In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
raw_dataset=pd.read_csv("S1V2.csv")
```

In [3]:
```
S1V2 = raw_dataset.copy()
S1V2.head()
```

Out[3]:

	Well	S1_mg_oil/g_TOC	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
0	Dadas_Dogan_1	3.08	0.44	2.49	10.60	0.70	33.7	22.8	30.5
1	Dadas_Dogan_1	3.53	0.44	2.52	11.80	0.73	22.5	61.5	14.1
2	Dadas_Dogan_1	3.34	0.44	2.54	9.74	0.71	15.8	68.6	13.9
3	Dadas_Dogan_1	2.12	0.44	2.57	4.50	0.81	17.0	15.3	63.2
4	Dadas_Dogan_1	2.63	0.44	2.60	4.20	0.63	47.4	13.4	22.6

In [4]:
```
S1V2.shape
```

Out[4]:
```
(1324, 9)
```

In [5]:
```
S1V2.describe()
```

Out[5]:

	S1_mg_oil/g_TOC	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
count	1324.000000	1324.000000	1324.000000	1324.000000	1324.000000	1324.000000	1324.000000	1324.000000
mean	2.910211	0.243353	2.339335	3.977613	0.899849	23.889758	30.663867	33.934169
std	2.133312	0.143739	0.992484	2.036037	0.200838	11.846613	15.686748	25.604037
min	0.100000	0.090000	1.110000	0.800000	0.580000	1.440000	0.000000	0.000000
25%	1.370000	0.090000	1.170000	2.540000	0.780000	15.720000	16.480000	10.800000
50%	2.440000	0.340000	2.160000	3.590000	0.840000	22.900000	31.000000	27.000000
75%	3.790000	0.360000	3.110000	5.200000	0.960000	32.460000	40.000000	56.540000
max	11.250000	0.440000	4.870000	11.800000	1.600000	53.330000	71.300000	95.160000

In [6]:
```
S1V2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1324 entries, 0 to 1323
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Well              1324 non-null   object 
 1   S1_mg_oil/g_TOC   1324 non-null   float64
 2   Age_BA            1324 non-null   float64
 3   Present_Depth_km  1324 non-null   float64
 4   TOC_%             1324 non-null   float64
 5   Requ_%            1324 non-null   float64
 6   Quartz_%          1324 non-null   float64
 7   Clay_%            1324 non-null   float64
 8   Carbonate_%       1324 non-null   float64
dtypes: float64(8), object(1)
memory usage: 93.2+ KB
```

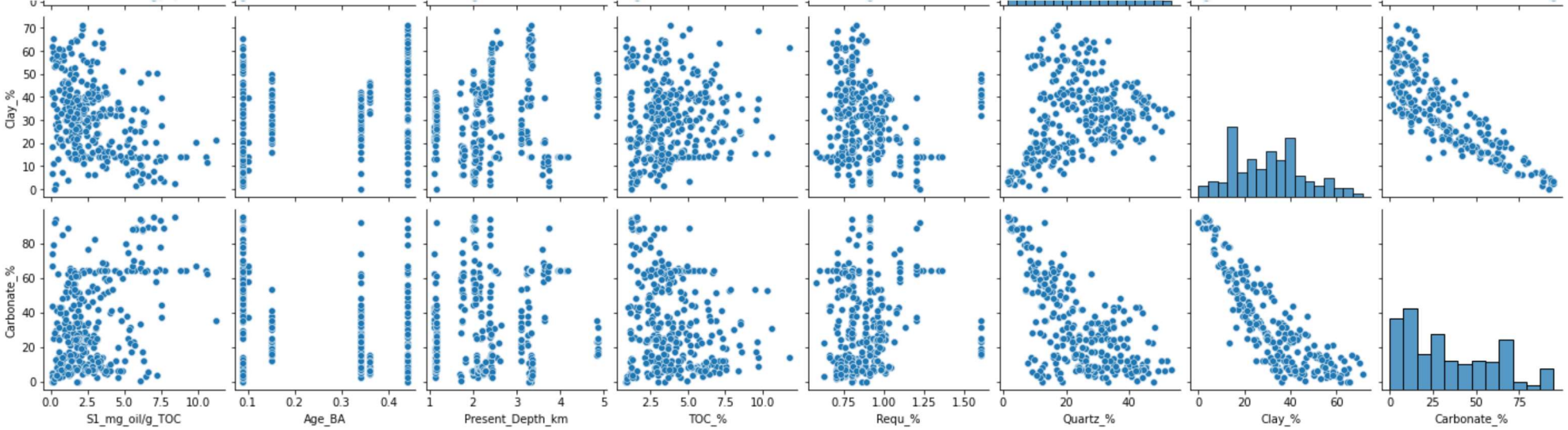
In [7]:
```
corr_matrix =S1V2.corr()
```

In [8]:
```
corr_matrix["S1_mg_oil/g_TOC"].sort_values(ascending=False)
```

Out[8]:
```
S1_mg_oil/g_TOC     1.000000
Carbonate_%         0.359380
TOC_%               0.324155
Present_Depth_km    0.185384
Requ_%              0.111920
Quartz_%           -0.175566
Clay_%             -0.405619
Age_BA             -0.453934
Name: S1_mg_oil/g_TOC, dtype: float64
```

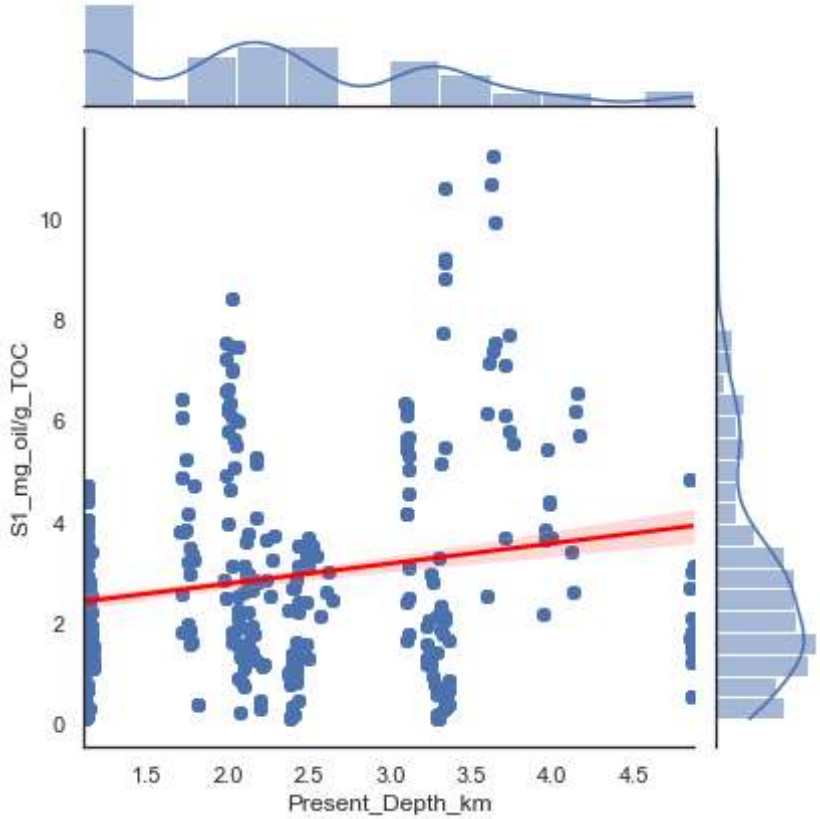
In [9]:
```
sns.pairplot(S1V2)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x17b4a42d310>


Clay_%
Carbonate_%
S1_mg_oil/g_TOC
Age_BA
Present_Depth_km
TOC_%
Requ_%
Quartz_%
Clay_%
Carbonate_%

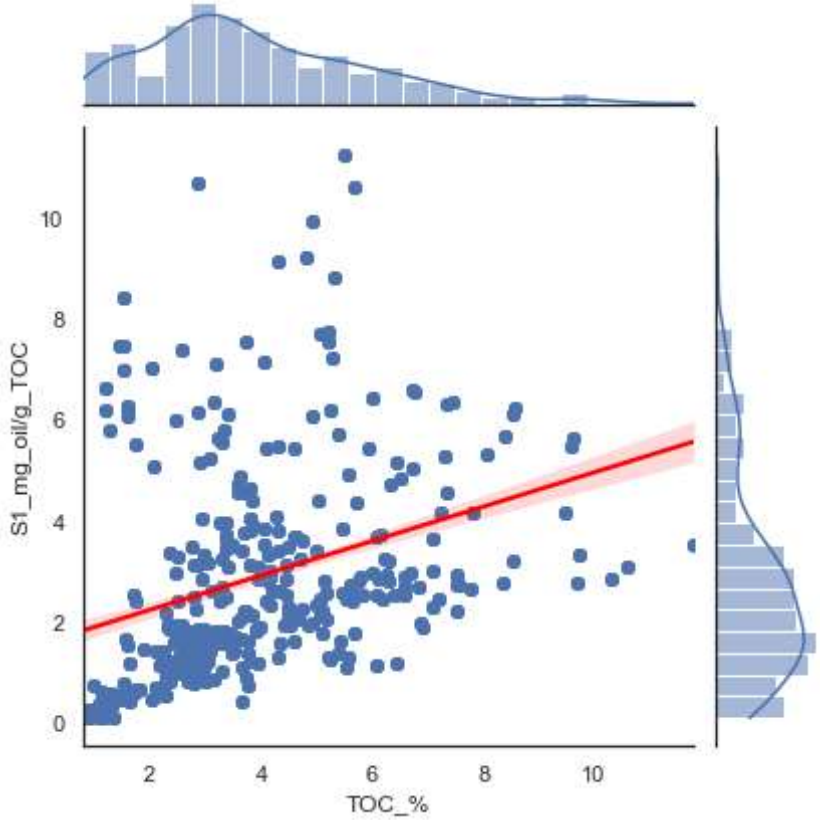
In [10]:
```
sns.set_theme(style="white")
plt.figure(figsize = (40,5), dpi = (100))
sns.jointplot(x = S1V2['Present_Depth_km'], y = S1V2['S1_mg_oil/g_TOC'], kind='reg', line_kws={"color": "red"})
```

Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x17b4a2c5940>

<Figure size 4000x500 with 0 Axes>
```



In [11]:
```
sns.set_theme(style="white")
plt.figure(figsize = (40,5), dpi = (100))
sns.jointplot(x = S1V2['TOC_%'], y = S1V2['S1_mg_oil/g_TOC'], kind='reg', line_kws={"color": "red"})
```

Out[11]:
```
<seaborn.axisgrid.JointGrid at 0x17b4e380b20>

<Figure size 4000x500 with 0 Axes>
```



In [12]:
```
X = S1V2.iloc[:, 2:].values
y = S1V2.iloc[:, 1].values
```

In [13]:
```
X
```

Out[13]:
```
array([[ 0.44,  2.49, 10.6 , ..., 33.7 , 22.8 , 30.5 ],
       [ 0.44,  2.52, 11.8 , ..., 22.5 , 61.5 , 14.1 ],
       [ 0.44,  2.54,  9.74, ..., 15.8 , 68.6 , 13.9 ],
       ...,
       [ 0.36,  2.26,  5.83, ..., 42.63, 41.56,  6.  ],
       [ 0.36,  2.27,  6.22, ..., 41.39, 40.33, 11.  ],
       [ 0.36,  2.28,  6.14, ..., 42.04, 42.04,  6.  ]])
```

In [14]:
```
y
```

Out[14]:
```
array([3.08, 3.53, 3.34, ..., 2.52, 3.23, 3.74])
```

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [16]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [17]:
```
X_train
```

Out[17]:
```
array([[-1.05128267, -0.28784462, -0.75511348, ..., -1.36831886,
        -1.08617345,  1.71072842],
       [ 0.68465536, -1.22754777, -1.17223165, ...,  1.2151847 ,
        -0.4909979 ,  0.27019623],
       [-1.05128267,  1.36163432, -0.28774012, ..., -0.68320949,
        -1.2490636 ,  1.23055102],
       ...,
       [ 0.68465536, -1.22754777, -1.17223165, ...,  1.2151847 ,
        -0.4909979 ,  0.27019623],
       [-0.63465754,  0.75182695,  1.93353973, ..., -0.1347845 ,
        -0.92954831,  0.73676536],
       [ 1.37903057,  0.07204169, -0.47368436, ...,  0.3123928 ,
         1.92102932, -1.18394423]])
```

In [18]:
```
from tensorflow.keras.layers import Input, Dense, Activation,Dropout
from tensorflow.keras.models import Model
```

In [19]:
```
input_layer = Input(shape=(X.shape[1],))
dense_layer_1 = Dense(1024, activation='relu')(input_layer)
dense_layer_2 = Dense(512, activation='relu')(dense_layer_1)
dense_layer_3 = Dense(256, activation='relu')(dense_layer_2)
dense_layer_4 = Dense(128, activation='relu')(dense_layer_3)
dense_layer_5 = Dense(128, activation='relu')(dense_layer_3)
output = Dense(1)(dense_layer_5)

model = Model(inputs=input_layer, outputs=output)
model.compile(loss="mean_squared_error" , optimizer="adam", metrics=["mean_squared_error"])
```

In [20]:
```
my_model = model
```

In [21]:
```
my_model.summary()
```

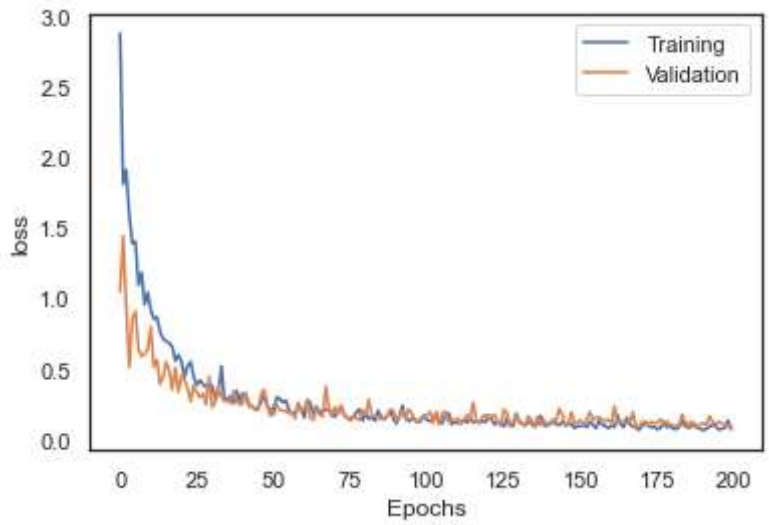
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 7)]               0
_________________________________________________________________
dense (Dense)                (None, 1024)              8192
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 697,345
Trainable params: 697,345
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
history = model.fit(X_train, y_train, batch_size=1, epochs=200, verbose=1, validation_split=0.2)
```

```
Epoch 1/200
794/794 [==============================] - 8s 10ms/step - loss: 2.8813 - mean_squared_error: 2.8813 - val_loss: 1.0491 - val_me
an_squared_error: 1.0491
Epoch 2/200
794/794 [==============================] - 9s 12ms/step - loss: 1.8119 - mean_squared_error: 1.8119 - val_loss: 1.4443 - val_me
an_squared_error: 1.4443
Epoch 3/200
794/794 [==============================] - 13s 17ms/step - loss: 1.9127 - mean_squared_error: 1.9127 - val_loss: 0.9473 - val_m
ean_squared_error: 0.9473
Epoch 4/200
794/794 [==============================] - 15s 19ms/step - loss: 1.5916 - mean_squared_error: 1.5916 - val_loss: 0.5142 - val_m
ean_squared_error: 0.5142
Epoch 5/200
794/794 [==============================] - 15s 19ms/step - loss: 1.3913 - mean_squared_error: 1.3913 - val_loss: 0.8587 - val_m
ean_squared_error: 0.8587
Epoch 6/200
794/794 [==============================] - 13s 17ms/step - loss: 1.4070 - mean_squared_error: 1.4070 - val_loss: 0.9082 - val_m
ean_squared_error: 0.9082
Epoch 7/200
```

In [23]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.legend(["Training","Validation"])
```

Out[23]: <matplotlib.legend.Legend at 0x17b4f1b0e50>



In [24]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
pred_train = model.predict(X_train)
print(np.sqrt(mean_squared_error(y_train,pred_train)))
pred = model.predict(X_test)
print(np.sqrt(mean_squared_error(y_test,pred)))
```

```
0.2544896026742563
0.3174872612227883
```

In [25]:
```
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
11/11 [==============================] - 0s 33ms/step - loss: 0.1008 - mean_squared_error: 0.1008
Test Score: 0.10079817473888397
Test Accuracy: 0.10079817473888397
```

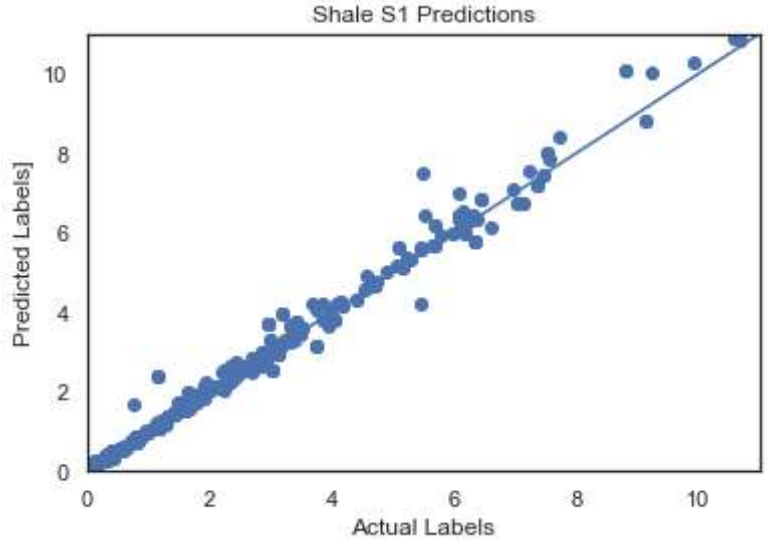
In [26]:
```
predictions = model.predict(X_test)
np.set_printoptions(suppress=True)
print('Predicted labels: ', np.round(predictions)[:10])
print('Actual labels : ' ,y_test[:10])
```

```
Predicted labels:  [[5.]
 [4.]
 [0.]
 [1.]
 [3.]
 [1.]
 [4.]
 [4.]
 [3.]
 [4.]]
Actual labels :  [4.91 3.43 0.18 1.41 3.   1.46 3.32 3.73 3.08 5.45]
```

In [27]:
```
from sklearn.metrics import r2_score
y_true = np.round(predictions)
y_pred = y_test
r2_score(y_true, y_pred)
```

Out[27]:
```
0.9682033980996608
```

In [28]:
```
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels]')
plt.title('Shale S1 Predictions')
lims = [0, 11]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
```



In [29]:
```
my_model.save('./saved_models/my_tf_model')
```

```
WARNING:tensorflow:From C:\Users\90532\anaconda3\lib\site-packages\tensorflow\python\training\tracking\tracking.py:111: Model.sta
te_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
WARNING:tensorflow:From C:\Users\90532\anaconda3\lib\site-packages\tensorflow\python\training\tracking\tracking.py:111: Layer.upd
ates (from tensorflow.python.keras.engine.base_layer) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
INFO:tensorflow:Assets written to: ./saved_models/my_tf_model\assets
```

In [30]:
```
my_tf_saved_model = tf.keras.models.load_model(
 './saved_models/my_tf_model')
my_tf_saved_model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 7)]               0
_________________________________________________________________
dense (Dense)                (None, 1024)              8192
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 697,345
Trainable params: 697,345
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
from tensorflow.keras.models import save_model, load_model
import pandas as pd
```

In [46]:
```
model = load_model('./saved_models/my_tf_model',
 custom_objects=None,
 compile=True)
```

In [47]:
```
raw_dataset=pd.read_csv("Marcellus1and2S1prediction.csv",sep=",")
```

In [48]:
```
Marcellus1and2S1prediction= raw_dataset.copy()
Marcellus1and2S1prediction.head()
```

Out[48]:

	Well	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
0	Marcellus_1	0.39	1.53	2.3	1.0	26.8	42.4	19.7
1	Marcellus_1	0.39	1.53	5.1	1.0	24.8	52.4	2.6
2	Marcellus_1	0.39	1.54	6.0	1.0	22.8	41.9	12.0
3	Marcellus_1	0.39	1.54	3.2	1.0	27.9	49.8	4.4
4	Marcellus_1	0.39	1.54	3.8	1.0	30.1	48.9	2.9

In [49]:
```
Marcellus1and2S1prediction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Well              14 non-null     object 
 1   Age_BA            14 non-null     float64
 2   Present_Depth_km  14 non-null     float64
 3   TOC_%             14 non-null     float64
 4   Requ_%            14 non-null     float64
 5   Quartz_%          14 non-null     float64
 6   Clay_%            14 non-null     float64
 7   Carbonate_%       14 non-null     float64
dtypes: float64(7), object(1)
memory usage: 1.0+ KB
```

In [50]:
```
Marcellus1and2S1prediction.describe()
```

Out[50]:

	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
count	14.00	14.000000	14.000000	14.000000	14.000000	14.000000	14.000000
mean	0.39	2.000000	6.928571	1.371429	29.378571	39.414286	6.728571
std	0.00	0.640493	3.760816	0.518027	6.227875	8.358861	4.757573
min	0.39	1.530000	2.300000	1.000000	19.600000	22.300000	2.400000
25%	0.39	1.540000	3.500000	1.000000	25.300000	34.375000	3.450000
50%	0.39	1.545000	6.200000	1.000000	29.300000	40.300000	5.450000
75%	0.39	2.820000	9.725000	2.000000	32.250000	44.275000	7.925000
max	0.39	2.840000	13.600000	2.100000	44.400000	52.400000	19.700000

In [51]:
```
X_new = Marcellus1and2S1prediction.iloc[:, 1:].values
```

In [52]:
```
X_new
```

Out[52]:
```
array([[ 0.39,  1.53,  2.3 ,  1.  , 26.8 , 42.4 , 19.7 ],
       [ 0.39,  1.53,  5.1 ,  1.  , 24.8 , 52.4 ,  2.6 ],
       [ 0.39,  1.54,  6.  ,  1.  , 22.8 , 41.9 , 12.  ],
       [ 0.39,  1.54,  3.2 ,  1.  , 27.9 , 49.8 ,  4.4 ],
       [ 0.39,  1.54,  3.8 ,  1.  , 30.1 , 48.9 ,  2.9 ],
       [ 0.39,  1.54,  8.1 ,  1.  , 28.5 , 38.  ,  4.9 ],
       [ 0.39,  1.54, 10.2 ,  1.  , 30.7 , 34.1 ,  2.4 ],
       [ 0.39,  1.55, 12.8 ,  1.  , 23.1 , 31.  ,  6.2 ],
       [ 0.39,  1.55, 13.6 ,  1.  , 19.6 , 30.3 , 10.6 ],
       [ 0.39,  2.82,  3.4 ,  2.  , 33.6 , 44.9 ,  3.4 ],
       [ 0.39,  2.82,  3.1 ,  2.  , 35.4 , 40.7 ,  8.1 ],
       [ 0.39,  2.83,  6.4 ,  2.  , 32.7 , 39.9 ,  3.6 ],
       [ 0.39,  2.83,  8.3 ,  2.1 , 30.9 , 35.2 ,  7.4 ],
       [ 0.39,  2.84, 10.7 ,  2.1 , 44.4 , 22.3 ,  6.  ]])
```

In [53]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_new = sc.fit_transform(X_new)
```

In [54]:
```
X_new
```

Out[54]:
```
array([[ 1.        , -0.76151083, -1.27719508, -0.74407295, -0.42966662,
         0.37067518,  2.82940193],
       [ 1.        , -0.76151083, -0.5045709 , -0.74407295, -0.76292604,
         1.612171  , -0.90054753],
       [ 1.        , -0.74530847, -0.25622741, -0.74407295, -1.09618547,
         0.30860039,  1.14983404],
       [ 1.        , -0.74530847, -1.02885159, -0.74407295, -0.24637393,
         1.28938209, -0.50792127],
       [ 1.        , -0.74530847, -0.86328927, -0.74407295,  0.12021144,
         1.17764747, -0.83510982],
       [ 1.        , -0.74530847,  0.32324073, -0.74407295, -0.1463961 ,
        -0.17558298, -0.39885842],
       [ 1.        , -0.74530847,  0.90270887, -0.74407295,  0.22018926,
        -0.65976635, -0.94417267],
       [ 1.        , -0.72910611,  1.62014561, -0.74407295, -1.04619655,
        -1.04463006, -0.11529501],
       [ 1.        , -0.72910611,  1.84089538, -0.74407295, -1.62940055,
        -1.13153476,  0.84445806],
       [ 1.        ,  1.32859336, -0.97366415,  1.25920038,  0.70341543,
         0.68104914, -0.72604697],
       [ 1.        ,  1.32859336, -1.05644531,  1.25920038,  1.00334891,
         0.15962089,  0.29914382],
       [ 1.        ,  1.34479572, -0.14585252,  1.25920038,  0.55344869,
         0.06030123, -0.68242183],
       [ 1.        ,  1.34479572,  0.37842817,  1.45952771,  0.25351521,
        -0.52320181,  0.14645583],
       [ 1.        ,  1.36099808,  1.04067747,  1.45952771,  2.50301633,
        -2.12473142, -0.15892015]])
```

In [55]:
```
print(model.predict(X_new))
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000020FE7AA
6700> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @t
f.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that re
laxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customi
zation/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) a
nd https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more deta
ils.
[[3.7120557]
 [2.849203 ]
 [4.387171 ]
 [1.7884814]
 [1.4356035]
 [4.7926135]
 [2.6676564]
 [5.8909554]
 [7.2012734]
 [1.1466393]
 [1.3902717]
 [2.9217923]
 [3.9973416]
 [2.667465 ]]
```

In [56]:
```
from tensorflow.keras.models import save_model, load_model
import pandas as pd
```

In [57]:
```
model = load_model('./saved_models/my_tf_model',
 custom_objects=None,
 compile=True)
```

In [58]:
```
raw_dataset=pd.read_csv("BakkenS1prediction.csv",sep=",")
```

In [59]:
```
BakkenS1prediction= raw_dataset.copy()
BakkenS1prediction.head()
```

Out[59]:

	Well	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
0	Upper_Bakken	0.35	3.17	13.57	0.94	39.57	44.87	1.60
1	Upper_Bakken	0.35	3.18	7.41	0.94	45.08	42.29	1.34
2	Upper_Bakken	0.35	3.18	9.14	0.94	48.90	36.24	0.60
3	Upper_Bakken	0.35	3.18	13.33	0.94	63.59	24.07	0.90
4	Lower_Bakken	0.35	3.21	6.58	0.94	38.36	49.23	1.50

In [60]:
```
BakkenS1prediction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Well              8 non-null      object 
 1   Age_BA            8 non-null      float64
 2   Present_Depth_km  8 non-null      float64
 3   TOC_%             8 non-null      float64
 4   Requ_%            8 non-null      float64
 5   Quartz_%          8 non-null      float64
 6   Clay_%            8 non-null      float64
 7   Carbonate_%       8 non-null      float64
dtypes: float64(7), object(1)
memory usage: 640.0+ bytes
```

In [61]:
```
BakkenS1prediction.describe()
```

Out[61]:

	Age_BA	Present_Depth_km	TOC_%	Requ_%	Quartz_%	Clay_%	Carbonate_%
count	8.000000e+00	8.000000	8.000000	8.000000e+00	8.000000	8.000000	8.000000
mean	3.500000e-01	3.197500	10.727500	9.400000e-01	45.281250	42.230000	1.036250
std	5.934392e-17	0.021876	3.313762	2.373757e-16	8.888149	9.706733	0.485149
min	3.500000e-01	3.170000	6.580000	9.400000e-01	38.040000	24.070000	0.150000
25%	3.500000e-01	3.180000	8.295000	9.400000e-01	38.420000	36.532500	0.825000
50%	3.500000e-01	3.195000	10.240000	9.400000e-01	42.325000	43.580000	1.100000
75%	3.500000e-01	3.220000	13.390000	9.400000e-01	49.242500	49.570000	1.380000
max	3.500000e-01	3.220000	15.860000	9.400000e-01	63.590000	53.920000	1.600000

In [62]:
```
X_new = BakkenS1prediction.iloc[:, 1:].values
```

In [63]:
```
X_new
```

Out[63]:
```
array([[ 0.35,  3.17, 13.57,  0.94, 39.57, 44.87,  1.6 ],
       [ 0.35,  3.18,  7.41,  0.94, 45.08, 42.29,  1.34],
       [ 0.35,  3.18,  9.14,  0.94, 48.9 , 36.24,  0.6 ],
       [ 0.35,  3.18, 13.33,  0.94, 63.59, 24.07,  0.9 ],
       [ 0.35,  3.21,  6.58,  0.94, 38.36, 49.23,  1.5 ],
       [ 0.35,  3.22, 11.34,  0.94, 38.04, 53.92,  1.  ],
       [ 0.35,  3.22, 15.86,  0.94, 50.27, 36.63,  1.2 ],
       [ 0.35,  3.22,  8.59,  0.94, 38.44, 50.59,  0.15]])
```

In [64]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_new = sc.fit_transform(X_new)
```

In [65]:
```
X_new
```

Out[65]:
```
array([[ 0.        , -1.34386389,  0.91701212,  0.        , -0.68693531,
         0.29075476,  1.24224513],
       [ 0.        , -0.85518611, -1.07025074,  0.        , -0.02420586,
         0.00660806,  0.66932498],
       [ 0.        , -0.85518611, -0.51213958,  0.        ,  0.43525448,
        -0.65970493, -0.96129391],
       [ 0.        , -0.85518611,  0.8395863 ,  0.        ,  2.20213209,
        -2.00004033, -0.30023219],
       [ 0.        ,  0.61084722, -1.33801505,  0.        , -0.83247118,
         0.77094066,  1.02189123],
       [ 0.        ,  1.099525  ,  0.19759716,  0.        , -0.87096001,
         1.2874709 , -0.07987829],
       [ 0.        ,  1.099525  ,  1.65578354,  0.        ,  0.60003476,
        -0.61675252,  0.36082952],
       [ 0.        ,  1.099525  , -0.68957376,  0.        , -0.82284897,
         0.92072341, -1.95288647]])
```

In [66]:
```
print(model.predict(X_new))
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000020FE7B81310> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args (https://www.tensorflow.org/tutorials/customization/performance#python_or_tensor_args) and https://www.tensorflow.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.
[[3.4653094]
 [1.573999 ]
 [2.7505193]
 [3.391518 ]
 [1.8404518]
 [3.1433775]
 [4.166593 ]
 [2.3330393]]
```

In [ ]: