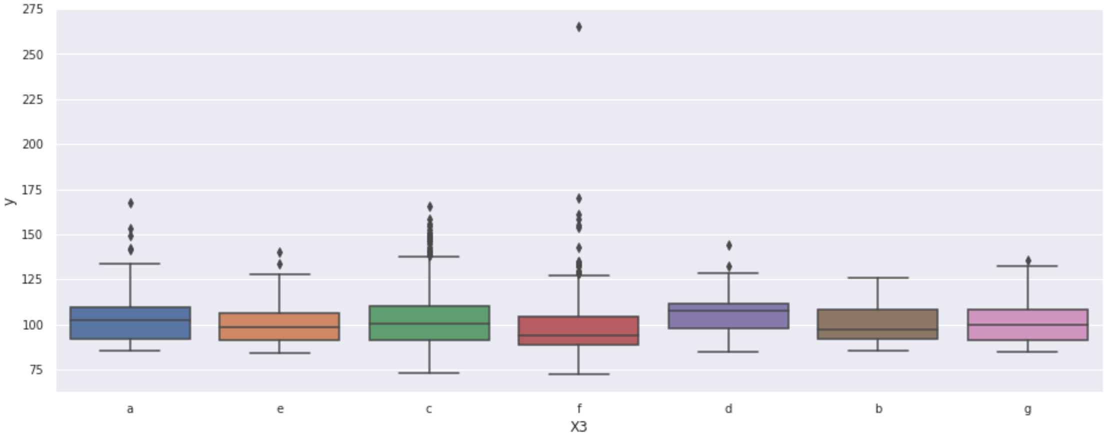
[55]:

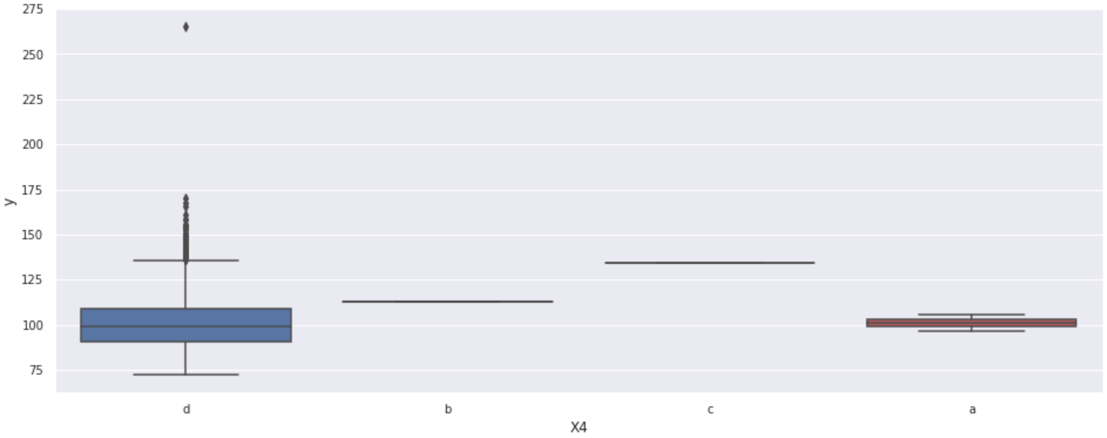
```
cols=['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']

for col in cols:
    plt.figure(figsize=(16,6))
    sns.boxplot(x=col,y='y',data=df_train)
    plt.xlabel(col,fontsize=12)
    plt.ylabel('y',fontsize=12)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
```

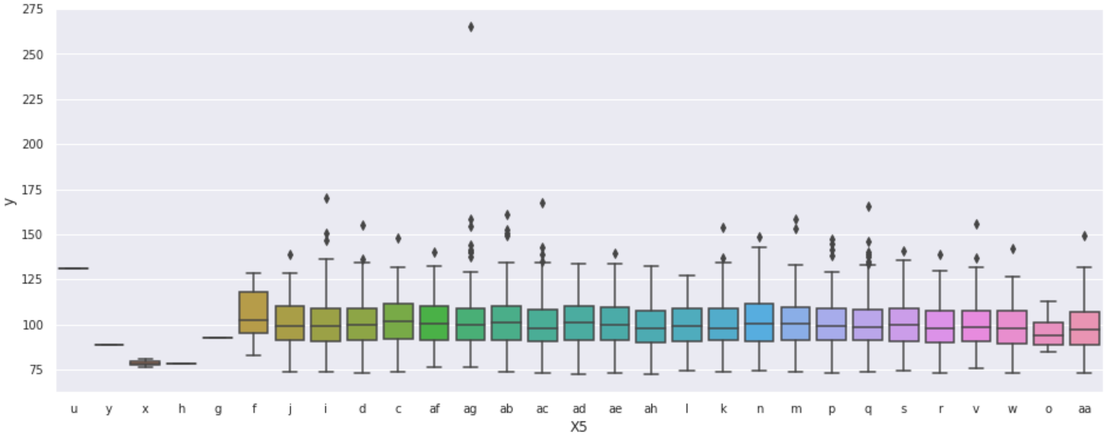


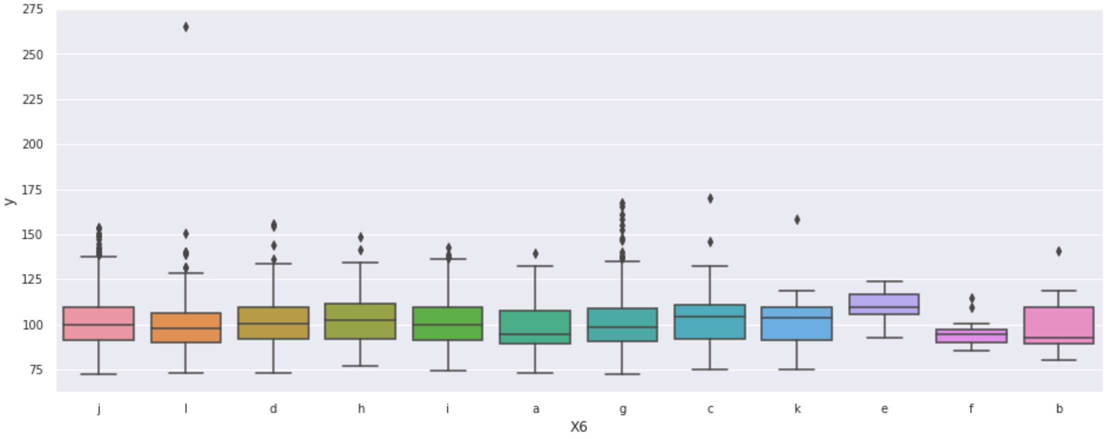

y
275
250
225
200
175
150
125
100
75
v t w b r l s aa c a e h z j o u p n i y d f m k g q ab
X1

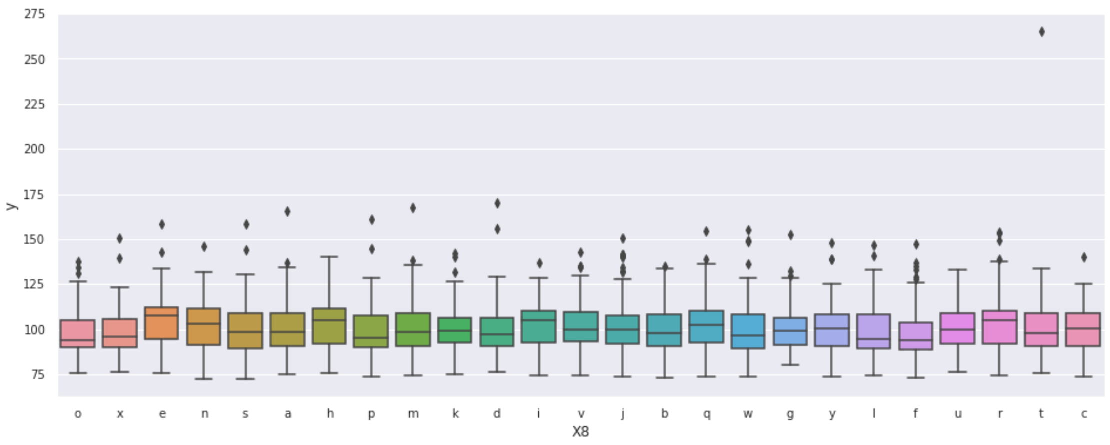

y
275
250
225
200
175
150
125
100
75
at av n e as aq r ai ak m a k ae s f d ag ay ac ap g i aw y b ao al h x au t an z ah p am j q af l aa c o ar
X2


y
275
250
225
200
175
150
125
100
75
a e c f d b g
X3


275
250
225
200
175
150
125
100
75
y
d
b
c
a
X4


275
250
225
200
175
150
125
100
75
y
u
y
x
h
g
f
j
i
d
c
af
ag
ab
ac
ad
ae
ah
l
k
n
m
p
q
s
r
v
w
o
aa
X5


275
250
225
200
175
150
125
100
75
y
j
l
d
h
i
a
g
c
k
e
f
b
X6



```
[56]: df_train.head()
```

```
[56]:    ID       y  X0 X1  X2 X3 X4 X5 X6 X8  …  X375  X376  X377  X378  X379  \
      0   0  130.81   k  v  at  a  d  u  j  o  …     0     0     1     0     0
      1   6   88.53   k  t  av  e  d  y  l  o  …     1     0     0     0     0
      2   7   76.26  az  w   n  c  d  x  j  x  …     0     0     0     0     0
      3   9   80.62  az  t   n  f  d  x  l  e  …     0     0     0     0     0
      4  13   78.02  az  v   n  f  d  h  d  n  …     0     0     0     0     0

         X380  X382  X383  X384  X385
      0     0     0     0     0     0
      1     0     0     0     0     0
      2     0     1     0     0     0
      3     0     0     0     0     0
      4     0     0     0     0     0

      [5 rows x 378 columns]
```

```
[57]: import statsmodels.api as sm
      from statsmodels.formula.api import ols
      model = ols('y ~C(X4)', data=df_train).fit()
      print('F-statistic : ', model.fvalue)
      print('p-value     : ', model.f_pvalue)
```

```
F-statistic :  2.6188965213725144
p-value     :  0.04920919630464415
```

```
[58]: anova_table = sm.stats.anova_lm(model, typ=2)
      anova_table
```

```
[58]:                  sum_sq     df         F    PR(>F)
      C(X4)       1261.638003    3.0  2.618897  0.049209
```

```
Residual  675244.676340  4205.0       NaN       NaN
```

```
[59]: colnames = ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
      for colname in colnames:
          model = ols('y ~ ' + colname , data=df_train).fit()
          print('column : {}, F-statistic : {:6.2f}, p-value : {:6.2f}'.
       ↪format(colname, model.fvalue, model.f_pvalue))
```

```
column : X0, F-statistic : 122.31, p-value :   0.00
column : X1, F-statistic :   6.99, p-value :   0.00
column : X2, F-statistic :  28.26, p-value :   0.00
column : X3, F-statistic :  30.99, p-value :   0.00
column : X4, F-statistic :   2.62, p-value :   0.05
column : X5, F-statistic :   2.15, p-value :   0.00
column : X6, F-statistic :   4.18, p-value :   0.00
column : X8, F-statistic :   5.03, p-value :   0.00
```

```
[60]: dtype_df = df_train.dtypes.reset_index()
      dtype_df.head(3)
```

```
[60]:   index        0
      0    ID    int64
      1     y  float64
      2    X0   object
```

```
[61]: dtype_df.columns = ["Count", "Column Type"]
      dtype_df.head(3)
```

```
[61]:   Count Column Type
      0    ID       int64
      1     y     float64
      2    X0      object
```

```
[62]: dtype_df[dtype_df['Column Type'] == 'int64'].head(6)
```

```
[62]:    Count Column Type
      0     ID       int64
      10   X10       int64
      11   X11       int64
      12   X12       int64
      13   X13       int64
      14   X14       int64
```

```
[63]: train = df_train_num
      train.head()
```

[63]:
```
   ID       y  X10  X11  X12  X13  X14  X15  X16  X17  …  X375  X376  X377  \
0   0  130.81    0    0    0    1    0    0    0    0  …     0     0     1
1   6   88.53    0    0    0    0    0    0    0    0  …     1     0     0
2   7   76.26    0    0    0    0    0    0    0    1  …     0     0     0
3   9   80.62    0    0    0    0    0    0    0    0  …     0     0     0
4  13   78.02    0    0    0    0    0    0    0    0  …     0     0     0

   X378  X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0     0
1     0     0     0     0     0     0     0
2     0     0     0     1     0     0     0
3     0     0     0     0     0     0     0
4     0     0     0     0     0     0     0

[5 rows x 370 columns]
```

[64]:
```
X_train_1 = train.drop(['y', 'ID'], axis=1)
y_train_1 = train.y
X_train, X_test, y_train, y_test = train_test_split(X_train_1,
                                                    y_train_1,
                                                    test_size=0.25,
                                                    random_state=4)
```

[66]:
```
#linear regression

linreg = LinearRegression()
```

[67]:
```
%%time
linreg.fit(X_train, y_train)
```

```
CPU times: user 236 ms, sys: 208 ms, total: 444 ms
Wall time: 286 ms
```

[67]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[68]:
```
y_pred = linreg.predict(X_train)

print("\nTraining score :")
print("Mean squared error: %.2f"% mean_squared_error(y_train, y_pred))
print('R2 score: %2f' % r2_score(y_train, y_pred))

#predicting testing samples

y_pred = linreg.predict(X_test)


print("\nTesting score :")
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))
print('R2 score: %2f' % r2_score(y_test, y_pred))
```

```
Training score :
Mean squared error: 65.91
R2 score: 0.595189

Testing score :
Mean squared error: 7918259811495546552582144.00
R2 score: -51272718979833708675072.000000
```

[69]:
```
#Random forest regressor
rf_reg = RandomForestRegressor(criterion= 'mse',
                               max_depth= 4,
                               max_features= 'auto',
                               min_samples_split= 0.05,
                               n_estimators= 20)
```

[70]:
```
%%time

rf_reg.fit(X_train, y_train)
```

```
CPU times: user 264 ms, sys: 0 ns, total: 264 ms
Wall time: 339 ms
```

[70]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=4, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=0.05, min_weight_fraction_leaf=0.0,
                      n_estimators=20, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

[71]:
```
%%time

y_pred = rf_reg.predict(X_train)

print("\nTraining score :")
print("Mean squared error: %.2f"% mean_squared_error(y_train, y_pred))
print('R2 score: %2f' % r2_score(y_train, y_pred))

#predicting testing samples

y_pred = rf_reg.predict(X_test)
```

```
print("\nTesting score :")
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))
print('R2 score: %2f' % r2_score(y_test, y_pred))
```

```
Training score :
Mean squared error: 68.44
R2 score: 0.579699

Testing score :
Mean squared error: 67.23
R2 score: 0.564656
CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 17.1 ms
```

[72]:

```
#knn regressor
from sklearn.neighbors import KNeighborsRegressor

Knn = KNeighborsRegressor(n_neighbors=17,
                          metric= 'hamming',
                          weights= 'uniform',
                          algorithm='brute')

Knn.fit(X_train, y_train)
```

[72]:

```
KNeighborsRegressor(algorithm='brute', leaf_size=30, metric='hamming',
                    metric_params=None, n_jobs=None, n_neighbors=17, p=2,
                    weights='uniform')
```

[73]:

```
%%time

y_pred = Knn.predict(X_train)

print("\nTraining score :")
print("Mean squared error: %.2f"% mean_squared_error(y_train, y_pred))
print('R2 score: %2f' % r2_score(y_train, y_pred))

#predicting testing samples

y_pred = Knn.predict(X_test)


print("\nTesting score :")
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))
print('R2 score: %2f' % r2_score(y_test, y_pred))
```

```
Training score :
```

```
Mean squared error: 79.15
R2 score: 0.513879

Testing score :
Mean squared error: 85.52
R2 score: 0.446230
CPU times: user 3.45 s, sys: 16 ms, total: 3.46 s
Wall time: 3.52 s
```

[34]:

```
pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/site-packages
(1.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/site-packages
(from xgboost) (1.18.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/site-packages
(from xgboost) (1.4.1)
WARNING: You are using pip version 20.0.2; however, version 20.1.1 is
available.
You should consider upgrading via the '/usr/local/bin/python3.7 -m pip install
--upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

[74]:

```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error
X_train, X_test, y_train, y_test = train_test_split(X_train_1,
                                                    y_train_1,
                                                    test_size=0.2,
                                                    random_state=123)
```

[75]:

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
mean_train = y_train.mean()
```

[76]:

```
#predictions on test side
baseline_predictions = np.ones(y_test.shape) * mean_train
```

[77]:

```
#MAE
mae_baseline = mean_absolute_error(y_test, baseline_predictions)
print("Baseline MAE is {: .2f}" .format(mae_baseline))
```

```
Baseline MAE is  10.07
```

[78]:

```
#params dictionary
xgb_params = {
    'max_depth': 8,
    'min_child_weight' : 1,
    'eta' : .35,
    'subsample' : 1,
    'colsample_bytree' : .9,
    'objective' : 'reg:squarederror',
    'reg_alpha' :4,
    #'reg_lambda' : 45,
    'eval_metric' : 'mae',
    'validate_parameters' : 1,
    'verbose_eval' : False
}
```

[79]:

```
%%time
model = xgb.train(
                xgb_params,
                dtrain,
                num_boost_round=999,
                evals=[(dtest, "Test")],
                early_stopping_rounds=10
)
```

```
[11:56:29] WARNING: /workspace/src/learner.cc:328:
Parameters: { verbose_eval } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


[0]     Test-mae:65.03382
Will train until Test-mae hasn't improved in 10 rounds.
[1]     Test-mae:42.25439
[2]     Test-mae:27.43620
[3]     Test-mae:17.79237
[4]     Test-mae:11.53217
[5]     Test-mae:7.55955
[6]     Test-mae:5.55699
[7]     Test-mae:4.96163
[8]     Test-mae:4.86904
[9]     Test-mae:4.94027
[10]    Test-mae:5.05144
[11]    Test-mae:5.13235
[12]    Test-mae:5.19993
[13]    Test-mae:5.22284
```

```
[14]    Test-mae:5.24831
[15]    Test-mae:5.29146
[16]    Test-mae:5.31442
[17]    Test-mae:5.39554
[18]    Test-mae:5.41576
Stopping. Best iteration:
[8]     Test-mae:4.86904

CPU times: user 2.6 s, sys: 0 ns, total: 2.6 s
Wall time: 1.32 s
```

[80]:

```
y_pred = model.predict(dtrain)

print("Training : metrics..")
print('Mean Abs Error MAE   : ', metrics.mean_absolute_error(y_train, y_pred))
print('Mean Sq Error MSE    : ', metrics.mean_squared_error(y_train, y_pred))

print('Root Mean Sq Error RMSE : ', np.sqrt(metrics.mean_squared_error(y_train,␣
 ↪y_pred)))

print('r2 value                : ', metrics.r2_score(y_train, y_pred))

y_pred = model.predict(dtest)

print('\n')

print("Training : metrics..")
print('Mean Abs Error MAE   : ', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Sq Error MSE    : ', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Sq Error RMSE : ', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))

print('r2 value                : ', metrics.r2_score(y_test, y_pred))
```

```
Training : metrics..
Mean Abs Error MAE   :  4.676936372961459
Mean Sq Error MSE    :  55.778860648861276
Root Mean Sq Error RMSE :  7.468524663470107
r2 value                :  0.6589438699760 47


Training : metrics..
Mean Abs Error MAE   :  5.415756630795586
Mean Sq Error MSE    :  59.47122115634578
Root Mean Sq Error RMSE :  7.711758629284619
r2 value                :  0.6020376950354267
```

[ ]:

```
```