A Project Report On

"Facies Classification and Synthetic Curve Generation in Well Logs using AI/ML based Data Science Workflow"

Submitted in partial fulfilment of the requirement for the award of degree of

B. Tech Hons. (Computer Science)

(Data Science & Artificial Intelligence)

of

Graphic Era Deemed to be University

Dehradun

Compiled by:

SPARSH SAXENA

Under the guidance of:

Mr. Vivek Gaurav Saini

(Sr. Programming Officer)

GEOPIC, ONGC, Dehradun

July 2021

Declaration

I hereby declare that the work presented in this project report entitled **"Facies Classification and Synthetic Curve Generation in Well Logs using AI/ML based Data Science Workflow"** in partial fulfilment of requirement for the award of degree of B.Tech (Computer Science), submitted in the department of Computer Science & Engineering, Graphic Era Deemed to be University, Dehradun is an authentic record of my work carried out during the Summer Training from 28 June 2021 to 27 August 2021, under the guidance of **Mr. Vivek Gaurav Saini**, GEOPIC, ONGC, Dehradun.

(Sparsh Saxena)

Certificate

This is to certify that **Mr. Sparsh Saxena**, a student of **B. Tech. Hons. (Computer Science)-(Data Science & Artificial Intelligence)** of Graphic Era Deemed to be University, Dehradun has done his Summer Training at **GEOPIC, ONGC Dehradun.** The project work entitled **"Facies Classification and Synthetic Curve Generation in Well Logs using AI/ML based Data Science Workflow"** embodies the original work done by **Mr. Sparsh Saxena** during his summer training period from 28 June 2021 to 27 August 2021.

Saini 31-08-2021

विवेक गौरव सैनी/ Vivek Gaurav Saini
वरिष्ठ प्रोग्रामिंग अधिकारी
Senior Programming Officer
सी. एस. सॉफ्टवेयर विभाग / C.S. (S/W) Division
जियोपिक,ओएनजीसी,देहरादून
GEOPIC, ONGC, Dehradun

Signature of Project Guide

(Mr. Vivek Gaurav Saini)

Sr. Programming Officer

P. 31.08.2021

पी.आर.मीणा / P.R. Meena
महाप्रबंधक (प्रोग्रामिंग) / GM (Prog.)
सी.एस. (सॉफ्टवेयर) विभाग / CS (S/W) Division
जियोपिक,ओएनजीसी,देहरादून
GEOPIC, ONGC, Dehradun

Signature of Training Coordinator

(Mr. P.R. Meena)

GM (Programming)

Acknowledgement

The summer training at ONGC is a golden opportunity for learning and self-development. I consider myself very fortunate and honoured to be able to be a part of it and have such experienced and expert professionals to lead me through the completion of this project. It gives me immense pleasure and a sense of satisfaction to have an opportunity to acknowledge and to express gratitude to those who were associated with me during my summer training at GEOPIC, ONGC Dehradun.

I express my sincere thanks and gratitude to ONGC authorities for allowing me to undergo the training in this prestigious organization. I would like to thank **Mr. V.K Sharma, Executive Director and Mr. Vivek Gaurav Saini, Sr. Programming Officer** for providing me the technical guidance and directions to work.

Finally, I would thank my parents for imparting me moral support and motivation during this project.

TABLE OF CONTENTS

1. ABSTRACT

In this project, the aim was to implement the steps of Data Science workflow in a real-world dataset and explore various algorithms at each step. The problem was to automate the process of facies classification and synthetic curve generation.

I explored the task of predicting the facies value and DT value for wells. The dataset provided was obtained from well-logging expeditions. Given a set of twenty las files (semi-structured) containing data of twenty different wells, the project focussed on getting meaningful data out of it for facies classification and synthetic curve generation. I applied certain algorithms at each step. The las files were converted into csv file and then merged. Six imputation techniques- Mean, bfill, ffill, k-nearest neighbour, Iterative Imputer, and two modified techniques were used to deal with null values. For model building, the algorithms used were **Gaussian Naïve Bayes, Logistic Regression (cross validation), Decision Tree Classifier, Cat Boost Classifier, XGBClassifier, LGBMClassifier, Random Forest Classifier, K-nearest neighbour** for Classification to predict Facies and **Linear Regression, CatBoostRegressor, GradientBoosting** for Regression to predict the DT-curve. Results of the machine learning algorithms were tabulated, and their comparison resulted in the conclusion that Random Forest was the best algorithm for facies classification. Feature scaling was required prior to building the models for generating the DT curve for better results.

2. PROBLEM STATEMENT

Background:

The Geodata Processing and Interpretation Centre (GEOPIC) is a premier seismic processing and interpretation work-centre of ONGC. Among various disciplines, the Petro-physicists at GEOPIC work upon specialised Well Log interpretation software in collaboration with other Geoscientists to help create a comprehensible picture of sub-surface (Earth's section below the surface), which ultimately leads to discovery of prospective hydrocarbon bearing location.

The Well Logs are the measurements at periodic locations in the subsurface which are recorded during or after drilling of well. These logs consist of various physical, chemical, and nuclear properties of layers in the earth's interior which helps to understand the stratigraphy.

Lith-facies: These are mappable subdivision of a designated stratigraphic unit, distinguished from adjacent subdivisions based on lithology; facies characterized by lithologic features. A Petro-physicist mark the facies in their interpretation software based on observations about various logs available at a particular depth. This lithology analysis becomes a base for determination the nature of matter at a particular depth.

Synthetic Curves: Synthetic curves are the curves generated artificially using various techniques to account for missing curves. The missing curve problem is manifested in two forms:

1. Unavailability of certain logs at some wells due to various limitations such as borehole problems or cost considerations.

2.Gaps in recorded log sequence at several intervals along the well path, due to various limitations such as borehole problems, instrument limitations.

Problem:

It is required to automate the process of Facies Identification using Machine Learning **Classification** method. This can be achieved by using the wells where facies have been already marked by the interpreters. This can be used to train the model as labelled examples. The trained model can then be used to predict facies on new wells.

Also, missing curve problem is to be addressed by Machine Learning **Regression** method, where a curve or part of a curve can be predicted using trained models.

3. SCOPE AND OBJECTIVES OF PROJECT

The scope of this project spans to analysing the LAS files data of twenty wells and refining the data by correcting its values by using various imputation techniques and preparing it for further future analysis.

The objective of the project is to get the huge data with various parameters then imputing that data into four main independent variables **DT, GR, NPHI, RHOB** and dependent variable **FACIES.**

Thereafter, it has been divided into two prospects those are CLASSIFICATION and REGRESSION where main objective was to perform various Machine Learning Algorithms.

4. SOLUTION DESIGN

PROJECT ROADMAP:

STEP 1: Las to Csv (Semi-Structured Data to Structured Data) :

First step was to convert given 20 Las files to csv files and merge into one csv using 2 different python libraries -: **lasio (to convert las files to csv), glob (to merge all converted csv files to one csv)**

STEP 2: Imputation:

It is the process of replacing missing data with substituted values. Performing imputation with different strategies: **Mean, Bfill, Ffill, KNNImputer, Iterative Imputer and dropping null values.**

STEP 3: Outlier Removal:

In data analysis, anomaly/outlier detection/removal is the identification of rare items, events or observations which raise suspicions by differing significantly from most of the data. In this project outlier removal was performed using **3-standard deviation, 4-standard deviation, and Interquartile range.**

STEP 4: Feature Selection:

In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features for use in model construction.
Feature selection methods used in this project were **Variance Threshold, Absolute Correlation, Correlation, Select K Best, Mutual info classifier.**

STEP 5: Scaling (Normalization or Standardization):

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.

STEP 6: Splitting Data into train and test (Using train test split):

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

STEP 7: Builiding Model / Training Model:

The process of training an ML model involves providing an ML algorithm (that is, the learning algorithm) with training data to learn from. Machine learning algorithms used are **Gaussian Naïve Bayes, Logistic Regression (using cross validation and Grid search), Decision Tree Classifier, Cat Boost Classifier, XGBClassifier, LGBMClassifier, Random Forest Classifier, K-nearest Neighbour** for classification and **Linear Regression, CatBoostRegressor, GradientBoosting** for regression.

STEP 8:Predicting Result Using Trained ML Model:

"Prediction" refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data in this case to predict Facies and DT curve.

5. IMPLEMENTATION TECHNOLOGIES AND PLATFORMS

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed, and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e., Python 2.7.x, was officially discontinued on January 1, 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported. Python interpreters are available for many operating systems.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions, list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML

Tools Used

Jupyter

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include - data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

6. DEPLOYMENT AND TESTING OF THE SOFTWARE

Twenty Model Well LAS files were merged into one csv file and then testing was done by applying different algorithms for performing imputation for reducing data complexity followed by classification and regression.

CLASSIFICATION

[27]:

	DT	GR	NPHI	RHOB	FACIES
0	50.2544	50.2128	0.5340	2.1228	NaN
1	50.3881	49.7509	0.5316	2.1250	NaN
2	49.8852	48.2513	0.5126	2.1316	NaN
3	49.9032	46.8212	0.5137	2.1437	NaN
4	50.0157	45.3463	0.5472	2.1611	NaN
5	50.6831	44.0819	0.5550	2.1740	NaN
6	51.4311	43.6654	0.5612	2.1707	NaN
7	52.1678	43.3915	0.5566	2.1595	NaN
8	52.2883	44.1249	0.5390	2.1534	NaN
9	51.5991	46.1805	0.5245	2.1551	NaN
10	50.6185	48.6156	0.5152	2.1542	NaN
11	50.5171	49.6999	0.5152	2.1535	NaN
12	50.1209	49.4600	0.5180	2.1586	NaN
13	50.0558	48.3665	0.5156	2.1662	NaN
14	49.4216	46.8647	0.5070	2.1705	NaN
15	47.9804	45.7345	0.4913	2.1702	NaN
16	46.3324	45.5512	0.4696	2.1657	NaN
17	45.1378	45.9222	0.4570	2.1579	NaN
18	45.2291	46.4844	0.4654	2.1533	NaN
19	45.6106	49.6481	0.4952	2.1526	NaN

Sonic (DT)	**Porosity calculation, wave velocity calculation, rock physics properties (AI, SI, σ, etc.) calculation, etc.**
Gamma Ray (GR)	**Lithology interpretation, shale volume calculation, calculate clay volume, permeability calculation, porosity calculation, wave velocity calculation, etc.**
Neutron Porosity (NPHI)	**Finding hydrocarbon bearing zone, porosity calculation, etc.**
Density (RHOB)	**Lithology interpretation, finding hydrocarbon bearing zone, porosity calculation, rock physics properties (AI, SI, σ, etc.) calculation, etc.**

Lith-facies : These are mappable subdivision of a designated stratigraphic unit, distinguished from adjacent subdivisions on the basis of lithology; a facies characterized by particular lithologic features. A Petro-physicist mark the facies in their interpretation software based on observations about various logs available at a particular depth. This lithology analysis becomes a base for determination the nature of matter at a particular depth.

REGRESSION

```
[493]: df
```

[493]:

	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]

Gamma Ray (GR)	**Lithology interpretation, shale volume calculation, calculate clay volume, permeability calculation, porosity calculation, wave velocity calculation, etc.**
Density (RHOB)	**Lithology interpretation, finding hydrocarbon bearing zone, porosity calculation, rock physics properties (AI, SI, σ, etc.) calculation, etc.**
Neutron Porosity (NPHI)	**Finding hydrocarbon bearing zone, porosity calculation, etc.**
Sonic (DT)	**Porosity calculation, wave velocity calculation, rock physics properties (AI, SI, σ, etc.) calculation, etc.**

7. OUTPUT SNAPSHOTS

Libraries Used:

These are the required libraries used for the project:

1 IMPORTANT LIBRARIES

[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]:
```
#pip install lasio
```

[5]:
```
#Libraries to convert .las files to .csv and merge

import lasio
from sys import stdout
import glob  ##For merging csv files
```

[6]:
```
#DATA IMPUTATION LIBRARY
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import KNNImputer
from sklearn.linear_model import LinearRegression
```

[7]:
```
#Feature Selection Libraries
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import SelectKBest
```

[8]:
```
#SCALING LIBRARIES
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer,␣
↪RobustScaler, MaxAbsScaler
```

[9]:
```
#pip install catboost
```

[10]:
```
#MODEL TRAINING LIBRARIES
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from catboost import CatBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
```

[11]:
```
#MODEL ACCURACY LIBRARIES
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

[12]:
```
#grid searching key hyperparametres for logistic regression
from sklearn.datasets import make_blobs
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
```

[13]:
```
path='/media/mr-robot/Local Disk/summer_training/Train'
os.chdir(path)
```

LAS TO CSV:

Converted Las file to csv (converted semi-structured files to structured file) and merged in file to train machine learning model.

2 LAS TO CSV

[14]:
```
# Converting all las files in csv by iterating using lasio
for file in os.listdir():
    if file.endswith(".las"):
        file_path = f"{path}/{file}"
        las=lasio.read(file_path)
        size=len(file_path)
        filepath1=file_path[:size-3]
        las.to_csv(filepath1+'csv', units=False)
```

[15]:
```
# Adding Well name to easily identify
for file in os.listdir():
    if file.endswith(".csv"):
        s=pd.read_csv(file)
        size=len(file)
        dict=[]
        filename= file[:size-4]
```

```
        t=s.shape[0]
        for i in range(t):
            dict.append(filename)
        s['WELL']=dict
        s.to_csv(filename+'.csv',index=False)
```

[16]:
```
## To avoid furthur merging data and redundancy
if(os.path.isfile('./merged_data.csv') ):
    os.remove("merged_data.csv")

if(os.path.isfile('./FACIES_imputed.csv')):
    os.remove("FACIES_imputed.csv")

if(os.path.isfile('./FACIES_TRAIN.csv')):
    os.remove("FACIES_TRAIN.csv")
```

[17]:
```
# Merging all Well Log using Glob
filenames = glob.glob(path + "/*.csv")
dfs = []
for filename in filenames:
    dfs.append(pd.read_csv(filename))
big_frame = pd.concat(dfs, ignore_index=True)
big_frame.to_csv('merged_data.csv',index=False)
```

IMPUTATION:

After converting to csv and merging into one csv, we did data exploration and performed imputation to deal with null values. One method was to drop rows/datapoints having null values other methods were to use different algorithms to fill null values. So, we used **bfill, ffill (Backward fill, Forward fill), K-nearest neighbour, Iterative Imputer (Experimental in Sklearn), and did some modifications in KNN and Iterative Imputer.**

3 IMPUTATION

[18]:
```
df = pd.read_csv('merged_data.csv')
df
```

[18]:

	DEPTH	ACOUSTICIMPEDANCE1	AI	AVG_PIGN	CALI	\
0	1275.0552	12875.0811	12875081.0	NaN	9.7141	
1	1275.2076	12854.2256	12854226.0	NaN	9.7848	
2	1275.3600	13024.1377	13024138.0	NaN	9.8300	
3	1275.5124	13093.3428	13093343.0	NaN	9.8587	
4	1275.6648	13169.9307	13169931.0	NaN	9.8756	
…	…	…	…	…	…	
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	DT	FACIES	FLD1	GR	…	CALI_1	NPHI_1	\
0	9.7141	50.2544	NaN	NaN	50.2128	…	NaN	NaN	
1	9.7848	50.3881	NaN	NaN	49.7509	…	NaN	NaN	
2	9.8300	49.8852	NaN	NaN	48.2513	…	NaN	NaN	
3	9.8587	49.9032	NaN	NaN	46.8212	…	NaN	NaN	

```
4              9.8756   50.0157     NaN   NaN  45.3463  …     NaN     NaN
…                …        …       …     …    …    …       …     …
58494            NaN  123.7404     NaN   NaN      NaN  …     NaN  0.4993
58495            NaN  123.8728     NaN   NaN      NaN  …     NaN  0.5313
58496            NaN  123.3722     NaN   NaN      NaN  …     NaN  0.5448
58497            NaN  122.6038     NaN   NaN      NaN  …     NaN  0.5364
58498            NaN  122.3045     NaN   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 67 columns]
```

[19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58499 entries, 0 to 58498
Data columns (total 67 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   DT                  58499 non-null  float64
 7   FACIES              52641 non-null  float64
 8   FLD1                3963 non-null   float64
 9   GR                  58379 non-null  float64
 10  LLD                 44942 non-null  float64
 11  LLS                 27394 non-null  float64
 12  DEPTH_1             50885 non-null  float64
 13  NPHI                58172 non-null  float64
 14  ONE-WAYTIME1        15713 non-null  float64
 15  PIGN_MODELLING      51101 non-null  float64
 16  PIMP                55259 non-null  float64
 17  RHOB                58499 non-null  float64
```

```
18  RT_MODELLING        53629 non-null  float64
19  SP                  55992 non-null  float64
20  SUWI_MODELLING      51099 non-null  float64
21  TDVSS               58437 non-null  float64
22  ZLT                 44562 non-null  float64
23  WELL                58499 non-null  object
24  DFL                 23458 non-null  float64
25  HDRS                26951 non-null  float64
26  HMRS                26951 non-null  float64
27  PERF_INT            1569 non-null   float64
28  PERMEABILITY        28149 non-null  float64
29  PIGN                46949 non-null  float64
30  RT_POWER            51379 non-null  float64
31  SUWI                46947 non-null  float64
32  VCL                 46947 non-null  float64
33  WATER_VOL           43735 non-null  float64
34  LL3                 12373 non-null  float64
35  BS                  6706 non-null   float64
36  CALI1               2389 non-null   float64
37  DEVI                10283 non-null  float64
38  DT1                 6130 non-null   float64
39  PHIT                16532 non-null  float64
40  PIGE                5245 non-null   float64
41  LLD_1               9518 non-null   float64
42  SXWI                27938 non-null  float64
43  PEF                 19419 non-null  float64
44  AZI1                2487 non-null   float64
45  TEMP                14514 non-null  float64
46  DRES                2765 non-null   float64
47  DT2                 2765 non-null   float64
48  DT4P                5854 non-null   float64
49  GR_EDTC             2765 non-null   float64
50  M2R2                8568 non-null   float64
51  LLS_1               238 non-null    float64
52  MSFL                2765 non-null   float64
53  PR                  2757 non-null   float64
54  TENS                2765 non-null   float64
55  VPVS                2757 non-null   float64
56  BIT                 5553 non-null   float64
57  CALI_1              2999 non-null   float64
58  NPHI_1              10811 non-null  float64
59  ZCOR                2998 non-null   float64
60  RHOB_1              10899 non-null  float64
61  RXO                 1552 non-null   float64
62  SPDH                3069 non-null   float64
63  DTDS                2546 non-null   float64
64  M2R1                2546 non-null   float64
65  TH                  2509 non-null   float64
```

```
 66  U                   2509 non-null   float64
dtypes: float64(66), object(1)
memory usage: 29.9+ MB
```

[20]:
```
df.shape[1]
```

[20]: 67

[21]:
```
obj = df.isnull().sum()
for key,value in obj.iteritems():
    print(key,",",value)
```

```
DEPTH , 0
ACOUSTICIMPEDANCE1 , 0
AI , 3240
AVG_PIGN , 58176
CALI , 3518
CALI[DERIVED]1 , 14409
DT , 0
FACIES , 5858
FLD1 , 54536
GR , 120
LLD , 13557
LLS , 31105
DEPTH_1 , 7614
NPHI , 327
ONE-WAYTIME1 , 42786
PIGN_MODELLING , 7398
PIMP , 3240
RHOB , 0
RT_MODELLING , 4870
SP , 2507
SUWI_MODELLING , 7400
TDVSS , 62
ZLT , 13937
WELL , 0
DFL , 35041
HDRS , 31548
HMRS , 31548
PERF_INT , 56930
PERMEABILITY , 30350
PIGN , 11550
RT_POWER , 7120
SUWI , 11552
VCL , 11552
WATER_VOL , 14764
LL3 , 46126
BS , 51793
```

```
CALI1 , 56110
DEVI , 48216
DT1 , 52369
PHIT , 41967
PIGE , 53254
LLD_1 , 48981
SXWI , 30561
PEF , 39080
AZI1 , 56012
TEMP , 43985
DRES , 55734
DT2 , 55734
DT4P , 52645
GR_EDTC , 55734
M2R2 , 49931
LLS_1 , 58261
MSFL , 55734
PR , 55742
TENS , 55734
VPVS , 55742
BIT , 52946
CALI_1 , 55500
NPHI_1 , 47688
ZCOR , 55501
RHOB_1 , 47600
RXO , 56947
SPDH , 55430
DTDS , 55953
M2R1 , 55953
TH , 55990
U , 55990
```

[22]:
```
#Selecting required feature
df=df[["DT","GR","NPHI","RHOB","FACIES"]]
```

[23]:
```
df
```

[23]:

	DT	GR	NPHI	RHOB	FACIES
0	50.2544	50.2128	0.5340	2.1228	NaN
1	50.3881	49.7509	0.5316	2.1250	NaN
2	49.8852	48.2513	0.5126	2.1316	NaN
3	49.9032	46.8212	0.5137	2.1437	NaN
4	50.0157	45.3463	0.5472	2.1611	NaN
…	…	…	…	…	…
58494	123.7404	NaN	0.4993	2.4639	NaN
58495	123.8728	NaN	0.5313	2.4660	NaN
58496	123.3722	NaN	0.5448	2.4714	NaN

```
58497  122.6038      NaN  0.5364  2.4750     NaN
58498  122.3045      NaN  0.5331  2.4709     NaN

[58499 rows x 5 columns]
```

[24]:
```
df.isnull().sum()
```

[24]:
```
DT           0
GR         120
NPHI       327
RHOB         0
FACIES    5858
dtype: int64
```

[25]:
```
#Exporting required features to csv
df.to_csv("FACIES_TRAIN.csv",index=False)
```

[26]:
```
df=pd.read_csv("FACIES_TRAIN.csv")
```

[27]:
```
df.head(20)
```

[27]:

	DT	GR	NPHI	RHOB	FACIES
0	50.2544	50.2128	0.5340	2.1228	NaN
1	50.3881	49.7509	0.5316	2.1250	NaN
2	49.8852	48.2513	0.5126	2.1316	NaN
3	49.9032	46.8212	0.5137	2.1437	NaN
4	50.0157	45.3463	0.5472	2.1611	NaN
5	50.6831	44.0819	0.5550	2.1740	NaN
6	51.4311	43.6654	0.5612	2.1707	NaN
7	52.1678	43.3915	0.5566	2.1595	NaN
8	52.2883	44.1249	0.5390	2.1534	NaN
9	51.5991	46.1805	0.5245	2.1551	NaN
10	50.6185	48.6156	0.5152	2.1542	NaN
11	50.5171	49.6999	0.5152	2.1535	NaN
12	50.1209	49.4600	0.5180	2.1586	NaN
13	50.0558	48.3665	0.5156	2.1662	NaN
14	49.4216	46.8647	0.5070	2.1705	NaN
15	47.9804	45.7345	0.4913	2.1702	NaN
16	46.3324	45.5512	0.4696	2.1657	NaN
17	45.1378	45.9222	0.4570	2.1579	NaN
18	45.2291	46.4844	0.4654	2.1533	NaN
19	45.6106	49.6481	0.4952	2.1526	NaN

[28]:
```
df.shape
```

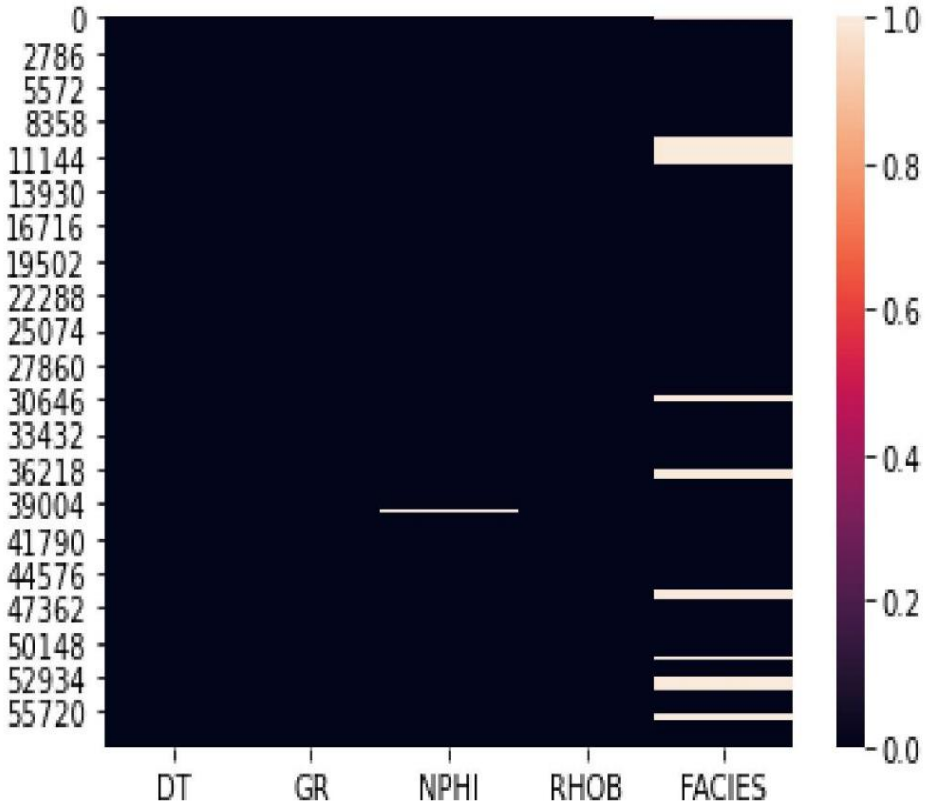
[28]:
```
(58499, 5)
```

[29]:
```
df.info()
```

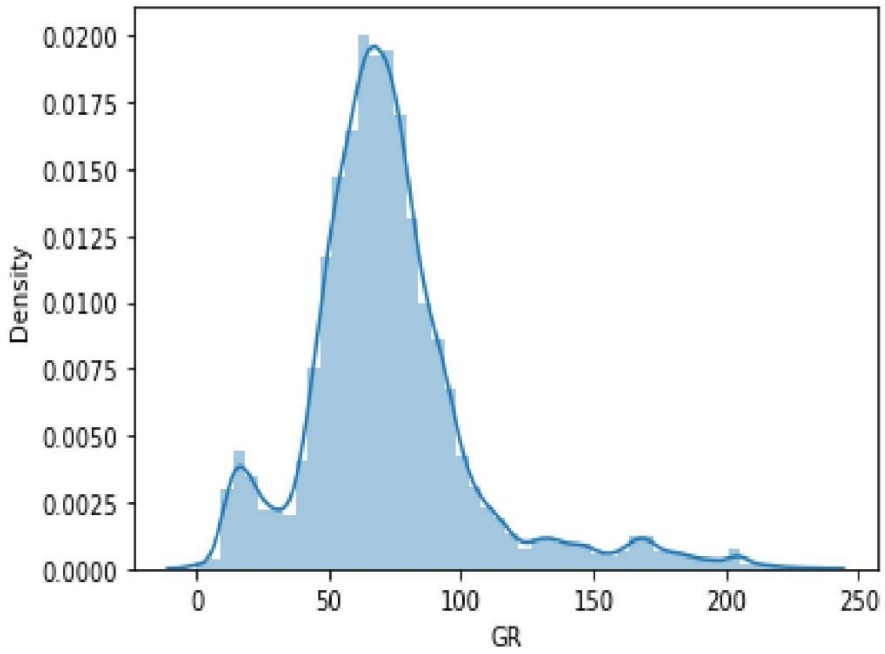
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58499 entries, 0 to 58498
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   DT      58499 non-null  float64
 1   GR      58379 non-null  float64
 2   NPHI    58172 non-null  float64
 3   RHOB    58499 non-null  float64
 4   FACIES  52641 non-null  float64
dtypes: float64(5)
memory usage: 2.2 MB
```

[30]:
```
sns.heatmap(df.isnull())
```

[30]: <AxesSubplot:>



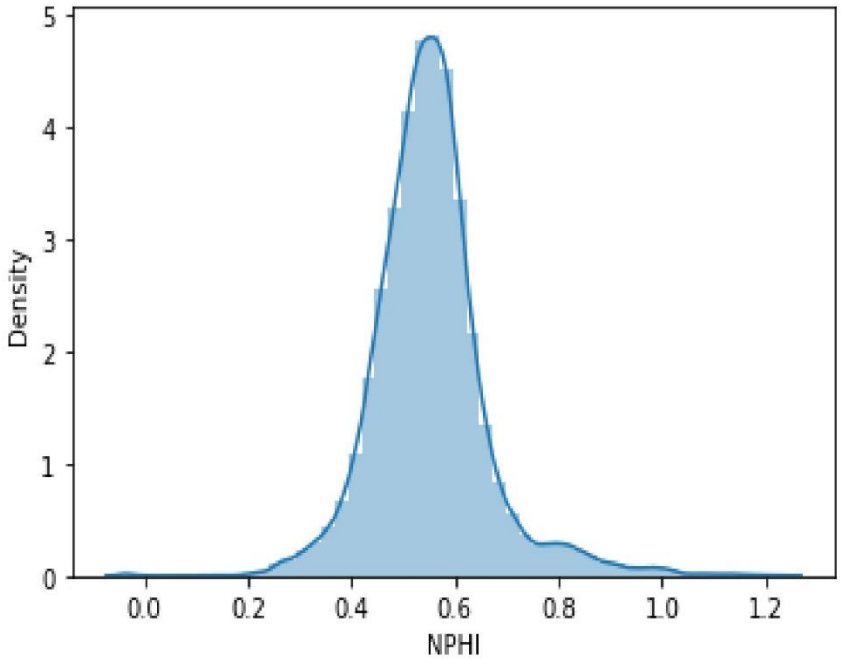
[31]:
```
null_gr = sns.distplot(df.GR.dropna())
```



[32]:
```
df.GR.describe()
```

[32]:
```
count    58379.000000
mean        72.610942
std         32.140407
min          0.000000
25%         55.340300
50%         68.939700
75%         83.758300
max        233.707400
Name: GR, dtype: float64
```

[33]:
```
null_nphi=sns.distplot(df.NPHI.dropna())
```

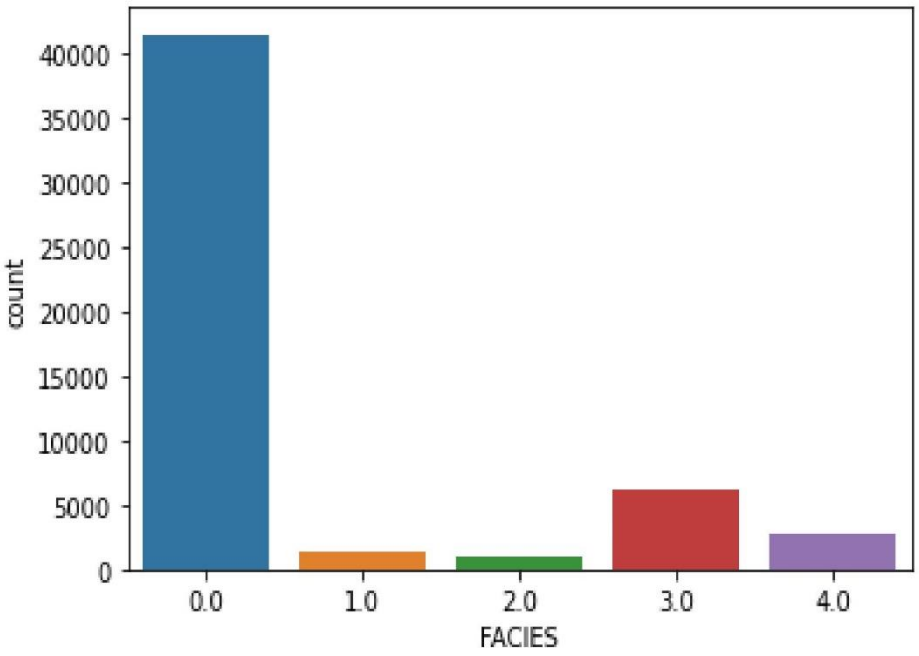


[34]:
```
df.NPHI.describe()
```

[34]:
```
count    58172.000000
mean         0.551710
std          0.109983
min         -0.038000
25%          0.489275
50%          0.546600
75%          0.600500
max          1.231200
Name: NPHI, dtype: float64
```

[35]:
```
sns.countplot(x="FACIES",data=df)
```

[35]:
```
<AxesSubplot:xlabel='FACIES', ylabel='count'>
```



[36]:
```
df.FACIES.value_counts(dropna=False)
```

[36]:
```
0.0    41514
3.0     6138
NaN     5858
4.0     2798
1.0     1281
2.0      910
Name: FACIES, dtype: int64
```

[37]:
```
def imputing(imputation_strategy,imputing_data):
    df=imputing_data
    if imputation_strategy == "Mean":
        df.GR.fillna(df.GR.mean(),inplace=True)
        print( df.GR.isnull().sum())
        print("Graph (GR) after filling null values with mean")
        sns.displot(df.GR.dropna())
        df.NPHI.fillna(df.NPHI.mean(),inplace=True)
        print("Graph (NPHI) after filling null values with mean")
        print(df.NPHI.isnull().sum())
        sns.displot(df.NPHI.dropna())
        #dropping FACIES rows with null
        df.dropna(axis=0,inplace=True)
        print(df.isnull().sum())
        df['FACIES'] = df.FACIES.astype(np.int64)
```

```
        df.info()
        df.FACIES.describe()
        return df

    elif imputation_strategy == "bffill":
        df = df.ffill(axis = 0)
        df = df.bfill(axis = 0)
        df['FACIES'] = df.FACIES.astype(np.int64)
        print(df.isnull().sum())
        return df

    elif imputation_strategy == "KNNImputer":
        knn= KNNImputer(n_neighbors=3)
        X=df.drop('FACIES',1)
        t=knn.fit_transform(X)
        X=pd.DataFrame(t)
        Y=df['FACIES']
        Y=Y.ffill(axis=0)
        Y=Y.bfill(axis=0)
        X['FACIES']=Y
        df=X
        df['FACIES'] = df.FACIES.astype(np.int64)
        d=['DT','GR','NPHI','RHOB']
        for i in range(4):
            df.columns.values[i]=d[i]
        return df

    elif imputation_strategy == "IterativeImputer":
        lr=LinearRegression()    #can use other regressions too. / default is␣
↪beysian
        imp=IterativeImputer(max_iter=3)
        X=df.drop('FACIES',1)
        t=imp.fit_transform(X)
        X=pd.DataFrame(t)
        Y=df['FACIES']
        Y=Y.ffill(axis=0)
        Y=Y.bfill(axis=0)
        X['FACIES']=Y
        df=X
        df['FACIES'] = df.FACIES.astype(np.int64)
        d=['DT','GR','NPHI','RHOB']
        for i in range(4):
            df.columns.values[i]=d[i]
        return df

    elif imputation_strategy == "KNNimputer_floor" :
        X=df
```

```
        knn= KNNImputer(n_neighbors=3)
        t=knn.fit_transform(df)
        df=pd.DataFrame(t)
        d=['DT','GR','NPHI','RHOB','FACIES']
        df['FACIES1'] = X.FACIES
        for i in range(5):
            df.columns.values[i]=d[i]
        df=df.drop('FACIES1',1)
        df['FACIES'] = df.FACIES.astype(np.int64)
        return df

    elif imputation_strategy == "IterativeImputer_floor" :
        X=df
        lr=LinearRegression()
        imp= IterativeImputer(max_iter=3)
        t=imp.fit_transform(df)
        df=pd.DataFrame(t)
        d=['DT','GR','NPHI','RHOB','FACIES']
        df['FACIES1'] = X.FACIES
        for i in range(5):
            df.columns.values[i]=d[i]
        df=df.drop('FACIES1',1)
        df['FACIES'] = df.FACIES.astype(np.int64)
        return df

    elif imputation_strategy == "KNNBinning" :
        X=df
        knn= KNNImputer(n_neighbors=3)
        t=knn.fit_transform(df)
        df=pd.DataFrame(t)
        d=['DT','GR','NPHI','RHOB','FACIES']
        df['FACIES1'] = X.FACIES
        for i in range(5):
            df.columns.values[i]=d[i]
        df=df.drop('FACIES1',1)
        #df['FACIES'] = pd.cut(x=df['FACIES'],bins=[0,0.5,1.5,2.5,3.5,4.0],␣
↪labels=['0','1','2','3','4'])
        return df

    elif imputation_strategy == "dropna":
        df=df.dropna(axis=0)
        return df
```

[38]:
```
imputation_strategy = ["Mean" , "bffill" , "KNNImputer" , "IterativeImputer" ,␣
↪"KNNimputer_floor" , "IterativeImputer_floor" , "KNNBinning","dropna"]
#select option from 0-7 (6 is experimental)
optionimputation=5
```

```
df=imputing(imputation_strategy[optionimputation],df)
```

[39]:
```
#if option==6:
#    df['FACIES'] = pd.cut(x=df['FACIES'],bins=[0.0,0.5,1.5,2.5,3.5,4.0],␣
 ↪labels=['0','1','2','3','4'])
```

[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58499 entries, 0 to 58498
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   DT      58499 non-null  float64
 1   GR      58499 non-null  float64
 2   NPHI    58499 non-null  float64
 3   RHOB    58499 non-null  float64
 4   FACIES  58499 non-null  int64
dtypes: float64(4), int64(1)
memory usage: 2.2 MB
```

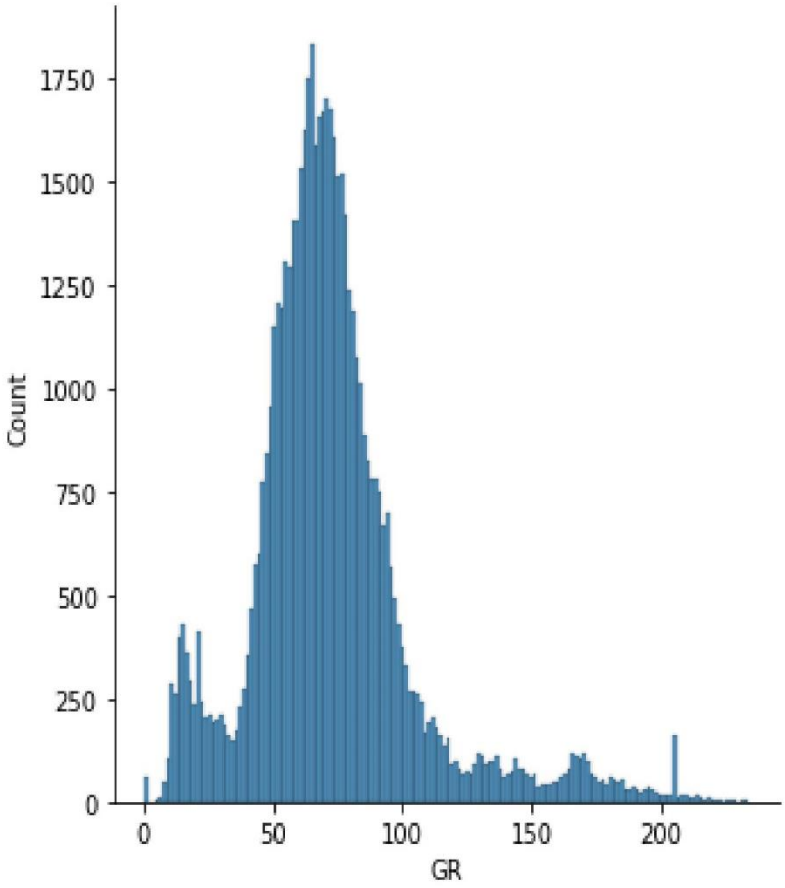
[41]:
```
df.isnull().sum()
```

[41]:
```
DT        0
GR        0
NPHI      0
RHOB      0
FACIES    0
dtype: int64
```

[42]:
```
df.to_csv("FACIES_imputed.csv",index=False)
df=pd.read_csv("FACIES_imputed.csv")
```

[43]:
```
sns.displot(df.GR.dropna())
```

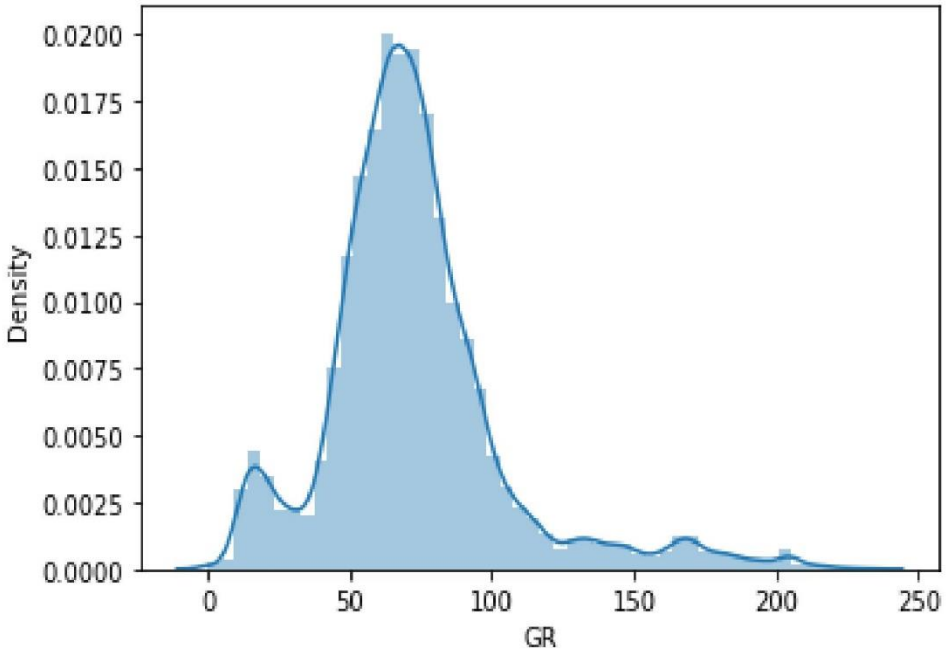
[43]:
```
<seaborn.axisgrid.FacetGrid at 0x7f43f16a3310>
```



[44]:
```
print("WHEN GR WAS NULL")
null_gr.figure
```

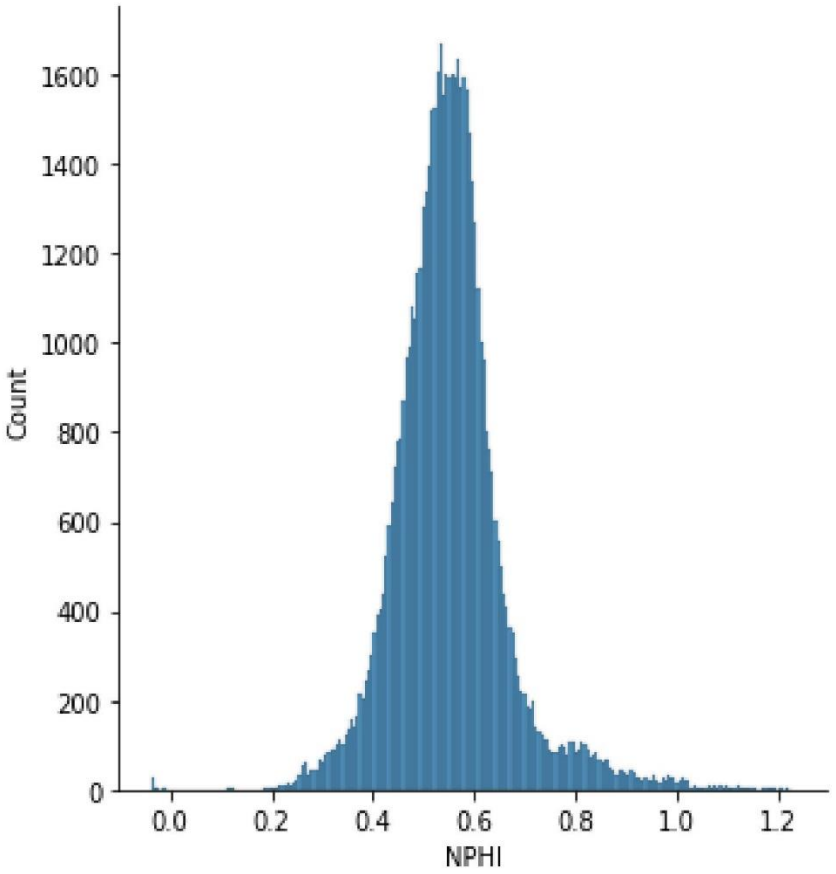
```
WHEN GR WAS NULL
```

[44]:



[45]:
```
sns.displot(df.NPHI.dropna())
```

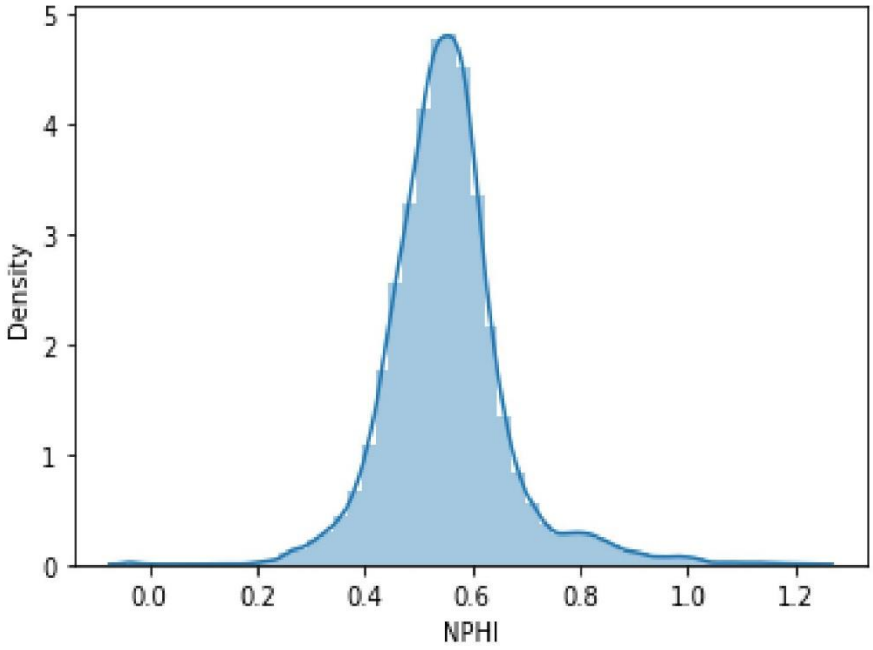
[45]: <seaborn.axisgrid.FacetGrid at 0x7f43f1b47760>



[46]:
```
print("WHEN NPHI WAS NULL")
null_nphi.figure
```

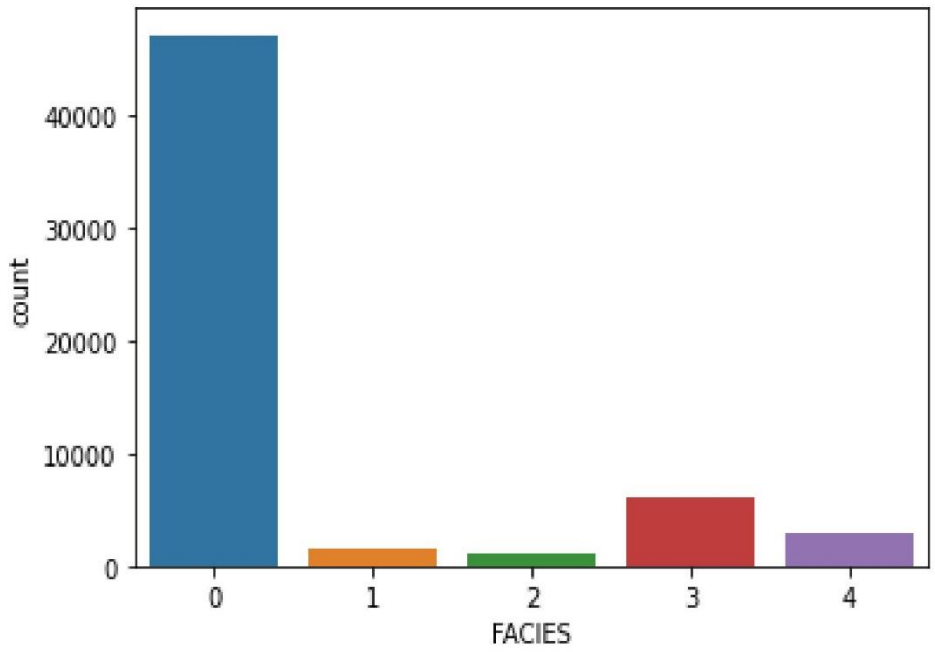
```
WHEN NPHI WAS NULL
```

[46]:



[47]:
```
sns.countplot(x="FACIES",data=df)
```

[47]:
```
<AxesSubplot:xlabel='FACIES', ylabel='count'>
```



Outlier Removal / Data Conditioning:

After exploring and imputing values to replace null values, we performed outlier removal. An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal) objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. Outliers can be detected using visualization, implementing mathematical formulas on the dataset, or using the statistical approach.

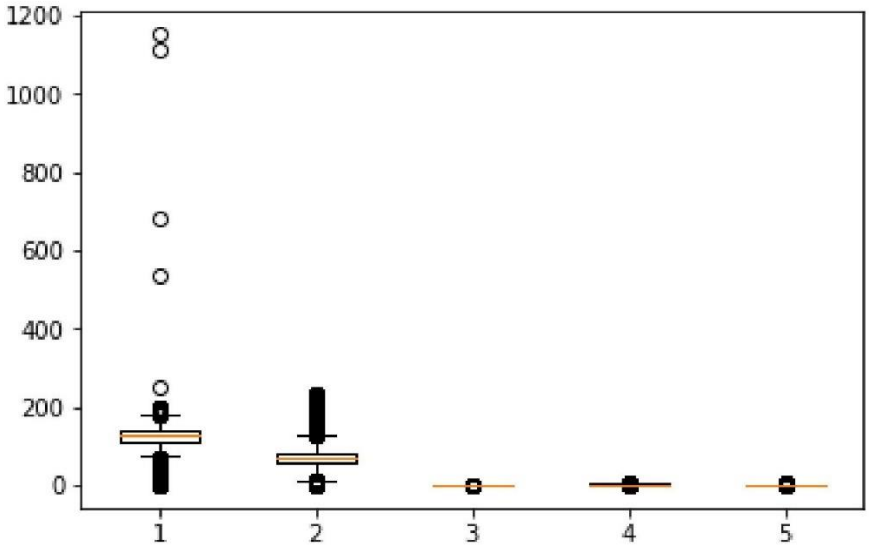
4 DATA CONDITIONING / OUTLIER REMOVAL

```
[48]: df.head
```

```
[48]: <bound method NDFrame.head of             DT         GR    NPHI    RHOB  FACIES
      0       50.2544  50.212800  0.5340  2.1228       1
      1       50.3881  49.750900  0.5316  2.1250       1
      2       49.8852  48.251300  0.5126  2.1316       1
      3       49.9032  46.821200  0.5137  2.1437       1
      4       50.0157  45.346300  0.5472  2.1611       1
      ...         ...        ...     ...     ...     ...
      58494  123.7404  80.913653  0.4993  2.4639       0
      58495  123.8728  82.952576  0.5313  2.4660       0
      58496  123.3722  84.044079  0.5448  2.4714       0
      58497  122.6038  83.725389  0.5364  2.4750       0
      58498  122.3045  83.329152  0.5331  2.4709       0

      [58499 rows x 5 columns]>
```

4.1 WHOLE DATA OUTLIER VISUALIZATION

```
[49]: plt.boxplot(df)
```

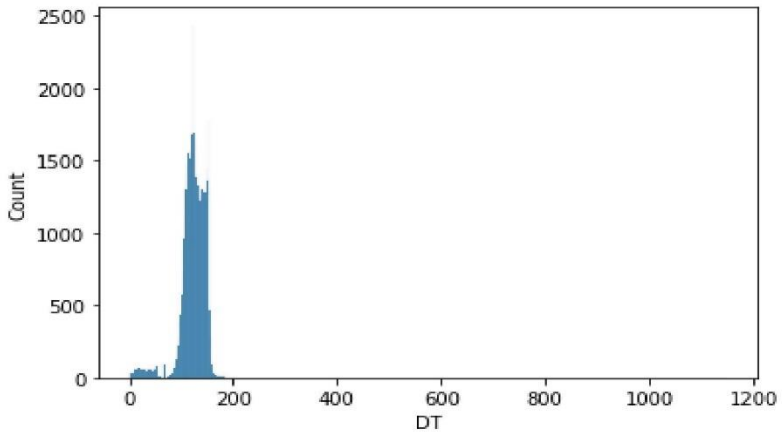


4.2 DT VISUALIZATION

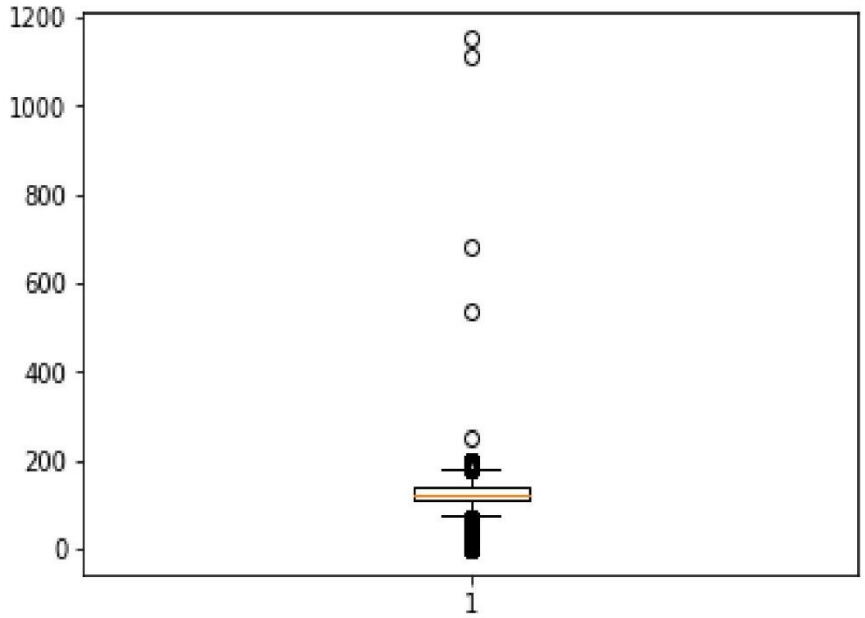
```
[50]: sns.histplot(df.DT)
```

```
[50]: <AxesSubplot:xlabel='DT', ylabel='Count'>
```

Here we can see major outlier in X = 1 i.e., in DT column



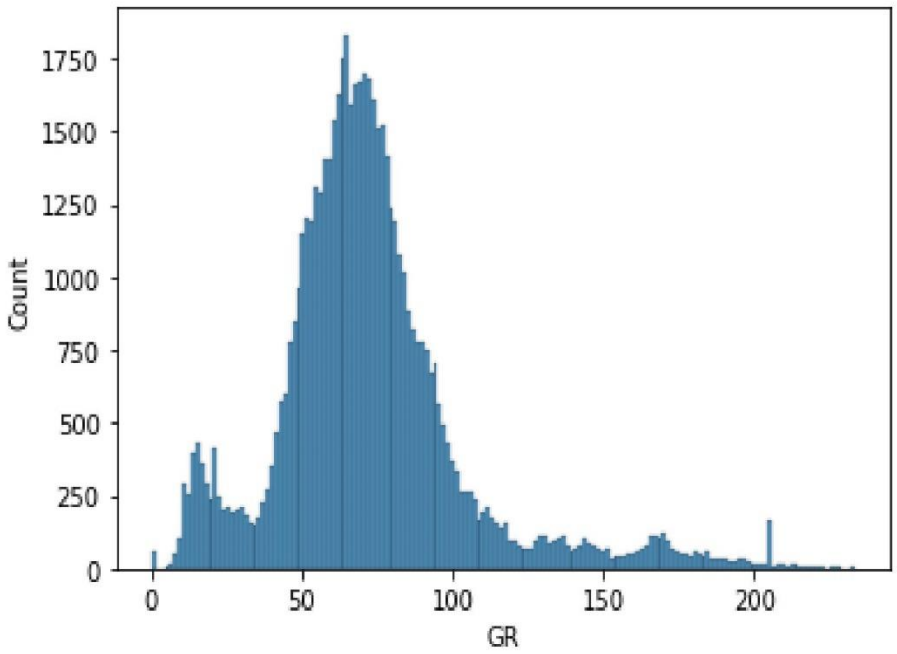

```
[51]: plt.boxplot(df["DT"])
```



4.3 GR VISUALIZATION

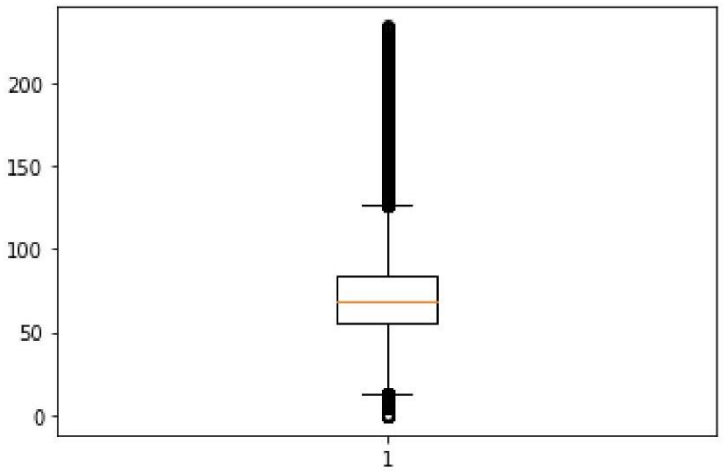
[52]:
```
sns.histplot(df.GR)
```

[52]:
```
<AxesSubplot:xlabel='GR', ylabel='Count'>
```



[53]:
```
plt.boxplot(df.GR)
```

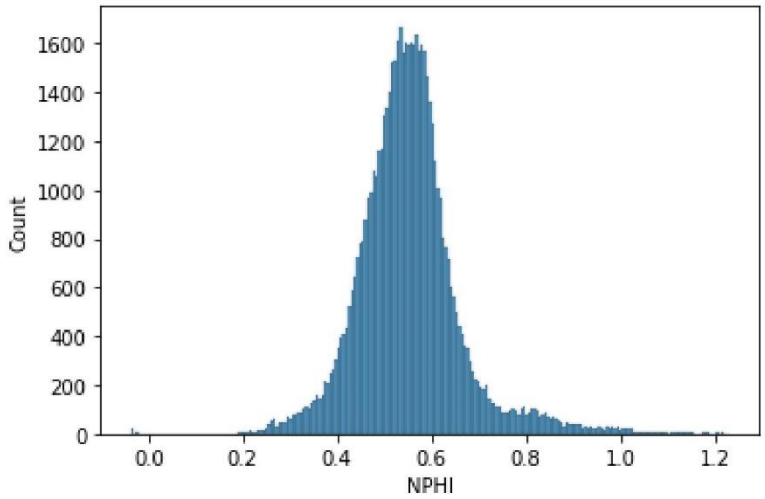
[53]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e3687550>,
  <matplotlib.lines.Line2D at 0x7f43e36878e0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e3687c70>,
  <matplotlib.lines.Line2D at 0x7f43e3693040>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e36871c0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e36933d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e3693760>],
 'means': []}
```



4.4 NPHI VISUALIZATION

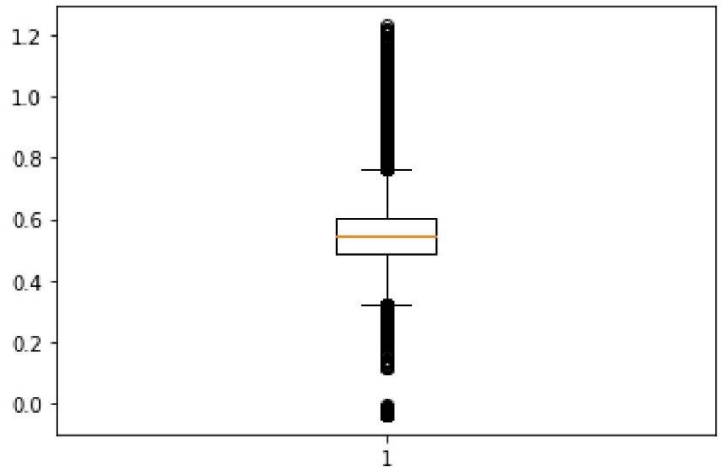
[54]:
```
sns.histplot(df.NPHI)
```

[54]:
```
<AxesSubplot:xlabel='NPHI', ylabel='Count'>
```



[55]:
```
plt.boxplot(df.NPHI)
```

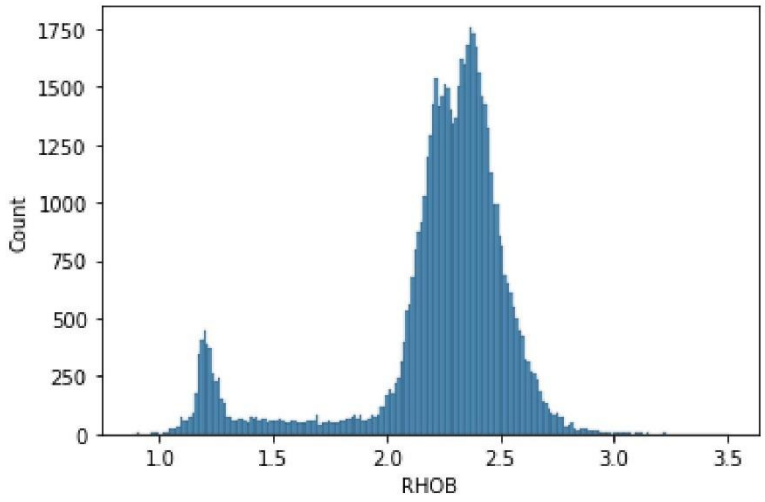
[55]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e336d310>,
  <matplotlib.lines.Line2D at 0x7f43e336d6a0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e336da30>,
  <matplotlib.lines.Line2D at 0x7f43e336ddc0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e335ef40>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e3376190>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e3376520>],
 'means': []}
```



4.5 RHOB VISUALIZATION

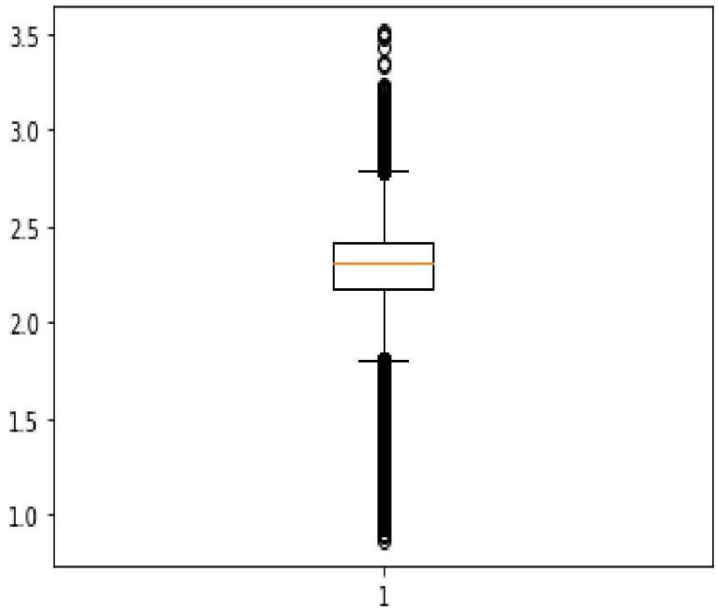
[56]:
```
sns.histplot(df.RHOB)
```

[56]:
```
<AxesSubplot:xlabel='RHOB', ylabel='Count'>
```



[57]:
```
plt.boxplot(df.RHOB)
```

[57]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e30a83a0>,
  <matplotlib.lines.Line2D at 0x7f43e30a8730>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e30a8ac0>,
  <matplotlib.lines.Line2D at 0x7f43e30a8e50>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e309bfd0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e30b4220>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e30b45b0>],
 'means': []}
```



[58]:
```
def outliers(dataConditioningStrategy,dataframe, dataconditioningcolumns):
    df=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) & (df.
↪FACIES >= 0) & (df.FACIES <= 4)]
            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) & (df.
↪FACIES >= 0) & (df.FACIES <= 4)]
            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) & (df.
↪FACIES >= 0) & (df.FACIES <= 4)]
            print(df)


    return df
```

[59]:
```
DATAConditioningStrategy =␣
↪["3_Standard_Deviation","4_Standard_Deviation","InterquartileRange"]
DATAConditioningColumns = ["DT","GR","NPHI","RHOB"]
optionoutlier = 2
df = outliers(DATAConditioningStrategy[optionoutlier] , df,␣
↪DATAConditioningColumns)
```

```
column DT
```

```
Percentiles: 25th=112.649, 75th=139.678, IQR=27.029
InterQuartile Range Outliers-:
             DT       GR    NPHI    RHOB  FACIES
0       50.2544  50.2128  0.5340  2.1228       1
1       50.3881  49.7509  0.5316  2.1250       1
2       49.8852  48.2513  0.5126  2.1316       1
3       49.9032  46.8212  0.5137  2.1437       1
4       50.0157  45.3463  0.5472  2.1611       1
…          …        …       …       …       …
44912   71.0756  39.1722  0.3397  3.1791       0
44913   71.3734  39.3511  0.3455  2.9875       0
44929   71.2182  55.9609  0.4199  2.7743       0
44930   70.1539  52.4927  0.3936  2.9376       0
44931   67.9970  48.9224  0.3727  3.0912       0

[2592 rows x 5 columns]
(2592, 5)
              DT          GR    NPHI     RHOB  FACIES
218      75.8412   47.663200  0.4526  2.4314       0
219      76.1991   47.016400  0.4514  2.4413       0
2026     76.4115   48.396700  0.5571  1.0846       0
2027     78.0536   47.637300  0.5496  1.1340       0
2028     75.2216   48.504000  0.5402  1.1749       0
…           …         …       …       …       …
58494   123.7404   80.913653  0.4993  2.4639       0
58495   123.8728   82.952576  0.5313  2.4660       0
58496   123.3722   84.044079  0.5448  2.4714       0
58497   122.6038   83.725389  0.5364  2.4750       0
58498   122.3045   83.329152  0.5331  2.4709       0

[55907 rows x 5 columns]
column GR
Percentiles: 25th=55.447, 75th=84.350, IQR=28.902
InterQuartile Range Outliers-:
              DT         GR    NPHI     RHOB  FACIES
4029    151.3950    11.6218  0.8730  1.1941       3
4030    151.2614    11.7061  0.8996  1.2056       3
4039    152.8249    11.7563  0.7718  1.1963       3
4040    152.8680    11.5903  0.7690  1.1947       3
4041    152.9320    12.0709  0.7689  1.1923       3
…          …         …       …       …       …
57812   116.8102   136.5899  0.5287  2.4344       0
58179   110.8288   128.6649  0.5213  2.3846       0
58180   110.9551   130.6794  0.5160  2.3705       0
58181   114.0812   131.8473  0.4959  2.3630       0
58182   115.8771   127.8300  0.4907  2.3684       0

[4132 rows x 5 columns]
```

```
(4132, 5)
             DT          GR    NPHI    RHOB  FACIES
218     75.8412   47.663200  0.4526  2.4314       0
219     76.1991   47.016400  0.4514  2.4413       0
2026    76.4115   48.396700  0.5571  1.0846       0
2027    78.0536   47.637300  0.5496  1.1340       0
2028    75.2216   48.504000  0.5402  1.1749       0
…          …         …      …       …      …
58494  123.7404  80.913653  0.4993  2.4639       0
58495  123.8728  82.952576  0.5313  2.4660       0
58496  123.3722  84.044079  0.5448  2.4714       0
58497  122.6038  83.725389  0.5364  2.4750       0
58498  122.3045  83.329152  0.5331  2.4709       0

[51775 rows x 5 columns]
column NPHI
Percentiles: 25th=0.491, 75th=0.595, IQR=0.104
InterQuartile Range Outliers-:
             DT        GR    NPHI    RHOB  FACIES
2361   151.4359   44.2168  0.7608  1.3596       3
2362   149.7643   36.0403  0.7885  1.2673       3
2363   149.5450   28.3286  0.7905  1.2324       3
2364   150.3661   22.7745  0.7713  1.2522       3
3039   143.1059   35.7501  0.7585  1.2089       3
…          …         …      …       …      …
54941  114.6628  114.2647  0.3314  2.2033       1
54953  109.6688  104.5008  0.3327  2.2693       1
57287  106.0689   97.8201  0.3325  2.2712       4
57981  150.8674   23.8442  0.7894  1.1197       3
58290  152.3449   17.4243  0.7641  1.1663       3

[2885 rows x 5 columns]
(2885, 5)
             DT          GR    NPHI    RHOB  FACIES
218     75.8412   47.663200  0.4526  2.4314       0
219     76.1991   47.016400  0.4514  2.4413       0
2026    76.4115   48.396700  0.5571  1.0846       0
2027    78.0536   47.637300  0.5496  1.1340       0
2028    75.2216   48.504000  0.5402  1.1749       0
…          …         …      …       …      …
58494  123.7404  80.913653  0.4993  2.4639       0
58495  123.8728  82.952576  0.5313  2.4660       0
58496  123.3722  84.044079  0.5448  2.4714       0
58497  122.6038  83.725389  0.5364  2.4750       0
58498  122.3045  83.329152  0.5331  2.4709       0

[48890 rows x 5 columns]
column RHOB
```

Here we can see the outliers removed from the dataset these can vary upon the method applied to remove outlier in this project we have applied **3-Standard Scaler, 4-Standard Scaler & Interquartile Range**

```
Percentiles: 25th=2.205, 75th=2.429, IQR=0.224
InterQuartile Range Outliers-:
             DT         GR    NPHI    RHOB  FACIES
2026    76.4115  48.396700  0.5571  1.0846       0
2027    78.0536  47.637300  0.5496  1.1340       0
2028    75.2216  48.504000  0.5402  1.1749       0
2359   153.0665  56.560300  0.6720  1.6982       3
2360   153.1740  51.458100  0.7148  1.5046       3
…           …         …      …       …     …
58400   94.9099  59.051400  0.4770  2.9270       0
58401   96.4064  57.049500  0.4644  2.7995       0
58477  100.5518  93.119065  0.4365  2.7816       0
58478   92.1297  99.825730  0.4509  2.8974       0
58479   92.0023  94.744734  0.4585  2.7846       0

[3498 rows x 5 columns]
(3498, 5)
             DT         GR    NPHI    RHOB  FACIES
218     75.8412  47.663200  0.4526  2.4314       0
219     76.1991  47.016400  0.4514  2.4413       0
2250   137.8066  61.327800  0.5643  2.1857       0
2251   139.5873  61.995400  0.5611  2.1762       0
2252   140.0185  63.518800  0.5630  2.1946       0
…           …         …      …       …     …
58494  123.7404  80.913653  0.4993  2.4639       0
58495  123.8728  82.952576  0.5313  2.4660       0
58496  123.3722  84.044079  0.5448  2.4714       0
58497  122.6038  83.725389  0.5364  2.4750       0
58498  122.3045  83.329152  0.5331  2.4709       0

[45392 rows x 5 columns]
```

[60]:
```
df.shape
```

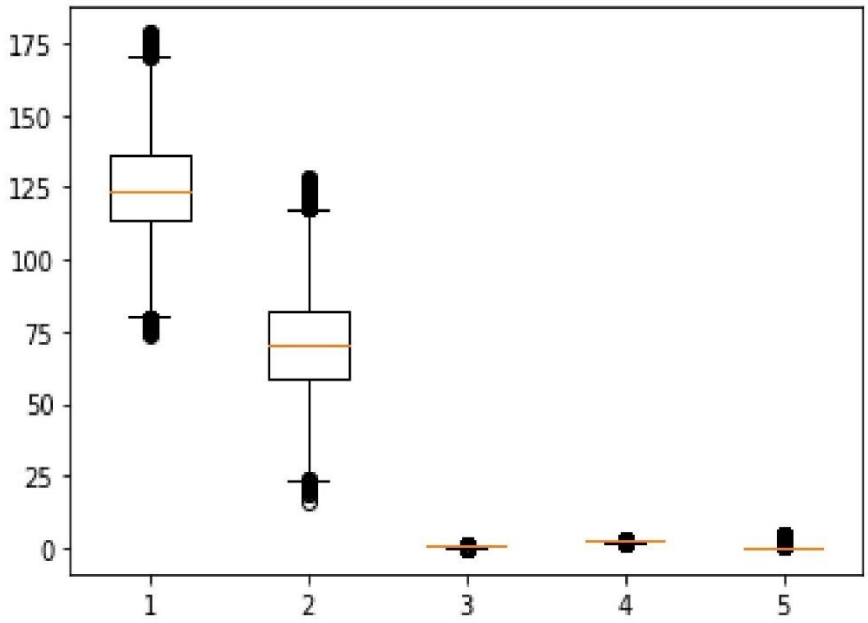
[60]:
```
(45392, 5)
```

4.6 WHOLE DATA AFTER REMOVING OUTLIERS

[61]:
```
plt.boxplot(df)
```

[61]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e308fc70>,
  <matplotlib.lines.Line2D at 0x7f43e2c5e040>,
  <matplotlib.lines.Line2D at 0x7f43e2c69610>,
  <matplotlib.lines.Line2D at 0x7f43e2c699a0>,
  <matplotlib.lines.Line2D at 0x7f43e2c75f40>,
  <matplotlib.lines.Line2D at 0x7f43e2c80310>,
  <matplotlib.lines.Line2D at 0x7f43e2c8c8b0>,
  <matplotlib.lines.Line2D at 0x7f43e2c8cc40>,
```

```
 <matplotlib.lines.Line2D at 0x7f43e3022220>,
 <matplotlib.lines.Line2D at 0x7f43e30225b0>],
'caps': [<matplotlib.lines.Line2D at 0x7f43e2c5e400>,
 <matplotlib.lines.Line2D at 0x7f43e2c5e790>,
 <matplotlib.lines.Line2D at 0x7f43e2c69d30>,
 <matplotlib.lines.Line2D at 0x7f43e2c75100>,
 <matplotlib.lines.Line2D at 0x7f43e2c806a0>,
 <matplotlib.lines.Line2D at 0x7f43e2c80a30>,
 <matplotlib.lines.Line2D at 0x7f43e2c8cfd0>,
 <matplotlib.lines.Line2D at 0x7f43e2c953a0>,
 <matplotlib.lines.Line2D at 0x7f43e3022940>,
 <matplotlib.lines.Line2D at 0x7f43e3022cd0>],
'boxes': [<matplotlib.lines.Line2D at 0x7f43e308f8e0>,
 <matplotlib.lines.Line2D at 0x7f43e2c69280>,
 <matplotlib.lines.Line2D at 0x7f43e2c75bb0>,
 <matplotlib.lines.Line2D at 0x7f43e2c8c520>,
 <matplotlib.lines.Line2D at 0x7f43e2c95e50>],
'medians': [<matplotlib.lines.Line2D at 0x7f43e2c5eb20>,
 <matplotlib.lines.Line2D at 0x7f43e2c75490>,
 <matplotlib.lines.Line2D at 0x7f43e2c80dc0>,
 <matplotlib.lines.Line2D at 0x7f43e2c95730>,
 <matplotlib.lines.Line2D at 0x7f43e302c0a0>],
'fliers': [<matplotlib.lines.Line2D at 0x7f43e2c5eeb0>,
 <matplotlib.lines.Line2D at 0x7f43e2c75820>,
 <matplotlib.lines.Line2D at 0x7f43e2c8c190>,
 <matplotlib.lines.Line2D at 0x7f43e2c95ac0>,
 <matplotlib.lines.Line2D at 0x7f43e302c430>],
'means': []}
```



[62]:
```
df.head(5)
```

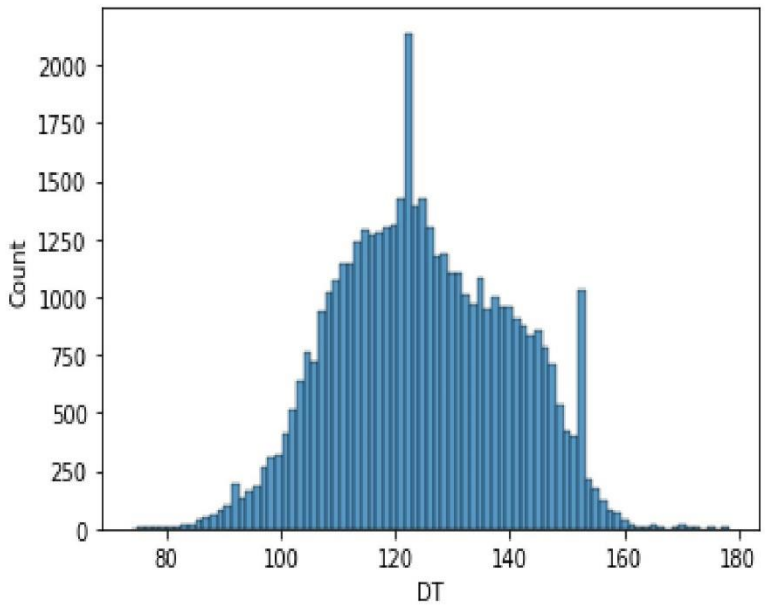
[62]:

	DT	GR	NPHI	RHOB	FACIES
218	75.8412	47.6632	0.4526	2.4314	0
219	76.1991	47.0164	0.4514	2.4413	0
2250	137.8066	61.3278	0.5643	2.1857	0
2251	139.5873	61.9954	0.5611	2.1762	0
2252	140.0185	63.5188	0.5630	2.1946	0

4.7 DT AFTER REMOVING OUTLIER

[63]:
```
sns.histplot(df.DT)
```

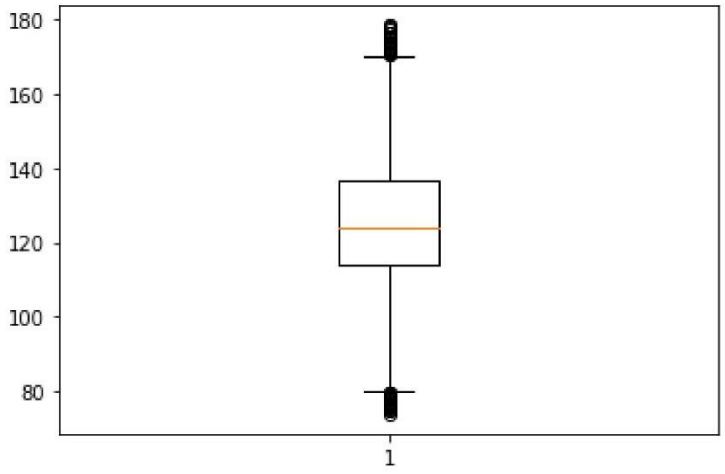
[63]:
```
<AxesSubplot:xlabel='DT', ylabel='Count'>
```



[64]:
```
plt.boxplot(df["DT"])
```

[64]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e2f269d0>,
  <matplotlib.lines.Line2D at 0x7f43e2f26d60>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e2f33130>,
  <matplotlib.lines.Line2D at 0x7f43e2f334c0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e2f26640>],
```

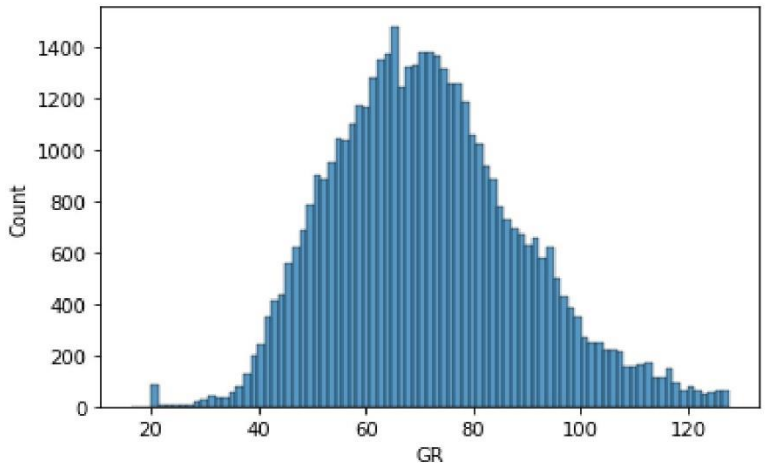
```
'medians': [<matplotlib.lines.Line2D at 0x7f43e2f33850>],
'fliers': [<matplotlib.lines.Line2D at 0x7f43e2f33be0>],
'means': []}
```



4.8 GR AFTER REMOVING OUTLIER

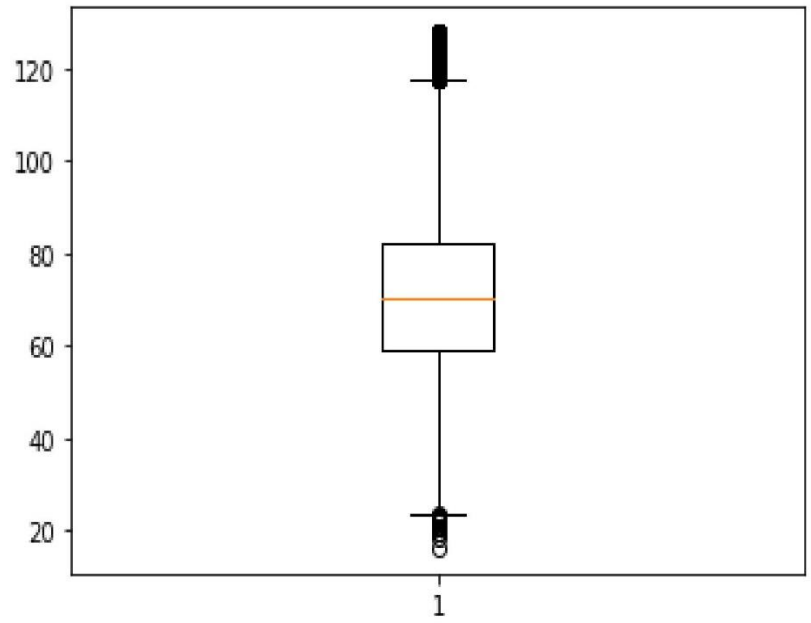
[65]:
```
sns.histplot(df.GR)
```

[65]:
```
<AxesSubplot:xlabel='GR', ylabel='Count'>
```



[66]:
```
plt.boxplot(df.GR)
```

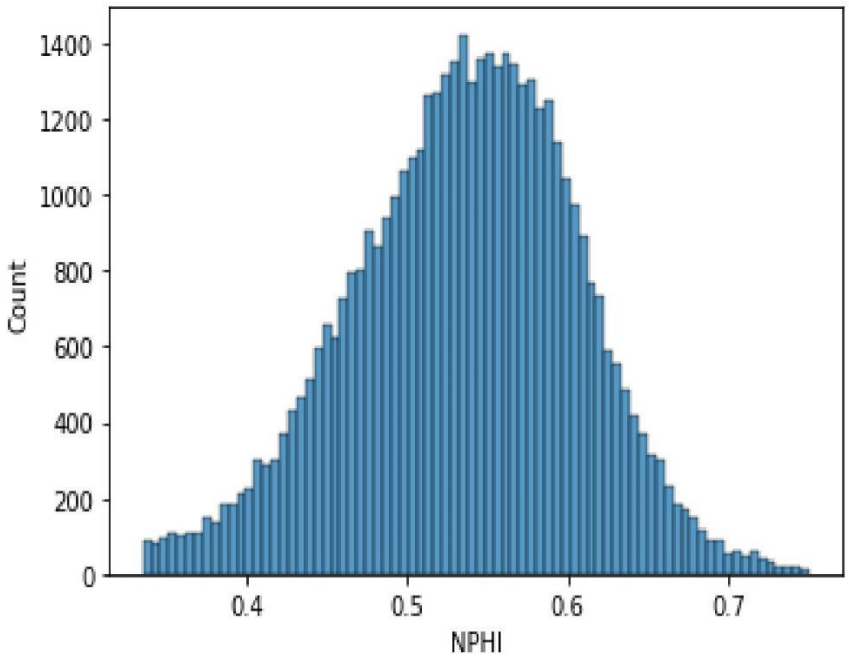
[66]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e2d6b400>,
  <matplotlib.lines.Line2D at 0x7f43e2d6b790>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e2d6bb20>,
  <matplotlib.lines.Line2D at 0x7f43e2d6beb0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e2d6b070>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e2d76280>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e2d76610>],
 'means': []}
```



4.9 NPHI AFTER REMOVING OUTLIER

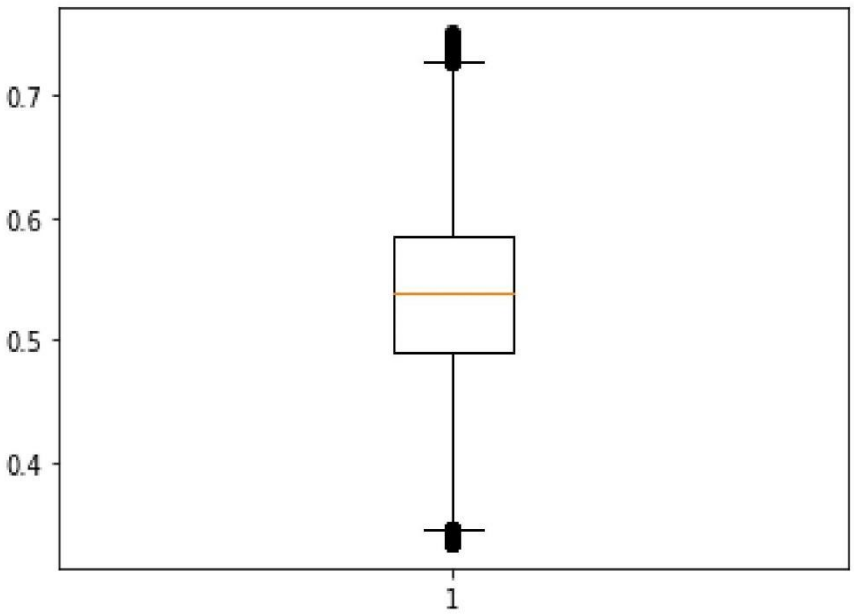
[67]:
```
sns.histplot(df.NPHI)
```

[67]:
```
<AxesSubplot:xlabel='NPHI', ylabel='Count'>
```



[68]:
```
plt.boxplot(df.NPHI)
```

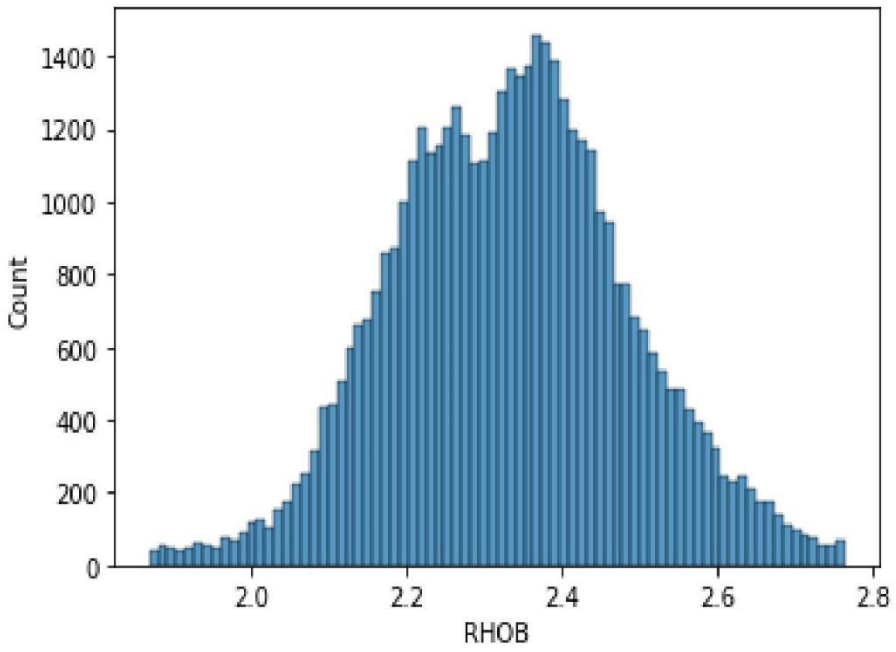
[68]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e2c3ffd0>,
  <matplotlib.lines.Line2D at 0x7f43e2c4d3a0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e2c4d730>,
  <matplotlib.lines.Line2D at 0x7f43e2c4dac0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e2c3fc40>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e2c4de50>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e2c57220>],
 'means': []}
```



4.10 RHOB AFTER REMOVING OUTLIER

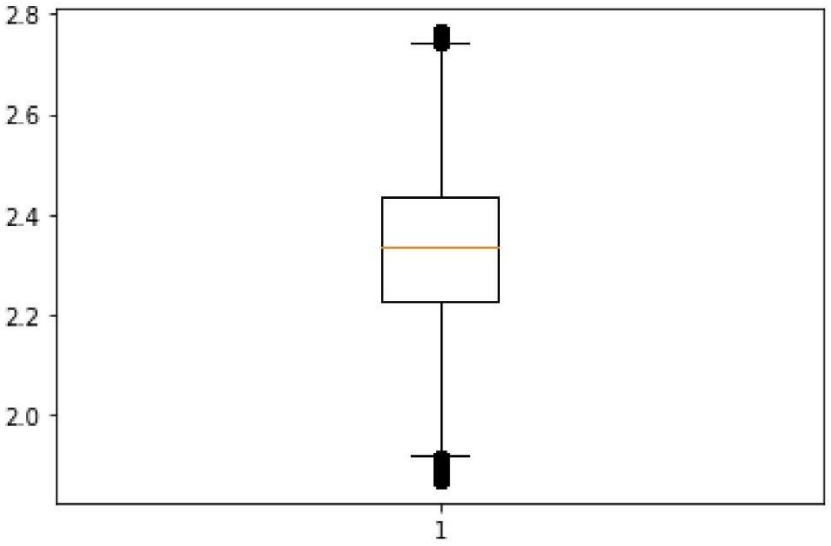
[69]:
```
sns.histplot(df.RHOB)
```

[69]:
```
<AxesSubplot:xlabel='RHOB', ylabel='Count'>
```



[70]:
```
plt.boxplot(df.RHOB)
```

[70]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f43e2a6aa90>,
  <matplotlib.lines.Line2D at 0x7f43e2a6ae20>],
 'caps': [<matplotlib.lines.Line2D at 0x7f43e2a791f0>,
  <matplotlib.lines.Line2D at 0x7f43e2a79580>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f43e2a6a700>],
 'medians': [<matplotlib.lines.Line2D at 0x7f43e2a79910>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f43e2a79ca0>],
 'means': []}
```



[71]:
```
df
```

[71]:

	DT	GR	NPHI	RHOB	FACIES
218	75.8412	47.663200	0.4526	2.4314	0
219	76.1991	47.016400	0.4514	2.4413	0
2250	137.8066	61.327800	0.5643	2.1857	0
2251	139.5873	61.995400	0.5611	2.1762	0
2252	140.0185	63.518800	0.5630	2.1946	0
…	…	…	…	…	…
58494	123.7404	80.913653	0.4993	2.4639	0
58495	123.8728	82.952576	0.5313	2.4660	0
58496	123.3722	84.044079	0.5448	2.4714	0
58497	122.6038	83.725389	0.5364	2.4750	0
58498	122.3045	83.329152	0.5331	2.4709	0

[45392 rows x 5 columns]

FEATURE ENGINEERING:

In machine learning and statistics, **feature selection**, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons:

- simplification of models to make them easier to interpret by researchers/users,
- shorter training times,
- to avoid the curse of dimensionality,
- improve data's compatibility with a learning model class,
- encode inherent symmetries present in the input space.

In this project we have implemented 5 methods of feature engineering-:

- **Variance Threshold**
- **Absolute Correlation**
- **Correlation**
- **SelectKBest**
- **Mutual_Info_Class**

5 FEATURE SELECTION

```
[72]: df.head(10)
```

[72]:

	DT	GR	NPHI	RHOB	FACIES
218	75.8412	47.6632	0.4526	2.4314	0
219	76.1991	47.0164	0.4514	2.4413	0
2250	137.8066	61.3278	0.5643	2.1857	0

```
2251  139.5873  61.9954  0.5611  2.1762       0
2252  140.0185  63.5188  0.5630  2.1946       0
2253  139.3474  64.9925  0.5677  2.1992       0
2254  138.8638  65.6985  0.5743  2.1992       0
2255  139.0847  65.1353  0.5844  2.2009       0
2256  139.2288  63.4583  0.5984  2.2021       0
2257  138.7143  61.7829  0.6146  2.2090       0
```

[73]:
```
df.shape
```

[73]:
```
(45392, 5)
```

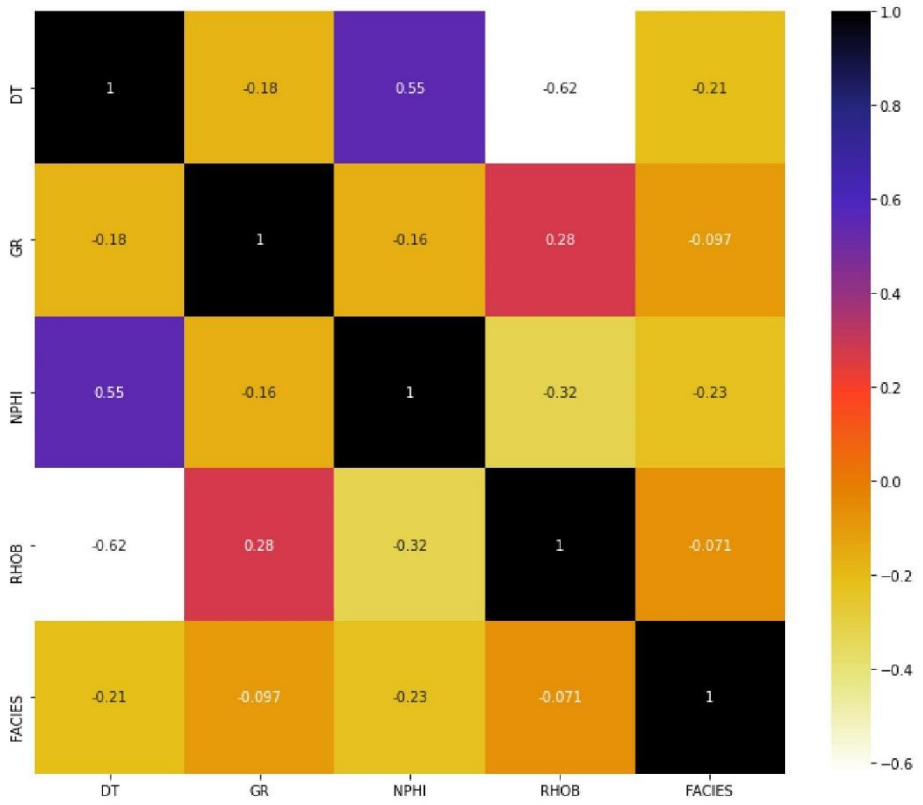
[74]:
```
features = df.shape[1]
features
```

[74]:
```
5
```

[75]:
```
df.var()
```

[75]:
```
DT         230.988291
GR         312.562035
NPHI         0.004939
RHOB         0.023286
FACIES       1.026135
dtype: float64
```

[76]:
```
plt.figure(figsize=(12,10))
cor = df.corr()
sns.heatmap(cor , annot=True , cmap=plt.cm.CMRmap_r)
plt.show()
```



[77]:
```
def FeatureSelection(FeatureSelectionStrategy,dataframe):
    df=dataframe

    if(FeatureSelectionStrategy=="Variance_Threshold"):
        var_thres=VarianceThreshold(threshold=0.0)
        var_thres.fit(df)
        df.columns[var_thres.get_support()]
        cols = [column for column in df.columns
                    if column not in df.columns[var_thres.get_support()]]
        print(cols)
        df = df.drop(cols,axis=1)
        return df

    if(FeatureSelectionStrategy=="Absolute_Correlation"):
        threshold = 0.6
        col_corr = set()
```

```
        corr_matrix = df.corr()
        for i in range(len(corr_matrix.columns)):
            for j in range(i):
                if abs(corr_matrix.iloc[i,j]) > threshold :
                    colname = corr_matrix.columns[i]
                    print(colname)
                    col_corr.add(colname)
        df = df.drop(col_corr,axis=1)
        return df

    if(FeatureSelectionStrategy=="Correlation"):
        threshold = 0.6
        col_corr = set()
        corr_matrix = df.corr()
        for i in range(len(corr_matrix.columns)):
            for j in range(i):
                if (corr_matrix.iloc[i,j]) > threshold :
                    colname = corr_matrix.columns[i]
                    print(colname)
                    col_corr.add(colname)
        df = df.drop(col_corr,axis=1)
        return df

    if(FeatureSelectionStrategy == "SelectKBest"):
        x = df.drop("FACIES",1)
        y = df["FACIES"]
        mutual_info = mutual_info_classif(x,y)
        print(mutual_info)
        mutual_info=pd.Series(mutual_info)
        mutual_info.sort_values(ascending=False)
        mutual_info.sort_values(ascending=False).plot.bar(figsize=(20,8))
        select_col = SelectKBest(mutual_info_classif,k=1)
        select_col.fit(x,y)
        column1 = df.columns[select_col.get_support()]
        df = df.drop(column1,axis=1)
        return df

    if(FeatureSelectionStrategy == "Mutual_Info_Class"):
        x = df.drop("FACIES",1)
        y = df["FACIES"]
        mutual_info = mutual_info_classif(x,y)
        print(mutual_info)
        mutual_info=pd.Series(mutual_info)
        mutual_info.sort_values(ascending=False)
        mutual_info.sort_values(ascending=False).plot.bar(figsize=(20,8))
        return df
```

[78]:
```
FeatureSelectionStrategy=["Variance_Threshold","Absolute_Correlation","Correlation","SelectKBe
optionfeature = 0
df=FeatureSelection(FeatureSelectionStrategy[optionfeature],df)
```

```
[]
```

[79]:
```
print("Deleted feature(s) = " + str(features-df.shape[1]))
```

```
Deleted feature(s) = 0
```

[80]:
```
df
```

[80]:

	DT	GR	NPHI	RHOB	FACIES
218	75.8412	47.663200	0.4526	2.4314	0
219	76.1991	47.016400	0.4514	2.4413	0
2250	137.8066	61.327800	0.5643	2.1857	0
2251	139.5873	61.995400	0.5611	2.1762	0
2252	140.0185	63.518800	0.5630	2.1946	0
…	…	…	…	…	…
58494	123.7404	80.913653	0.4993	2.4639	0
58495	123.8728	82.952576	0.5313	2.4660	0
58496	123.3722	84.044079	0.5448	2.4714	0
58497	122.6038	83.725389	0.5364	2.4750	0
58498	122.3045	83.329152	0.5331	2.4709	0

[45392 rows x 5 columns]

SCALING DATA:

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

In this project 2 methods are used to scale down the data :

- Robust Scaler
- MinMaxScaler

6 SCALING DATA

[81]:
```
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])

    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
```

[82]:
```
scaling_strategy = ["RobustScaler","MinMaxScaler"]
optionscaling = 0
```

```
df = data_scaling( scaling_strategy[optionscaling] , df ,␣
↪DATAConditioningColumns )
```

[83]:
```
df
```

[83]:

	DT	GR	NPHI	RHOB	FACIES
218	-2.123320	-0.960120	-0.908901	0.465184	0
219	-2.107499	-0.987602	-0.921466	0.513056	0
2250	0.615845	-0.379535	0.260733	-0.722921	0
2251	0.694561	-0.351170	0.227225	-0.768859	0
2252	0.713622	-0.286443	0.247120	-0.679884	0
…	…	…	…	…	…
58494	-0.005948	0.452634	-0.419895	0.622340	0
58495	-0.000095	0.539265	-0.084817	0.632495	0
58496	-0.022224	0.585641	0.056545	0.658607	0
58497	-0.056191	0.572100	-0.031414	0.676015	0
58498	-0.069421	0.555265	-0.065969	0.656190	0

[45392 rows x 5 columns]

[84]:
```
df.to_csv("Preprocessed_data.csv",index=False)
```

SPLITTING DATA INTO TEST AND TRAIN:

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

7 SPLITTING DATA USING TRAIN_TEST_SPLIT

[85]:
```
df=pd.read_csv('Preprocessed_data.csv')
```

[86]:
```
df.head()
```

[86]:

	DT	GR	NPHI	RHOB	FACIES
0	-2.123320	-0.960120	-0.908901	0.465184	0
1	-2.107499	-0.987602	-0.921466	0.513056	0
2	0.615845	-0.379535	0.260733	-0.722921	0
3	0.694561	-0.351170	0.227225	-0.768859	0
4	0.713622	-0.286443	0.247120	-0.679884	0

[87]:
```
df.isnull().sum()
```

[87]:
```
DT        0
GR        0
NPHI      0
RHOB      0
FACIES    0
dtype: int64
```

[88]:
```
x = df.drop("FACIES",1)
y = df["FACIES"]
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
↪random_state=8)
```

[89]:
```
X_train.shape
```

[89]:
```
(31774, 4)
```

[90]:
```
X_test.shape
```

[90]:
```
(13618, 4)
```

[91]:
```
X_test
```

[91]:

	DT	GR	NPHI	RHOB
13107	0.585512	0.618141	0.524607	1.169729
24761	-0.492187	-0.387412	0.144503	1.200193
44043	0.314996	0.928135	0.108901	0.179400
17707	0.409511	0.461954	0.432461	0.064797
39859	0.550480	0.761862	-0.366492	-0.692456
…	…	…	…	…
17881	0.034243	-0.048364	0.727749	0.693907
43199	0.641064	0.074669	0.531937	-0.492263
1059	-0.678947	-0.525402	-1.783246	-0.685203
9662	-0.699114	-0.065674	-0.014660	-0.001934
6669	-0.587117	1.145858	1.061780	0.933752

[13618 rows x 4 columns]

MODEL TRAINING:

This is the stage where the ML algorithm is trained by feeding datasets. This is the stage where the learning takes place. Consistent training can significantly improve the prediction rate of the ML model. The weights of the model must be initialized randomly. This way the algorithm will learn to adjust the weights accordingly.

We have used various algorithms to train our model and fitted in one algorithm known as **Voting Classifier.**

A **Voting Classifier** is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output. It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each of them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

For **Voting Classifier,** we have used-

- **Gaussian Naïve Bayes**
- **Logistic Regression (with cross validation and grid search cv to identify most optimum hyperparameter)**
- **Decision Tree Classifier**
- **CatBoostClassifier**
- **XGBClassifier**
- **LGBMClassifier**
- **K-nearest Neighbour**
- **Random forest Classifier**

8 MODEL TRAINING

[92]:
```
estimator=[]
```

[93]:
```
gnb = GaussianNB()
```

[94]:
```
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

grid = {'solver':solvers,'penalty':penalty,'C':c_values}
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
```

```
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.902321 using {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.902216 (0.002300) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.902205 (0.002308) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.900401 (0.001923) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.902226 (0.002296) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.902216 (0.002304) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.900390 (0.001977) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.902279 (0.002317) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.902279 (0.002317) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.900275 (0.001884) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.902321 (0.002006) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.902321 (0.002006) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.899624 (0.001890) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.900716 (0.001635) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.900716 (0.001635) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.899541 (0.001054) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

[95]:
```
dtclf = DecisionTreeClassifier(max_depth=5)
```

[96]:
```
cat = CatBoostClassifier()
```

[97]:
```
xgb= XGBClassifier(learning_rate =0.09,
n_estimators=494,
max_depth=5,
subsample = 0.70,
verbosity = 0,)
```

[98]:
```
lgbm=LGBMClassifier(importance_type = "gain",
verbosity = -1,
max_bin = 60,
num_leaves=300,
boosting_type = 'dart',
learning_rate=0.1,
n_estimators=494,
max_depth=5, )
```

[99]:
```
neigh = KNeighborsClassifier(n_neighbors=3)
```

[100]:
```
rdmclf = RandomForestClassifier(n_estimators=494,max_depth=5)
```

[101]:
```
estimator.append(('gaussian',gnb))
estimator.append(('Gridlogistic',grid_search))
estimator.append(('catboost_classifier',cat))
```

```
estimator.append(('decision_tree',dtclf))
estimator.append(('xgbclassifier',xgb))
estimator.append(('LGBMclassifier',lgbm))
estimator.append(('KNN',neigh))
```

[102]:
```
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
```

[103]:
```
vot_soft.fit(X_train,y_train)
```

```
Learning rate set to 0.094391
0:      learn: 1.3265685        total: 53.9ms   remaining: 53.9s
1:      learn: 1.1441447        total: 60.2ms   remaining: 30s
2:      learn: 1.0098936        total: 66.1ms   remaining: 22s
3:      learn: 0.9070909        total: 72.2ms   remaining: 18s
4:      learn: 0.8233265        total: 78.8ms   remaining: 15.7s
5:      learn: 0.7555721        total: 84.8ms   remaining: 14s
6:      learn: 0.7003256        total: 90.8ms   remaining: 12.9s
7:      learn: 0.6518884        total: 97.1ms   remaining: 12s
8:      learn: 0.6107570        total: 103ms    remaining: 11.4s
9:      learn: 0.5748652        total: 109ms    remaining: 10.8s
10:     learn: 0.5440267        total: 115ms    remaining: 10.4s
11:     learn: 0.5172645        total: 121ms    remaining: 9.97s
12:     learn: 0.4945976        total: 127ms    remaining: 9.63s
13:     learn: 0.4744959        total: 133ms    remaining: 9.39s
14:     learn: 0.4563637        total: 139ms    remaining: 9.14s
15:     learn: 0.4395808        total: 146ms    remaining: 8.95s
16:     learn: 0.4252144        total: 152ms    remaining: 8.78s
17:     learn: 0.4125212        total: 158ms    remaining: 8.61s
18:     learn: 0.4014292        total: 164ms    remaining: 8.46s
19:     learn: 0.3912449        total: 170ms    remaining: 8.33s
20:     learn: 0.3819971        total: 176ms    remaining: 8.21s
21:     learn: 0.3739274        total: 182ms    remaining: 8.1s
22:     learn: 0.3671660        total: 189ms    remaining: 8.04s
23:     learn: 0.3607492        total: 195ms    remaining: 7.94s
24:     learn: 0.3546127        total: 202ms    remaining: 7.87s
25:     learn: 0.3493236        total: 208ms    remaining: 7.78s
26:     learn: 0.3445168        total: 214ms    remaining: 7.7s
27:     learn: 0.3396638        total: 221ms    remaining: 7.66s
28:     learn: 0.3348014        total: 227ms    remaining: 7.6s
29:     learn: 0.3314200        total: 233ms    remaining: 7.54s
30:     learn: 0.3277385        total: 240ms    remaining: 7.49s
31:     learn: 0.3249679        total: 246ms    remaining: 7.43s
32:     learn: 0.3221409        total: 252ms    remaining: 7.38s
33:     learn: 0.3194326        total: 258ms    remaining: 7.32s
34:     learn: 0.3163122        total: 264ms    remaining: 7.29s
35:     learn: 0.3139723        total: 271ms    remaining: 7.25s
36:     learn: 0.3118089        total: 277ms    remaining: 7.2s
37:     learn: 0.3099771        total: 283ms    remaining: 7.16s
```

```
950:    learn: 0.1808604        total: 6.01s    remaining: 309ms
951:    learn: 0.1808094        total: 6.01s    remaining: 303ms
952:    learn: 0.1807565        total: 6.02s    remaining: 297ms
953:    learn: 0.1806931        total: 6.02s    remaining: 290ms
954:    learn: 0.1806400        total: 6.03s    remaining: 284ms
955:    learn: 0.1806084        total: 6.04s    remaining: 278ms
956:    learn: 0.1805417        total: 6.04s    remaining: 272ms
957:    learn: 0.1804648        total: 6.05s    remaining: 265ms
958:    learn: 0.1804367        total: 6.05s    remaining: 259ms
959:    learn: 0.1803765        total: 6.06s    remaining: 253ms
960:    learn: 0.1803410        total: 6.07s    remaining: 246ms
961:    learn: 0.1803168        total: 6.07s    remaining: 240ms
962:    learn: 0.1802625        total: 6.08s    remaining: 234ms
963:    learn: 0.1801915        total: 6.09s    remaining: 227ms
964:    learn: 0.1801313        total: 6.09s    remaining: 221ms
965:    learn: 0.1801005        total: 6.1s     remaining: 215ms
966:    learn: 0.1799676        total: 6.11s    remaining: 208ms
967:    learn: 0.1799060        total: 6.11s    remaining: 202ms
968:    learn: 0.1798571        total: 6.12s    remaining: 196ms
969:    learn: 0.1798203        total: 6.12s    remaining: 189ms
970:    learn: 0.1797556        total: 6.13s    remaining: 183ms
971:    learn: 0.1797036        total: 6.14s    remaining: 177ms
972:    learn: 0.1796488        total: 6.14s    remaining: 170ms
973:    learn: 0.1796218        total: 6.15s    remaining: 164ms
974:    learn: 0.1795801        total: 6.16s    remaining: 158ms
975:    learn: 0.1795017        total: 6.16s    remaining: 152ms
976:    learn: 0.1794816        total: 6.17s    remaining: 145ms
977:    learn: 0.1794352        total: 6.18s    remaining: 139ms
978:    learn: 0.1793843        total: 6.18s    remaining: 133ms
979:    learn: 0.1793591        total: 6.19s    remaining: 126ms
980:    learn: 0.1792623        total: 6.2s     remaining: 120ms
981:    learn: 0.1792027        total: 6.2s     remaining: 114ms
982:    learn: 0.1790638        total: 6.21s    remaining: 107ms
983:    learn: 0.1790086        total: 6.21s    remaining: 101ms
984:    learn: 0.1789560        total: 6.22s    remaining: 94.7ms
985:    learn: 0.1789085        total: 6.23s    remaining: 88.4ms
986:    learn: 0.1788723        total: 6.23s    remaining: 82.1ms
987:    learn: 0.1788502        total: 6.24s    remaining: 75.8ms
988:    learn: 0.1787975        total: 6.25s    remaining: 69.5ms
989:    learn: 0.1787243        total: 6.25s    remaining: 63.2ms
990:    learn: 0.1786958        total: 6.26s    remaining: 56.8ms
991:    learn: 0.1786443        total: 6.27s    remaining: 50.5ms
992:    learn: 0.1786059        total: 6.27s    remaining: 44.2ms
993:    learn: 0.1785555        total: 6.28s    remaining: 37.9ms
994:    learn: 0.1785192        total: 6.29s    remaining: 31.6ms
995:    learn: 0.1784798        total: 6.29s    remaining: 25.3ms
996:    learn: 0.1784028        total: 6.3s     remaining: 19ms
997:    learn: 0.1783568        total: 6.3s     remaining: 12.6ms
```

```
998:    learn: 0.1783011        total: 6.31s    remaining: 6.32ms
999:    learn: 0.1782462        total: 6.32s    remaining: 0us
```

[103]:
```
VotingClassifier(estimators=[('gaussian', GaussianNB()),
                             ('Gridlogistic',
 GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10,
 random_state=1),
                                         error_score=0,
                                         estimator=LogisticRegression(),
                                         n_jobs=-1,
                                         param_grid={'C': [100, 10, 1.0, 0.1,
                                                           0.01],
                                                     'penalty': ['l2'],
                                                     'solver': ['newton-cg',
                                                                'lbfgs',
                                                                'liblinear']},
                                         scoring='accuracy')),
                             ('catboost_classifier',
                              <…
                                          num_parallel_tree=None,
                                          random_state=None, reg_alpha=None,
                                          reg_lambda=None,
                                          scale_pos_weight=None,
                                          subsample=0.7, tree_method=None,
                                          validate_parameters=None,
                                          verbosity=0)),
                             ('LGBMclassifier',
                              LGBMClassifier(boosting_type='dart',
                                             importance_type='gain', max_bin=60,
                                             max_depth=5, n_estimators=494,
                                             num_leaves=300, verbosity=-1)),
                             ('KNN', KNeighborsClassifier(n_neighbors=3))],
                 voting='soft')
```

[104]:
```
y_pred = vot_soft.predict(X_test)
```

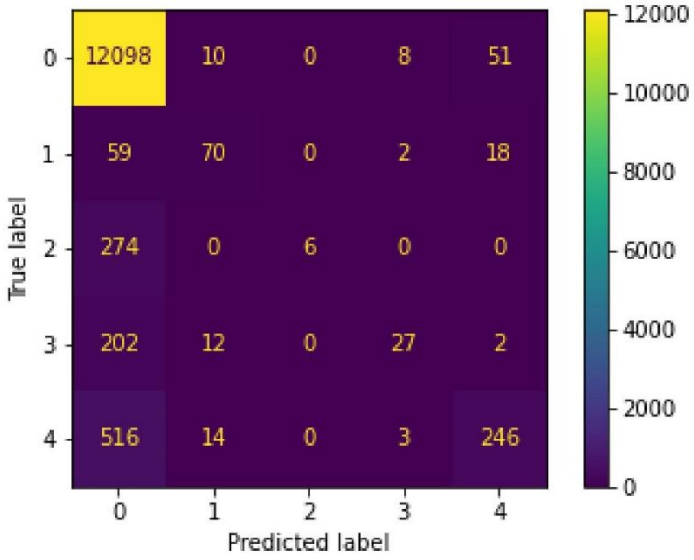
[105]:
```
metrics.accuracy_score(y_test, y_pred)*100
```

[105]:
```
91.40108679688647
```

[106]:
```
t = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix= t, display_labels= vot_soft.
 ↪classes_)
disp.plot()
```

[106]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f44b4c45790>
```

Here we can see that we have got accuracy of 91.4% on train data.



[107]:
```
#metrics.accuracy_score(y_test, y_pred_gnb)*100
```

[108]:
```
#confusion_matrix(y_test, y_pred_gnb)
```

[109]:
```
#t = confusion_matrix(y_test, y_pred_gnb)
#disp = ConfusionMatrixDisplay(confusion_matrix= t, display_labels= gnb.
 ↪classes_)
```

[110]:
```
#disp.plot()
```

[111]:
```
#metrics.accuracy_score(y_test, y_pred_log)*100
```

[112]:
```
#t = confusion_matrix(y_test, y_pred_log)
#disp = ConfusionMatrixDisplay(confusion_matrix= t, display_labels= grid_search.
 ↪classes_)
#disp.plot()
```

[113]:
```
#metrics.accuracy_score(y_test, y_pred_cat)*100
```

[114]:
```
#t = confusion_matrix(y_test, y_pred_cat)
#disp = ConfusionMatrixDisplay(confusion_matrix= t, display_labels= cat.
 ↪classes_)
#disp.plot()
```

PREDICTION ON TESTING DATA:

"Prediction" refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data, in this case to predict Facies.

Before predictions we need to pre-process test data like we have done in the case of training data. Hence, we will call the functions we used to pre-process training data to make our work easier

9 TESTING DATA

[117]:
```
path = '/media/mr-robot/Local Disk/summer_training/test'
os.chdir(path)
```

[118]:
```
# Converting all las files in csv by iterating using lasio
for file in os.listdir():
    if file.endswith(".las"):
        file_path = f"{path}/{file}"
        las=lasio.read(file_path)
        size=len(file_path)
        filepath1=file_path[:size-3]
        las.to_csv(filepath1+'csv', units=False)
```

[119]:
```
## To avoid furthur merging data and redundancy
if(os.path.isfile('./merged_data.csv') ):
    os.remove("merged_data.csv")

if(os.path.isfile('./FACIES_imputed.csv')):
    os.remove("FACIES_imputed.csv")

if(os.path.isfile('./FACIES_TRAIN.csv')):
    os.remove("FACIES_TRAIN.csv")
```

[120]:
```
# Merging all Well Log using Glob
filenames = glob.glob(path + "/*.csv")
dfs = []
for filename in filenames:
    dfs.append(pd.read_csv(filename))
big_frame = pd.concat(dfs, ignore_index=True)
big_frame.to_csv('merged_data.csv',index=False)
```

[121]:
```
df = pd.read_csv('merged_data.csv')
df
```

[121]:

	DEPTH	ACOUSTICIMPEDANCE1	AI	AVG_PIGN	BIT	CALI	\
0	1197.4072	5252.3882	5252388.0	NaN	0.2159	8.9012	
1	1197.5596	5289.7070	5289707.0	NaN	0.2159	8.9005	
2	1197.7120	5245.4429	5245443.0	NaN	0.2159	8.8957	
3	1197.8644	5181.9023	5181902.5	NaN	0.2159	8.8932	

```
4      1198.0168           5131.1343  5131134.5       NaN  0.2159  8.8980
…           …                   …          …         …      …
29560  1689.5065           6013.4722  6013472.5       NaN  0.2159     NaN
29561  1689.6589           5953.0059  5953006.0       NaN  0.2159     NaN
29562  1689.8113           5954.4824  5954482.0       NaN  0.2159     NaN
29563  1689.9637           5911.3301  5911330.0       NaN  0.2159     NaN
29564  1690.1161           5930.9585  5930958.5       NaN  0.2159     NaN

          NPHI        DT  FACIES  FLD1  …  SPSD  ZCOR  BS  CALI[DERIVED]1  \
0       0.4682  133.4417     NaN   NaN  …   NaN   NaN NaN              NaN
1       0.4585  132.4196     NaN   NaN  …   NaN   NaN NaN              NaN
2       0.4543  133.3569     NaN   NaN  …   NaN   NaN NaN              NaN
3       0.4827  134.7392     NaN   NaN  …   NaN   NaN NaN              NaN
4       0.5361  135.7694     NaN   NaN  …   NaN   NaN NaN              NaN
…          …         …      …    …  …   …   …  ..             …
29560      NaN  126.6800     NaN   NaN  …   NaN   NaN NaN              NaN
29561      NaN  127.9872     NaN   NaN  …   NaN   NaN NaN              NaN
29562      NaN  127.9657     NaN   NaN  …   NaN   NaN NaN              NaN
29563      NaN  128.9050     NaN   NaN  …   NaN   NaN NaN              NaN
29564      NaN  128.4784     NaN   NaN  …   NaN   NaN NaN              NaN

        DFL  GRCO  HDRS  HMRS  PHIT  TEMP1
0       NaN   NaN   NaN   NaN   NaN    NaN
1       NaN   NaN   NaN   NaN   NaN    NaN
2       NaN   NaN   NaN   NaN   NaN    NaN
3       NaN   NaN   NaN   NaN   NaN    NaN
4       NaN   NaN   NaN   NaN   NaN    NaN
…    …   …   …   …   …    …
29560   NaN   NaN   NaN   NaN   NaN    NaN
29561   NaN   NaN   NaN   NaN   NaN    NaN
29562   NaN   NaN   NaN   NaN   NaN    NaN
29563   NaN   NaN   NaN   NaN   NaN    NaN
29564   NaN   NaN   NaN   NaN   NaN    NaN

[29565 rows x 55 columns]
```

[122]:
```
#Selecting required feature
df=df[["DT","GR","NPHI","RHOB","FACIES"]]
```

[123]:
```
df
```

[123]:
```
             DT       GR    NPHI    RHOB  FACIES
0      133.4417  87.3154  0.4682  2.2995     NaN
1      132.4196  88.5412  0.4585  2.2981     NaN
2      133.3569  87.5764  0.4543  2.2950     NaN
3      134.7392  86.0361  0.4827  2.2907     NaN
4      135.7694  85.0393  0.5361  2.2856     NaN
```

```
…          …       …      …      …      …
29560  126.6800      NaN    NaN  2.4993    NaN
29561  127.9872      NaN    NaN  2.4997    NaN
29562  127.9657      NaN    NaN  2.4999    NaN
29563  128.9050      NaN    NaN  2.5000    NaN
29564  128.4784      NaN    NaN  2.5000    NaN

[29565 rows x 5 columns]
```

[124]:
```
df=imputing(imputation_strategy[optionimputation],df)
df
```

[124]:
```
              DT          GR      NPHI    RHOB  FACIES
0       133.4417   87.315400  0.468200  2.2995       0
1       132.4196   88.541200  0.458500  2.2981       0
2       133.3569   87.576400  0.454300  2.2950       0
3       134.7392   86.036100  0.482700  2.2907       0
4       135.7694   85.039300  0.536100  2.2856       0
…          …          …        …      …      …
29560  126.6800  102.326070  0.506785  2.4993       0
29561  127.9872  102.490830  0.510428  2.4997       0
29562  127.9657  102.498159  0.510361  2.4999       0
29563  128.9050  102.607440  0.512985  2.5000       0
29564  128.4784  102.560015  0.511792  2.5000       0

[29565 rows x 5 columns]
```

[125]:
```
df = outliers(DATAConditioningStrategy[optionoutlier] , df,␣
 ↪DATAConditioningColumns)
```

```
column DT
Percentiles: 25th=114.139, 75th=137.342, IQR=23.202
InterQuartile Range Outliers-:
             DT          GR    NPHI    RHOB  FACIES
2632    77.7408   55.287400  0.3062  2.6430       0
2633    77.3217   53.629600  0.3052  2.5920       1
3981    75.3027   73.368300  0.5153  2.5090       0
3982    73.6734   73.261800  0.5041  2.4475       0
6097    79.0923   87.085800  0.3700  2.8019       0
6110    76.3801   96.356900  0.3313  2.7004       0
6406    78.6538   59.692300  0.4038  2.6646       0
6448    79.3029   64.718200  0.3632  2.7212       0
13938   79.2984  108.679600  0.4490  2.8759       0
13939   70.9828   95.723000  0.4255  3.0317       0
13940   75.5917   94.711500  0.4245  2.9428       0
15679  175.1408   94.713206  0.5044  2.3501       0
15680  173.8879   96.256515  0.4875  2.3948       0
15706  172.7409   97.818285  0.5074  2.4185       0
```

```
15707  174.8540   97.349782  0.4967  2.4147       0
15708  172.7833   96.989636  0.4784  2.4165       0
16123   76.3119  121.788437  0.3927  3.0026       0
16907  173.0850   78.984443  0.6734  1.8918       0
23404   72.9019   86.674800  0.3879  2.6145       0
23405   73.6668   86.070200  0.3612  2.5231       0
25171   79.3205   78.216300  0.5893  2.2124       0
28926   78.1889   66.276900  0.4540  2.9479       0
(22, 5)
             DT          GR      NPHI    RHOB  FACIES
0      133.4417   87.315400  0.468200  2.2995       0
1      132.4196   88.541200  0.458500  2.2981       0
2      133.3569   87.576400  0.454300  2.2950       0
3      134.7392   86.036100  0.482700  2.2907       0
4      135.7694   85.039300  0.536100  2.2856       0
…         …         …        …     …     …
29560  126.6800  102.326070  0.506785  2.4993       0
29561  127.9872  102.490830  0.510428  2.4997       0
29562  127.9657  102.498159  0.510361  2.4999       0
29563  128.9050  102.607440  0.512985  2.5000       0
29564  128.4784  102.560015  0.511792  2.5000       0

[29534 rows x 5 columns]
column GR
Percentiles: 25th=77.174, 75th=102.911, IQR=25.736
InterQuartile Range Outliers-:
             DT       GR    NPHI    RHOB  FACIES
1342   144.1047  35.5685  0.6130  1.2752       3
1516   115.6053  37.5339  0.6715  1.1197       3
1517   116.5264  37.2150  0.6474  1.2269       3
1625   149.5008  36.3442  0.6133  1.1143       3
1626   150.9417  29.3642  0.6122  1.0951       3
…         …       …     …     …     …
28969  151.0522  27.8672  0.7510  1.0626       3
28970  152.6379  27.9862  0.7093  1.0935       3
28971  154.5247  28.4657  0.6571  1.1246       3
28972  155.4262  29.3424  0.6296  1.1451       3
28973  154.5691  32.9489  0.6210  1.1474       3

[1851 rows x 5 columns]
(1851, 5)
             DT          GR      NPHI    RHOB  FACIES
0      133.4417   87.315400  0.468200  2.2995       0
1      132.4196   88.541200  0.458500  2.2981       0
2      133.3569   87.576400  0.454300  2.2950       0
3      134.7392   86.036100  0.482700  2.2907       0
4      135.7694   85.039300  0.536100  2.2856       0
…         …         …        …     …     …
```

```
29560  126.6800  102.326070  0.506785  2.4993       0
29561  127.9872  102.490830  0.510428  2.4997       0
29562  127.9657  102.498159  0.510361  2.4999       0
29563  128.9050  102.607440  0.512985  2.5000       0
29564  128.4784  102.560015  0.511792  2.5000       0

[27683 rows x 5 columns]
column NPHI
Percentiles: 25th=0.468, 75th=0.550, IQR=0.083
InterQuartile Range Outliers-:
             DT       GR    NPHI    RHOB  FACIES
263    143.7784  72.7236  0.6766  2.1787       0
513    138.5944  75.0486  0.6775  2.2283       0
644    143.2483  78.0601  0.6805  1.9364       0
645    144.5881  78.3862  0.6749  1.7739       3
647    148.7089  60.5277  0.6966  1.3747       3
…         …       …     …      …      …
29028  105.8965  77.9666  0.2990  1.9829       1
29029  105.0871  73.8077  0.2886  1.9849       1
29030  105.3242  68.5815  0.2919  1.9918       1
29031  107.1987  64.1699  0.3269  2.0025       1
29038  113.2466  74.4795  0.3385  1.9897       1

[1568 rows x 5 columns]
(1568, 5)
             DT          GR      NPHI    RHOB  FACIES
0      133.4417   87.315400  0.468200  2.2995       0
1      132.4196   88.541200  0.458500  2.2981       0
2      133.3569   87.576400  0.454300  2.2950       0
3      134.7392   86.036100  0.482700  2.2907       0
4      135.7694   85.039300  0.536100  2.2856       0
…         …          …       …      …      …
29560  126.6800  102.326070  0.506785  2.4993       0
29561  127.9872  102.490830  0.510428  2.4997       0
29562  127.9657  102.498159  0.510361  2.4999       0
29563  128.9050  102.607440  0.512985  2.5000       0
29564  128.4784  102.560015  0.511792  2.5000       0

[26115 rows x 5 columns]
column RHOB
Percentiles: 25th=2.207, 75th=2.416, IQR=0.209
InterQuartile Range Outliers-:
             DT       GR    NPHI    RHOB  FACIES
646    146.9913  72.1231  0.6718  1.5568       3
1228   130.3615  77.5789  0.5451  1.6171       3
1229   133.5854  69.1480  0.5995  1.4461       3
1230   137.1125  58.9514  0.6035  1.4420       3
1231   139.1413  55.2131  0.5432  1.5727       3
```

```
…         …       …      …       …      …
29074   133.7901  78.6751  0.5387  1.8071      0
29125    96.3199  80.4237  0.4219  2.7614      0
29181   131.6097  94.2842  0.4822  1.8686      0
29182   130.0865  81.8287  0.4741  1.7645      0
29183   124.4891  75.3927  0.4875  1.7919      0

[1440 rows x 5 columns]
(1440, 5)
             DT          GR      NPHI    RHOB  FACIES
0      133.4417   87.315400  0.468200  2.2995       0
1      132.4196   88.541200  0.458500  2.2981       0
2      133.3569   87.576400  0.454300  2.2950       0
3      134.7392   86.036100  0.482700  2.2907       0
4      135.7694   85.039300  0.536100  2.2856       0
…         …          …        …      …      …
29560  126.6800  102.326070  0.506785  2.4993       0
29561  127.9872  102.490830  0.510428  2.4997       0
29562  127.9657  102.498159  0.510361  2.4999       0
29563  128.9050  102.607440  0.512985  2.5000       0
29564  128.4784  102.560015  0.511792  2.5000       0

[24675 rows x 5 columns]
```

[126]:
```
df = data_scaling( scaling_strategy[optionscaling] , df ,␣
↪DATAConditioningColumns )
```

[127]:
```
df.to_csv("testing_preprocessed.csv",index=False)
```

[128]:
```
df=pd.read_csv('testing_preprocessed.csv')
```

[129]:
```
df
```

[129]:
```
             DT        GR      NPHI      RHOB  FACIES
0      0.417594 -0.265769 -0.554201 -0.069035       0
1      0.368665 -0.208540 -0.685637 -0.076038       0
2      0.413534 -0.253584 -0.742547 -0.091546       0
3      0.479706 -0.325497 -0.357724 -0.113057       0
4      0.529023 -0.372035  0.365854 -0.138569       0
…         …        …        …        …      …
24670  0.093904  0.435043 -0.031369  0.930465       0
24671  0.156481  0.442736  0.017991  0.932466       0
24672  0.155452  0.443078  0.017092  0.933467       0
24673  0.200417  0.448180  0.052639  0.933967       0
24674  0.179995  0.445966  0.036476  0.933967       0

[24675 rows x 5 columns]
```

[130]:
```
X_testing=df[["DT","GR","NPHI","RHOB"]]
y_testing=df[["FACIES"]]
```

[131]:
```
X_testing.isnull().sum()
```

[131]:
```
DT      0
GR      0
NPHI    0
RHOB    0
dtype: int64
```

[132]:
```
#X_testing=FeatureSelection(FeatureSelectionStrategy[optionfeature],X_testing,y_testing)
```

[ ]:
```

```

[133]:
```
X_testing
```

[133]:

	DT	GR	NPHI	RHOB
0	0.417594	-0.265769	-0.554201	-0.069035
1	0.368665	-0.208540	-0.685637	-0.076038
2	0.413534	-0.253584	-0.742547	-0.091546
3	0.479706	-0.325497	-0.357724	-0.113057
4	0.529023	-0.372035	0.365854	-0.138569
…	…	…	…	…
24670	0.093904	0.435043	-0.031369	0.930465
24671	0.156481	0.442736	0.017991	0.932466
24672	0.155452	0.443078	0.017092	0.933467
24673	0.200417	0.448180	0.052639	0.933967
24674	0.179995	0.445966	0.036476	0.933967

[24675 rows x 4 columns]

[134]:
```
y_testing.describe()
```

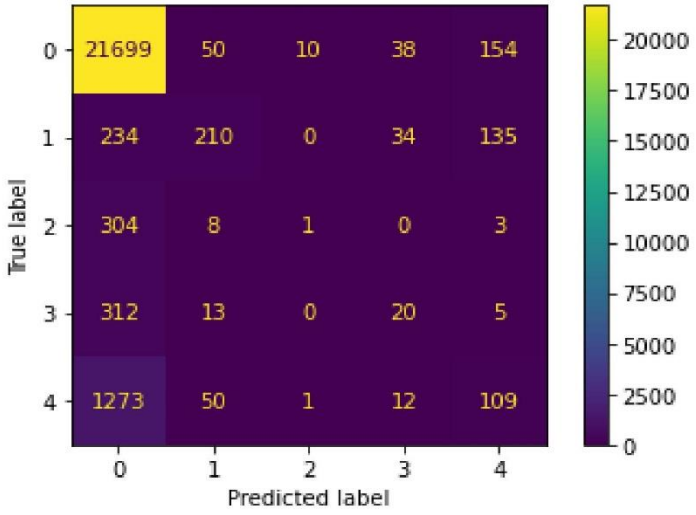
[134]:

	FACIES
count	24675.000000
mean	0.327254
std	1.016689
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	4.000000

[135]:
```
y_predicted = vot_soft.predict(X_testing)
```

[136]:
```
y_predicted
```

```
[136]: array([0, 0, 0, …, 0, 0, 0])

[137]: metrics.accuracy_score(y_testing, y_predicted)*100

[137]: 89.31712259371834

[138]: confusion_matrix(y_testing, y_predicted)

[138]: array([[21699,    50,    10,    38,   154],
              [  234,   210,     0,    34,   135],
              [  304,     8,     1,     0,     3],
              [  312,    13,     0,    20,     5],
              [ 1273,    50,     1,    12,   109]])

[139]: t = confusion_matrix(y_testing, y_predicted)
       disp = ConfusionMatrixDisplay(confusion_matrix= t, display_labels= vot_soft.
        ↪classes_)
       disp.plot()

[139]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
       0x7f44b47c0820>
```




```
[140]: t1=pd.DataFrame(y_testing)

[141]: t1.to_csv('y_given.csv',index=False)

[142]: t2=pd.DataFrame(y_predicted)

[143]: t2.to_csv('y_predicted.csv',index=False)

[ ]:
```

in above screenshot we can see that we have got accuracy of 89.3% on test dataset.

Here are some output files with different method used and different accuracy for Classification Problem.

Imputation strategy used	Outlier removal strategy	Feature selection strategy	Scaling data strategy	Train data accuracy	Test data accuracy	PDF FILE
Iterative imputer	4 Standard Deviation	Variance Threshold	Robust Scaler	**91.49 %**	**89.7%**	main_new_5_1_0_0.pdf
Knn Imputer	4 standard deviation	Correlation	Robust Scaler	**90.01 %**	**89.1%**	main_new_4_2_2_0.pdf
Mean	3 Standard deviation	Variance Threshold	Robust scaler	**90.52 %**	**87.58 %**	main_new_0_0_0_0.pdf
Mean	4 Standard Deviation	Correlation	MinMaxScaler	**90.65 %**	**85.47 %**	main_new_0_1_2_1.pdf
Iterative imputer	Interquartile Range	Variance Threshold	Robust Scaler	**91.13 %**	**89.50 %**	main_new_5_2_0_0.pdf
Knn Imputer	Interquartile Range	Correlation	Robust Scaler	**90.02 %**	**88.61 %**	main_new_4_2_2_0.pdf

REGRESSION

[478]:
```
# Warning Libraries :
import warnings
warnings.filterwarnings("ignore")
```

[479]:
```
# 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
```

[480]:
```
# Data Visualization Libraries :
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
```

[481]:
```
#Libraries to convert .las files to .csv and merge

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

[482]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import KNNImputer
from sklearn.linear_model import LinearRegression
```

[483]:
```
#Feature Selection Libraries
from sklearn.feature_selection import VarianceThreshold
```

[484]:
```
#SCALING LIBRARIES
from sklearn.preprocessing import StandardScaler, MinMaxScaler, Normalizer,␣
 ↪RobustScaler, MaxAbsScaler
```

[485]:
```
#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
```

[486]:
```
#MODEL ACCURACY LIBRARIES
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

[487]:
```
path='/media/mr-robot/Local Disk/summer_training/Train'
os.chdir(path)
```

[488]:
```
df = pd.read_csv('merged_data.csv')
df
```

[488]:

	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]
```

[489]:
```
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
```

[490]:
```
#Selecting required feature
df=df[["GR","RHOB","NPHI","DT"]]
```

[491]:
```
df.isnull().sum()
```

[491]:
```
GR      120
RHOB      0
NPHI    327
DT        0
```

```
dtype: int64
```

[492]:
```
df= df.dropna(axis=0)
```

[493]:
```
df
```

[493]:

	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]

[495]:
```
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)
```

[496]:
```
X_train.shape
```

[496]:
```
(46477, 3)
```

[497]:
```
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"]
```

[498]:
```
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
…          …     …       …        …
```

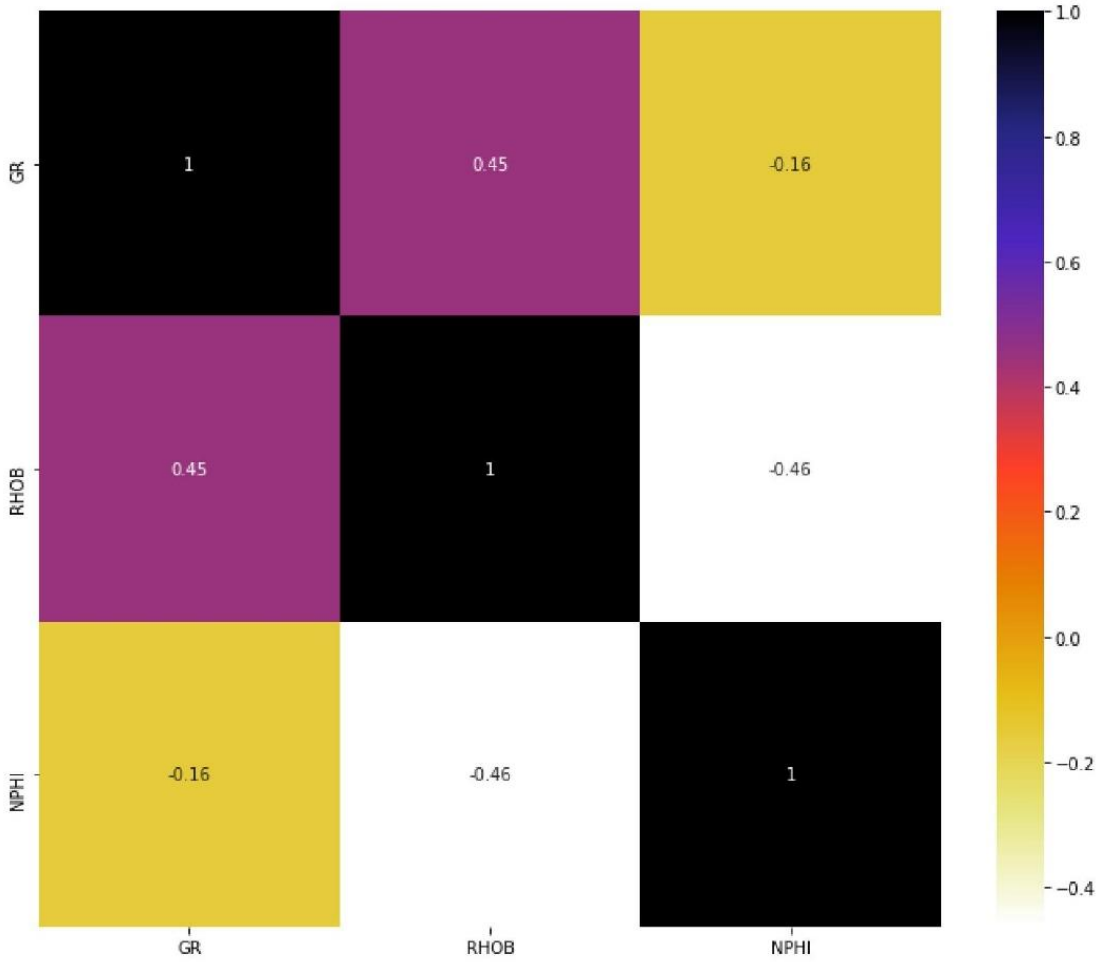
```
55488  103.1246  2.5150  0.4686   98.6188
50169   84.2108  2.3961  0.4774  108.7165
27063   58.8217  2.4845  0.5033  103.9533
8366    69.2729  2.0863  0.6274  147.9099
17530   68.4194  2.2210  0.3759  107.0126

[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]
```

[499]:
```
X_train
```

[499]:
```
             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]
```

[500]:
```
plt.figure(figsize=(12,10))
cor = X_train.corr()
sns.heatmap(cor , annot=True , cmap=plt.cm.CMRmap_r)
plt.show()
```



[501]:
```
X_train.var()
```

[501]:
```
GR      621.097210
RHOB      0.093920
NPHI      0.009019
dtype: float64
```

[502]:
```
X_train.corr()
```

[502]:
```
             GR      RHOB      NPHI
GR    1.000000  0.447292 -0.157540
RHOB  0.447292  1.000000 -0.463009
NPHI -0.157540 -0.463009  1.000000
```

[504]:
```
X_train['NPHI2'] = 2 * X_train.NPHI
X_train.drop('NPHI',1)
```

[504]:
```
             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]
```

[505]:
```
X_train
```

[505]:
```
             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)

[506]:
```
X_train
```

[506]:

	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]

[507]:
```
X_train.corr()
```

[507]:

	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

[508]:
```
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)

[510]:
```
X_test['NPHI2'] = 2 * X_test.NPHI
X_test.drop('NPHI',1)
```

[510]:

	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]

[511]:
```
cat = CatBoostRegressor()
```

[512]:
```
cat.fit(X_train,y_train)
```

```
Learning rate set to 0.076527
0:      learn: 25.2606402       total: 2.99ms   remaining: 2.99s
1:      learn: 24.5182811       total: 5.73ms   remaining: 2.86s
2:      learn: 23.8395858       total: 8.19ms   remaining: 2.72s
3:      learn: 23.2385256       total: 10.6ms   remaining: 2.63s
4:      learn: 22.7179691       total: 13.4ms   remaining: 2.66s
5:      learn: 22.2549527       total: 16ms     remaining: 2.65s
6:      learn: 21.8298942       total: 18.6ms   remaining: 2.64s
7:      learn: 21.4559203       total: 21.3ms   remaining: 2.64s
8:      learn: 21.1198636       total: 24.7ms   remaining: 2.72s
9:      learn: 20.8178518       total: 27.1ms   remaining: 2.69s
10:     learn: 20.5482451       total: 29.7ms   remaining: 2.67s
11:     learn: 20.3021294       total: 32ms     remaining: 2.63s
12:     learn: 20.0725712       total: 34.3ms   remaining: 2.6s
13:     learn: 19.8768744       total: 36.8ms   remaining: 2.59s
14:     learn: 19.7047498       total: 39.1ms   remaining: 2.57s
15:     learn: 19.5451385       total: 41.7ms   remaining: 2.56s
16:     learn: 19.4099704       total: 44.1ms   remaining: 2.55s
17:     learn: 19.2946994       total: 46.6ms   remaining: 2.54s
18:     learn: 19.1876368       total: 49ms     remaining: 2.53s
19:     learn: 19.0818952       total: 51.4ms   remaining: 2.52s
20:     learn: 18.9195356       total: 53.7ms   remaining: 2.5s
21:     learn: 18.8230745       total: 56.1ms   remaining: 2.5s
22:     learn: 18.7161046       total: 58.8ms   remaining: 2.5s
23:     learn: 18.6273215       total: 61.3ms   remaining: 2.49s
24:     learn: 18.5548344       total: 63.4ms   remaining: 2.47s
25:     learn: 18.4899022       total: 65.7ms   remaining: 2.46s
26:     learn: 18.4276311       total: 68.6ms   remaining: 2.47s
27:     learn: 18.3854681       total: 70.9ms   remaining: 2.46s
```

```
988:    learn: 14.5487315       total: 2.3s     remaining: 25.6ms
989:    learn: 14.5478657       total: 2.31s    remaining: 23.3ms
990:    learn: 14.5472582       total: 2.31s    remaining: 21ms
991:    learn: 14.5468474       total: 2.31s    remaining: 18.6ms
992:    learn: 14.5464146       total: 2.31s    remaining: 16.3ms
993:    learn: 14.5455159       total: 2.32s    remaining: 14ms
994:    learn: 14.5451597       total: 2.32s    remaining: 11.7ms
995:    learn: 14.5447414       total: 2.32s    remaining: 9.32ms
996:    learn: 14.5441536       total: 2.32s    remaining: 6.99ms
997:    learn: 14.5432415       total: 2.33s    remaining: 4.66ms
998:    learn: 14.5421538       total: 2.33s    remaining: 2.33ms
999:    learn: 14.5416389       total: 2.33s    remaining: 0us
```

[512]: <catboost.core.CatBoostRegressor at 0x7fa94e9defd0>

[513]:
```
y_pred = cat.predict(X_test)
```

[514]:
```
r2_score(y_test,y_pred)
```

[514]: 0.6215964899517009

ALGORITHM USED	ACCURACY	PDF
Cat Boost Regressor	**62.1%**	test1.pdf
Linear Regression	**29.26%**	linear_regression.pdf
Gradient Boosting	**57.85%**	GRADIENT.pdf

8. FUTURE SCOPE

- We can do some more data wrangling and apply some neural network algorithms to increase efficiency and accuracy of our machine learning model.

- We can make an app to determine Facies and Synthetic curve in real time using machine learning.

9. CONCLUSION

The huge Well Log datasets have been converted and refined into proper meaningful datasets which are in proper arrangement and scale to be used in predictions for various features in the oil and gas industry.

The Summer Training has helped me to get acquainted with all the on-going activities of GEOPIC, ONGC at Dehradun and various responsibilities of this group. Involvement in development activities have also given me the opportunity to explore, learn and know various technologies, languages, and programming platforms. I would once again like to express heartfelt gratitude to all those who contributed towards making my training a rich experience and to the staff of ONGC for assisting me throughout the project.