GRADIENT

August 29, 2021

[1]:
```
# Warning Libraries :
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# Scientific and Data Manipulation Libraries :
import pandas as pd
import numpy as np
from numpy import percentile
import math
import os
from sklearn.model_selection import train_test_split
```

[3]:
```
# Data Visualization Libraries :
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
```

[4]:
```
#Libraries to convert .las files to .csv and merge

import lasio
import glob  ##For merging csv files
```

[5]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import KNNImputer
from sklearn.linear_model import LinearRegression
```

[6]:
```
#Feature Selection Libraries
from sklearn.feature_selection import VarianceThreshold
```

[7]:
```
#SCALING LIBRARIES
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer,␣
↪RobustScaler, MaxAbsScaler
```

[8]:
```
#MODEL TRAINING LIBRARIES
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from catboost import CatBoostRegressor
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
```

[9]:
```
#MODEL ACCURACY LIBRARIES
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

[12]:
```
path='/media/mr-robot/Local Disk/summer_training/Train'
os.chdir(path)
```

[13]:
```
df = pd.read_csv('merged_data.csv')
df
```

[13]:

	DEPTH	ACOUSTICIMPEDANCE1	AI	AVG_PIGN	CALI	\
0	1295.9144	4834.3213	4834321.0	NaN	9.1419	
1	1296.0668	4751.9272	4751927.0	NaN	9.2247	
2	1296.2192	4777.4341	4777434.5	NaN	9.2680	
3	1296.3716	4810.3301	4810330.0	NaN	9.2766	
4	1296.5240	4827.2563	4827256.5	NaN	9.2866	
…	…	…	…	…		
58494	1622.6028	6069.1309	6069130.5	NaN	8.5257	
58495	1622.7552	6067.8120	6067812.0	NaN	8.5282	
58496	1622.9076	6105.7729	6105773.0	NaN	8.5313	
58497	1623.0600	6152.9897	6152977.5	NaN	8.5331	
58498	1623.2124	6157.8291	6157829.5	NaN	8.5338	

	CALI[DERIVED]1	DFL	DT	FACIES	FLD1	…	CALI_1	NPHI_1	\
0	9.1419	1.0697	137.8066	NaN	NaN	…	NaN	NaN	
1	9.2247	1.2028	139.5873	0.0	NaN	…	NaN	NaN	
2	9.2680	1.2145	140.0185	0.0	NaN	…	NaN	NaN	
3	9.2766	1.0487	139.3474	0.0	NaN	…	NaN	NaN	
4	9.2866	0.9479	138.8638	0.0	NaN	…	NaN	NaN	
…	…	…	…	…	…	…	…		
58494	NaN	NaN	123.7404	NaN	NaN	…	NaN	0.4993	
58495	NaN	NaN	123.8728	NaN	NaN	…	NaN	0.5313	
58496	NaN	NaN	123.3722	NaN	NaN	…	NaN	0.5448	
58497	NaN	NaN	122.6038	NaN	NaN	…	NaN	0.5364	
58498	NaN	NaN	122.3045	NaN	NaN	…	NaN	0.5331	

	ZCOR	RHOB_1	RXO	SPDH	DTDS	M2R1	TH	U
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
4       NaN     NaN  NaN   NaN       NaN     NaN NaN NaN
…     …     …  …   …       …     …   ..   ..
58494   NaN  2.4639  NaN   NaN  123.7404  1.5970 NaN NaN
58495   NaN  2.4660  NaN   NaN  123.8728  1.6128 NaN NaN
58496   NaN  2.4714  NaN   NaN  123.3722  1.7043 NaN NaN
58497   NaN  2.4750  NaN   NaN  122.6038  1.8375 NaN NaN
58498   NaN  2.4709  NaN   NaN  122.3045  1.9363 NaN NaN

[58499 rows x 66 columns]
```

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58499 entries, 0 to 58498
Data columns (total 66 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   DEPTH                58499 non-null  float64
 1   ACOUSTICIMPEDANCE1   58499 non-null  float64
 2   AI                   55259 non-null  float64
 3   AVG_PIGN             323 non-null    float64
 4   CALI                 54981 non-null  float64
 5   CALI[DERIVED]1       44090 non-null  float64
 6   DFL                  23458 non-null  float64
 7   DT                   58499 non-null  float64
 8   FACIES               52641 non-null  float64
 9   FLD1                 3963 non-null   float64
 10  GR                   58379 non-null  float64
 11  HDRS                 26951 non-null  float64
 12  HMRS                 26951 non-null  float64
 13  DEPTH_1              50885 non-null  float64
 14  NPHI                 58172 non-null  float64
 15  ONE-WAYTIME1         15713 non-null  float64
 16  PERF_INT             1569 non-null   float64
 17  PERMEABILITY         28149 non-null  float64
 18  PIGN                 46949 non-null  float64
 19  PIGN_MODELLING       51101 non-null  float64
 20  PIMP                 55259 non-null  float64
 21  RHOB                 58499 non-null  float64
 22  RT_MODELLING         53629 non-null  float64
 23  RT_POWER             51379 non-null  float64
 24  SP                   55992 non-null  float64
 25  SUWI                 46947 non-null  float64
 26  SUWI_MODELLING       51099 non-null  float64
 27  TDVSS                58437 non-null  float64
 28  VCL                  46947 non-null  float64
 29  WATER_VOL            43735 non-null  float64
 30  ZLT                  44562 non-null  float64
```

```
 31  LLD                44942 non-null  float64
 32  LLS                27394 non-null  float64
 33  LL3                12373 non-null  float64
 34  BS                 6706 non-null   float64
 35  CALI1              2389 non-null   float64
 36  DEVI               10283 non-null  float64
 37  DT1                6130 non-null   float64
 38  PHIT               16532 non-null  float64
 39  PIGE               5245 non-null   float64
 40  LLD_1              9518 non-null   float64
 41  SXWI               27938 non-null  float64
 42  PEF                19419 non-null  float64
 43  AZI1               2487 non-null   float64
 44  TEMP               14514 non-null  float64
 45  DRES               2765 non-null   float64
 46  DT2                2765 non-null   float64
 47  DT4P               5854 non-null   float64
 48  GR_EDTC            2765 non-null   float64
 49  M2R2               8568 non-null   float64
 50  LLS_1              238 non-null    float64
 51  MSFL               2765 non-null   float64
 52  PR                 2757 non-null   float64
 53  TENS               2765 non-null   float64
 54  VPVS               2757 non-null   float64
 55  BIT                5553 non-null   float64
 56  CALI_1             2999 non-null   float64
 57  NPHI_1             10811 non-null  float64
 58  ZCOR               2998 non-null   float64
 59  RHOB_1             10899 non-null  float64
 60  RXO                1552 non-null   float64
 61  SPDH               3069 non-null   float64
 62  DTDS               2546 non-null   float64
 63  M2R1               2546 non-null   float64
 64  TH                 2509 non-null   float64
 65  U                  2509 non-null   float64
dtypes: float64(66)
memory usage: 29.5 MB
```

[15]:
```
#Selecting required feature
df=df[["GR","RHOB","NPHI","DT"]]
```

[16]:
```
df.isnull().sum()
```

[16]:
```
GR      120
RHOB      0
NPHI    327
DT        0
```

```
dtype: int64
```

[17]:
```
df= df.dropna(axis=0)
```

[18]:
```
df
```

[18]:
```
             GR    RHOB    NPHI        DT
0       61.3278  2.1857  0.5643  137.8066
1       61.9954  2.1762  0.5611  139.5873
2       63.5188  2.1946  0.5630  140.0185
3       64.9925  2.1992  0.5677  139.3474
4       65.6985  2.1992  0.5743  138.8638
…          …      …       …         …
58461   82.2480  2.6072  0.5111  110.8313
58462   81.6189  2.5490  0.5079  110.6059
58463   82.5907  2.4944  0.4909  113.7010
58464   83.2526  2.4870  0.4823  116.2950
58465   82.9096  2.5198  0.4803  115.6295

[58097 rows x 4 columns]
```

[19]:
```
x = df.drop("DT",1)
y = df["DT"]
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=4)
```

[20]:
```
X_train.shape
```

[20]:
```
(46477, 3)
```

[21]:
```
def outliers(dataConditioningStrategy,dataframe, y_dataframe,␣
↪dataconditioningcolumns):
    df=dataframe
    df["y"]=y_dataframe
    if dataConditioningStrategy == "3_Standard_Deviation":
        for column in dataconditioningcolumns:
            print("column",column )
            upperlimit = df[column].mean() + 3*df[column].std()
            lowerlimit = df[column].mean() - 3*df[column].std()

            print("3 standard deviation outliers -:")
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)])
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)].
↪shape)
            df= df[(df[column] < upperlimit) & (df[column] > lowerlimit)]
            print(df)
```

```
    elif dataConditioningStrategy == "4_Standard_Deviation":
        for column in dataconditioningcolumns:
            print("column",column )
            upperlimit = df[column].mean() + 4*df[column].std()
            lowerlimit = df[column].mean() - 4*df[column].std()

            print("4 standard deviation outliers -:")
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)])
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)].
↪shape)
            df= df[(df[column] < upperlimit) & (df[column] > lowerlimit)]
            print(df)

    elif dataConditioningStrategy == "InterquartileRange":
        for column in dataconditioningcolumns:
            print("column",column )
            q25, q75 = percentile(df[column], 25), percentile(df[column], 75)
            iqr = q75 - q25
            print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75,␣
↪iqr))
            cut_off = iqr * 1.5
            lowerlimit, upperlimit = q25 - cut_off, q75 + cut_off

            print("InterQuartile Range Outliers-:")
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)])
            print(df[(df[column] > upperlimit) | (df[column] < lowerlimit)].
↪shape)
            df= df[(df[column] < upperlimit) & (df[column] > lowerlimit)]
            print(df)


    return df.drop("y",axis=1) , df["y"]
```

[22]:
```
DATAConditioningStrategy =␣
↪["3_Standard_Deviation","4_Standard_Deviation","InterquartileRange"]
DATAConditioningColumns = ["GR","RHOB","NPHI"]
optionoutlier = 0
X_train,y_train = outliers(DATAConditioningStrategy[optionoutlier] , X_train ,␣
↪y_train, DATAConditioningColumns)
```

```
column GR
3 standard deviation outliers -:
             GR    RHOB    NPHI         y
38872  177.2283  2.6526  0.5631  159.2438
37912  187.7777  2.4635  0.5864  128.4088
39003  185.1136  2.4443  0.6586  147.0338
37685  166.0200  2.3162  0.6634  131.6756
```

```
39289  167.0888  1.8879  0.8120  143.4888
…         …      …       …         …
37868  171.6167  2.4143  0.5227  103.8630
39152  177.3839  1.9921  0.7775  145.7015
37412  169.6837  2.4180  0.6515  156.8676
39209  177.0399  2.0120  0.7306  132.1170
38963  193.1186  2.5238  0.6102  109.3014

[1149 rows x 4 columns]
(1149, 4)
             GR    RHOB    NPHI         y
7174    54.9827  2.4818  0.5497  100.8784
34641   95.0442  2.5565  0.5258  101.1751
48215   69.2090  2.3328  0.5124  106.7575
18175   67.8533  2.4396  0.6228  119.8530
50056   88.0100  2.4424  0.4396  114.9634
…         …      …       …         …
55488  103.1246  2.5150  0.4686   98.6188
50169   84.2108  2.3961  0.4774  108.7165
27063   58.8217  2.4845  0.5033  103.9533
8366    69.2729  2.0863  0.6274  147.9099
17530   68.4194  2.2210  0.3759  107.0126

[45328 rows x 4 columns]
column RHOB
3 standard deviation outliers -:
            GR    RHOB    NPHI         y
30074  59.2786  1.0765  0.5302   41.9701
52295  21.2960  1.1319  0.6436  145.1185
45811  20.5741  1.1716  0.5891  151.3362
58023  29.6655  1.1436  0.6993  152.1691
24979  12.7422  1.1715  0.9076  152.7837
…        …      …       …         …
48327  16.0175  1.1652  0.6124  148.6804
1078   20.4708  1.1724  0.9567  147.1638
30067  52.1434  0.9053  0.6049   45.4202
50900  19.0558  1.1364  0.7113  150.3924
30205  51.0205  1.0330  0.5134   39.0167

[676 rows x 4 columns]
(676, 4)
             GR    RHOB    NPHI         y
7174    54.9827  2.4818  0.5497  100.8784
34641   95.0442  2.5565  0.5258  101.1751
48215   69.2090  2.3328  0.5124  106.7575
18175   67.8533  2.4396  0.6228  119.8530
50056   88.0100  2.4424  0.4396  114.9634
…         …      …       …         …
```

```
55488  103.1246  2.5150  0.4686   98.6188
50169   84.2108  2.3961  0.4774  108.7165
27063   58.8217  2.4845  0.5033  103.9533
8366    69.2729  2.0863  0.6274  147.9099
17530   68.4194  2.2210  0.3759  107.0126

[44652 rows x 4 columns]
column NPHI
3 standard deviation outliers -:
             GR    RHOB    NPHI         y
14548   10.9182  1.2070  1.0210  148.2126
52087   36.8205  2.2072  0.1976   95.5439
23781   31.1632  1.4102  0.8840  140.0048
24997   15.3672  1.2002  0.9074  153.3714
37306  138.7086  1.6408  0.9094  133.6813
…         …      …       …        …
18917   22.7157  1.2346  0.9733  150.0270
19123   19.1466  1.2230  0.9875  148.9810
20220    0.0000  2.1634 -0.0380  125.1803
1732    13.2997  1.1853  0.9820  152.9783
37290  118.3981  1.4250  1.0000  147.8758

[720 rows x 4 columns]
(720, 4)
             GR    RHOB    NPHI         y
7174    54.9827  2.4818  0.5497  100.8784
34641   95.0442  2.5565  0.5258  101.1751
48215   69.2090  2.3328  0.5124  106.7575
18175   67.8533  2.4396  0.6228  119.8530
50056   88.0100  2.4424  0.4396  114.9634
…         …      …       …        …
55488  103.1246  2.5150  0.4686   98.6188
50169   84.2108  2.3961  0.4774  108.7165
27063   58.8217  2.4845  0.5033  103.9533
8366    69.2729  2.0863  0.6274  147.9099
17530   68.4194  2.2210  0.3759  107.0126

[43932 rows x 4 columns]
```

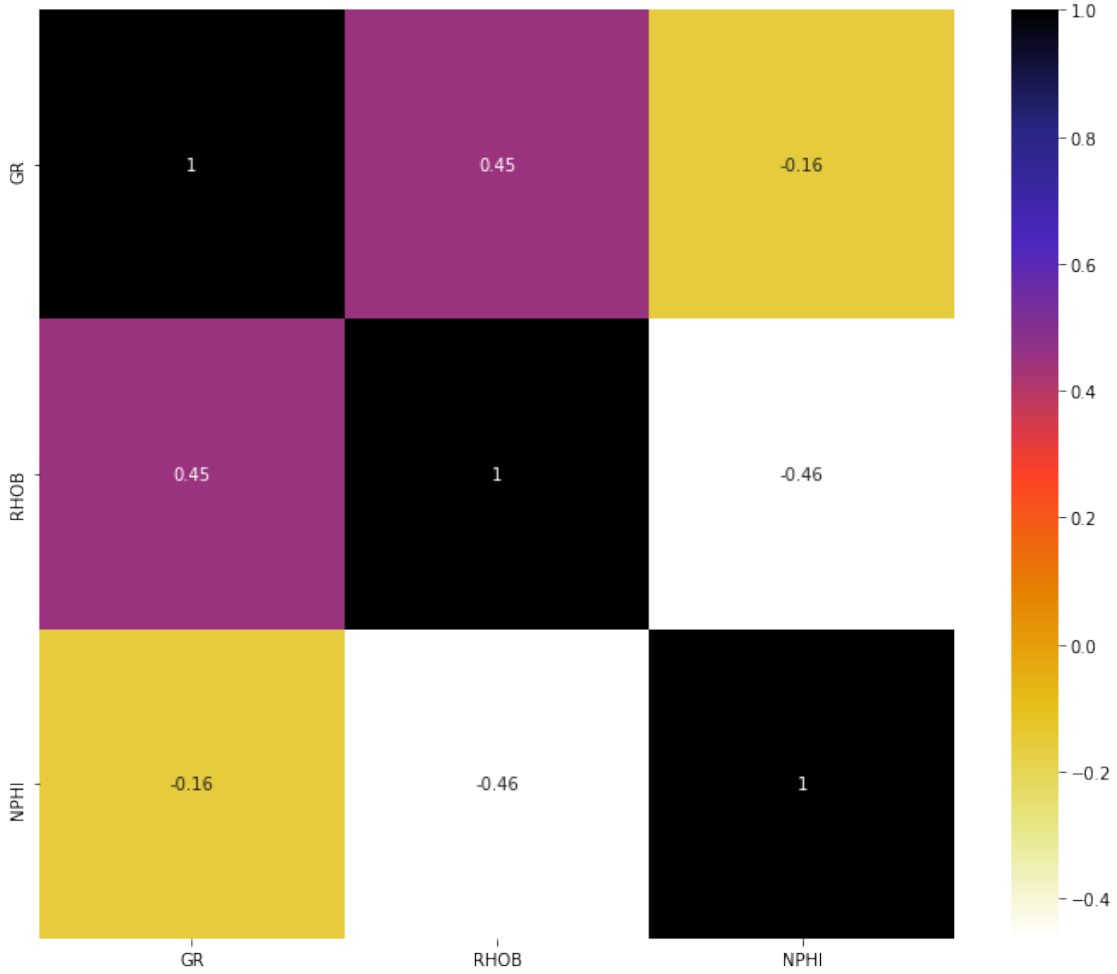
[23]:
```
X_train
```

[23]:
```
             GR    RHOB    NPHI
7174    54.9827  2.4818  0.5497
34641   95.0442  2.5565  0.5258
48215   69.2090  2.3328  0.5124
18175   67.8533  2.4396  0.6228
50056   88.0100  2.4424  0.4396
```

```
…          …       …       …
55488   103.1246   2.5150   0.4686
50169    84.2108   2.3961   0.4774
27063    58.8217   2.4845   0.5033
8366     69.2729   2.0863   0.6274
17530    68.4194   2.2210   0.3759

[43932 rows x 3 columns]
```

[24]:
```
plt.figure(figsize=(12,10))
cor = X_train.corr()
sns.heatmap(cor , annot=True , cmap=plt.cm.CMRmap_r)
plt.show()
```



[25]:
```
X_train.var()
```

[25]:
```
GR      621.097210
RHOB      0.093920
NPHI      0.009019
dtype: float64
```

[26]:
```
X_train.corr()
```

[26]:
```
            GR      RHOB      NPHI
GR    1.000000  0.447292 -0.157540
RHOB  0.447292  1.000000 -0.463009
NPHI -0.157540 -0.463009  1.000000
```

[27]:
```
X_train['NPHI2'] = 2 * X_train.NPHI
X_train.drop('NPHI',1)
```

[27]:
```
             GR    RHOB   NPHI2
7174    54.9827  2.4818  1.0994
34641   95.0442  2.5565  1.0516
48215   69.2090  2.3328  1.0248
18175   67.8533  2.4396  1.2456
50056   88.0100  2.4424  0.8792
…         …      …       …
55488  103.1246  2.5150  0.9372
50169   84.2108  2.3961  0.9548
27063   58.8217  2.4845  1.0066
8366    69.2729  2.0863  1.2548
17530   68.4194  2.2210  0.7518

[43932 rows x 3 columns]
```

[28]:
```
X_train
```

[28]:
```
             GR    RHOB    NPHI   NPHI2
7174    54.9827  2.4818  0.5497  1.0994
34641   95.0442  2.5565  0.5258  1.0516
48215   69.2090  2.3328  0.5124  1.0248
18175   67.8533  2.4396  0.6228  1.2456
50056   88.0100  2.4424  0.4396  0.8792
…         …      …       …       …
55488  103.1246  2.5150  0.4686  0.9372
50169   84.2108  2.3961  0.4774  0.9548
27063   58.8217  2.4845  0.5033  1.0066
8366    69.2729  2.0863  0.6274  1.2548
17530   68.4194  2.2210  0.3759  0.7518

[43932 rows x 4 columns]
```

def data_scaling(scaling_strategy , scaling_data , scaling_columns):

```
if scaling_strategy =="RobustScaler" :
    scaling_data[scaling_columns] = RobustScaler().fit_transform(scaling_data[scaling_columns])

elif scaling_strategy =="MinMaxScaler" :
    scaling_data[scaling_columns] = MinMaxScaler().fit_transform(scaling_data[scaling_columns])

elif scaling_strategy == "StandardScaler" :
    scaling_data[scaling_columns] = StandardScaler().fit_transform(scaling_data[scaling_columns

else : # If any other scaling send by mistake still perform Robust Scalar
    scaling_data[scaling_columns] = RobustScaler().fit_transform(scaling_data[scaling_columns])

return scaling_data
```

scaling_strategy = [“RobustScaler”,“MinMaxScaler”,“StandardScaler”] optionscaling = 0 X_train = data_scaling(scaling_strategy[optionscaling] , X_train , X_train.columns)

[29]:
```
X_train
```

[29]:

	GR	RHOB	NPHI	NPHI2
7174	54.9827	2.4818	0.5497	1.0994
34641	95.0442	2.5565	0.5258	1.0516
48215	69.2090	2.3328	0.5124	1.0248
18175	67.8533	2.4396	0.6228	1.2456
50056	88.0100	2.4424	0.4396	0.8792
…	…	…	…	…
55488	103.1246	2.5150	0.4686	0.9372
50169	84.2108	2.3961	0.4774	0.9548
27063	58.8217	2.4845	0.5033	1.0066
8366	69.2729	2.0863	0.6274	1.2548
17530	68.4194	2.2210	0.3759	0.7518

[43932 rows x 4 columns]

[30]:
```
X_train.corr()
```

[30]:

	GR	RHOB	NPHI	NPHI2
GR	1.000000	0.447292	-0.157540	-0.157540
RHOB	0.447292	1.000000	-0.463009	-0.463009
NPHI	-0.157540	-0.463009	1.000000	1.000000
NPHI2	-0.157540	-0.463009	1.000000	1.000000

[31]:
```
X_test,y_test= outliers(DATAConditioningStrategy[optionoutlier] , X_test ,␣
↪y_test, DATAConditioningColumns)
```

```
column GR
3 standard deviation outliers -:
             GR    RHOB    NPHI         y
37849  175.6751  2.4016  0.5775  142.2255
```

```
37536  168.3488  2.2749  0.7082  132.1620
39217  184.6027  2.4462  0.5585  120.8720
38482  169.4325  2.3972  0.5975  122.9748
38892  210.6389  2.4015  0.6152  146.8278
…         …     …     …       …
38529  174.2404  2.6524  0.5968  131.5116
38438  175.3356  2.5883  0.5503  123.4890
38209  167.6908  2.2047  0.5728  119.7681
38614  215.1731  2.4510  0.5954  130.6578
37673  169.2242  2.3503  0.6405  126.5171

[268 rows x 4 columns]
(268, 4)
             GR    RHOB    NPHI         y
9485    81.7370  2.3038  0.6009  131.3419
17537   69.1520  2.2208  0.3879  111.5650
51851   96.9270  2.3463  0.3173  129.8816
51900   89.3109  2.4608  0.3544  109.1182
18016   59.3945  2.4996  0.5110  116.1016
…          …     …     …       …
36698  146.2150  2.1922  0.7058  143.0331
23619   78.3988  2.4924  0.5179  124.7717
22116   69.6871  2.4871  0.4940  122.9038
53717   84.2177  2.2280  0.5453  127.1126
22310   74.2334  2.5713  0.4707  100.7292

[11352 rows x 4 columns]
column RHOB
3 standard deviation outliers -:
            GR    RHOB    NPHI         y
55729  32.4591  1.0608  0.5852  147.9725
30169  52.0435  1.1398  0.5231   52.8476
58118  16.3385  1.1101  0.7049  153.0591
57970  15.8033  1.0989  0.6436  153.0778
30130  64.4212  1.0831  0.5529   36.3987
…         …     …     …       …
58129  17.0859  1.0973  0.6187  152.5417
55731  31.0793  1.0589  0.6465  145.9229
30133  63.9850  1.1125  0.5336   34.8224
29946  61.2440  0.9751  0.5561   33.2426
30137  61.1217  1.1144  0.5031   34.2003

[100 rows x 4 columns]
(100, 4)
             GR    RHOB    NPHI         y
9485    81.7370  2.3038  0.6009  131.3419
17537   69.1520  2.2208  0.3879  111.5650
51851   96.9270  2.3463  0.3173  129.8816
```

```
51900   89.3109  2.4608  0.3544  109.1182
18016   59.3945  2.4996  0.5110  116.1016
…          …     …       …          …
36698  146.2150  2.1922  0.7058  143.0331
23619   78.3988  2.4924  0.5179  124.7717
22116   69.6871  2.4871  0.4940  122.9038
53717   84.2177  2.2280  0.5453  127.1126
22310   74.2334  2.5713  0.4707  100.7292

[11252 rows x 4 columns]
column NPHI
3 standard deviation outliers -:
             GR    RHOB    NPHI         y
20206    0.0000  2.0817  1.0220  127.4388
14522   10.8924  1.2086  0.8916  148.2818
39056  136.9001  1.3953  0.9863  148.9868
38271  129.2833  1.6065  0.9657  177.8300
19495   14.4687  1.2134  0.9673  150.9600
…          …     …       …          …
38806  152.2032  1.4683  1.1107  139.0581
37285  138.8664  1.7244  0.8842  148.5310
26861   19.5026  1.2367  0.9996  149.9351
20239    0.0000  2.3781 -0.0242  123.8350
27506   18.4056  1.3308  1.0105  151.0704

[187 rows x 4 columns]
(187, 4)
             GR    RHOB    NPHI         y
9485    81.7370  2.3038  0.6009  131.3419
17537   69.1520  2.2208  0.3879  111.5650
51851   96.9270  2.3463  0.3173  129.8816
51900   89.3109  2.4608  0.3544  109.1182
18016   59.3945  2.4996  0.5110  116.1016
…          …     …       …          …
36698  146.2150  2.1922  0.7058  143.0331
23619   78.3988  2.4924  0.5179  124.7717
22116   69.6871  2.4871  0.4940  122.9038
53717   84.2177  2.2280  0.5453  127.1126
22310   74.2334  2.5713  0.4707  100.7292

[11065 rows x 4 columns]
```

X_test = data_scaling(scaling_strategy[optionscaling] , X_test , X_test.columns)

[32]:
```
X_test['NPHI2'] = 2 * X_test.NPHI
X_test.drop('NPHI',1)
```

[32]:

	GR	RHOB	NPHI2
9485	81.7370	2.3038	1.2018
17537	69.1520	2.2208	0.7758
51851	96.9270	2.3463	0.6346
51900	89.3109	2.4608	0.7088
18016	59.3945	2.4996	1.0220
…	…	…	…
36698	146.2150	2.1922	1.4116
23619	78.3988	2.4924	1.0358
22116	69.6871	2.4871	0.9880
53717	84.2177	2.2280	1.0906
22310	74.2334	2.5713	0.9414

[11065 rows x 3 columns]

[45]:
```
gr = GradientBoostingRegressor()
```

[46]:
```
gr.fit(X_train,y_train)
```

[46]:
```
GradientBoostingRegressor()
```

[47]:
```
y_pred = gr.predict(X_test)
```

[48]:
```
r2_score(y_test,y_pred)
```

[48]:
```
0.5785622631299989
```

[ ]:
```

```