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]:
```
print(tf.__version__)
```

```
2.3.0
```

In [3]:
```
raw_dataset=pd.read_csv("DadasBV1.csv",sep=",")
```

In [4]:
```
DadasBV1 = raw_dataset.copy()
DadasBV1.head()
```

Out[4]:

	Well	Depth_km	Brittleness	Clay_%	GR	DT
0	ABDULAZIZ-1	2.74	0.27	69.6	125	80
1	ABDULAZIZ-1	2.75	0.49	50.3	124	90
2	ABDULAZIZ-1	2.76	0.47	52.1	121	75
3	ABDULAZIZ-1	2.78	0.56	43.1	85	70
4	ABDULAZIZ-1	2.79	0.23	75.8	123	92

In [5]:
```
DadasBV1.shape
```

Out[5]:
```
(399, 6)
```

In [6]:
```
DadasBV1.describe()
```

Out[6]:

	Depth_km	Brittleness	Clay_%	GR	DT
count	399.000000	399.000000	399.000000	399.000000	399.000000
mean	2.821689	0.574887	39.328571	133.932331	96.225564
std	0.308511	0.171260	17.063916	34.829202	19.658753
min	2.371100	0.230000	2.100000	54.000000	48.000000
25%	2.428000	0.460000	26.300000	113.000000	84.000000
50%	2.870000	0.560000	40.500000	138.000000	90.000000
75%	3.040000	0.690000	51.200000	162.000000	113.000000
max	3.360000	0.970000	75.800000	195.000000	143.000000

In [7]:
```
DadasBV1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 399 entries, 0 to 398
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Well         399 non-null    object
 1   Depth_km     399 non-null    float64
 2   Brittleness  399 non-null    float64
 3   Clay_%       399 non-null    float64
 4   GR           399 non-null    int64
 5   DT           399 non-null    int64
dtypes: float64(3), int64(2), object(1)
memory usage: 18.8+ KB
```

In [8]:
```
corr_matrix =DadasBV1.corr()
```

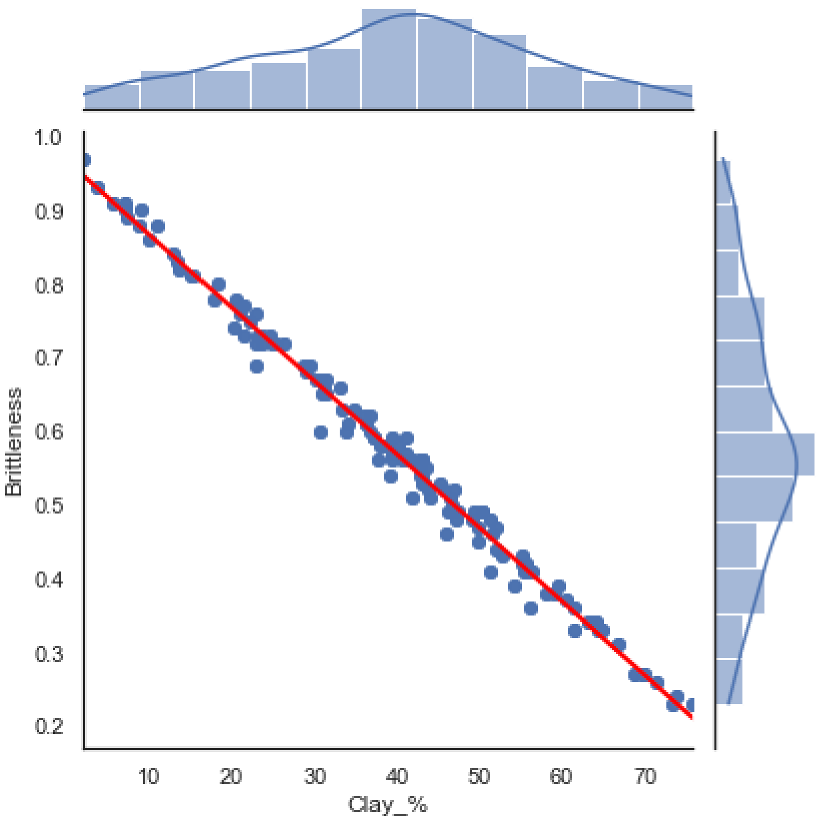
In [9]:
```
corr_matrix["Brittleness"].sort_values(ascending=False)
```

Out[9]:
```
Brittleness    1.000000
Depth_km       0.014462
DT            -0.182549
GR            -0.292045
Clay_%        -0.995075
Name: Brittleness, dtype: float64
```

In [10]:
```
sns.set_theme(style="white")
plt.figure(figsize = (20,5), dpi = (500))
sns.jointplot(x = DadasBV1['Clay_%'], y = DadasBV1['Brittleness'], kind='reg', line_kws={"color": "red"})
```

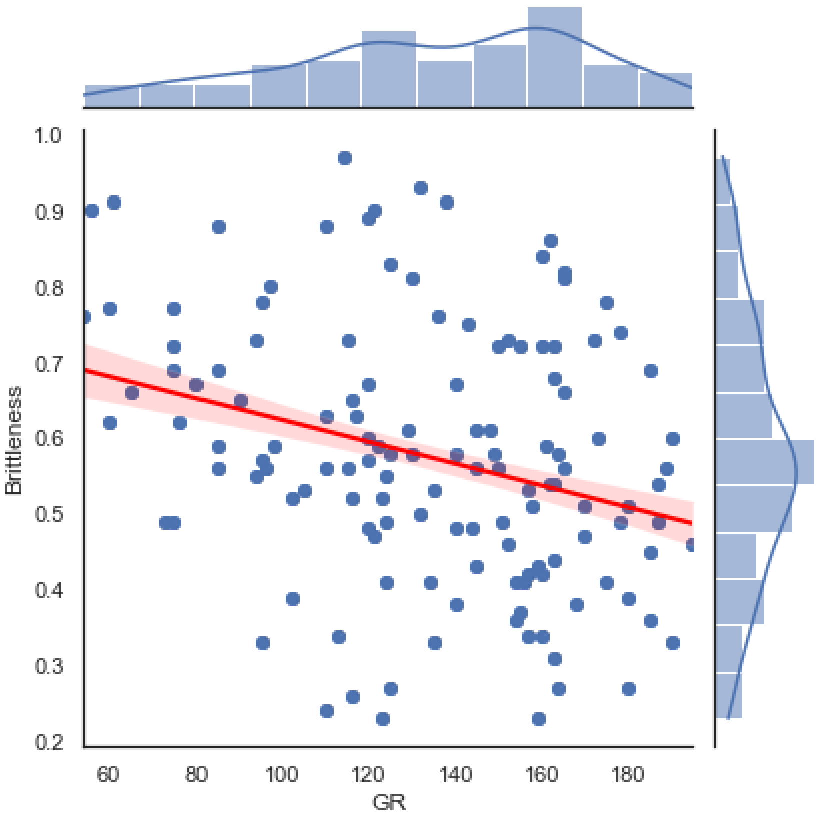
Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x262556eb790>

<Figure size 10000x2500 with 0 Axes>
```



In [11]:
```
sns.set_theme(style="white")
plt.figure(figsize = (20,5), dpi = (500))
sns.jointplot(x = DadasBV1['GR'], y = DadasBV1['Brittleness'], kind='reg', line_kws={"color": "red"})
font_size =80
```

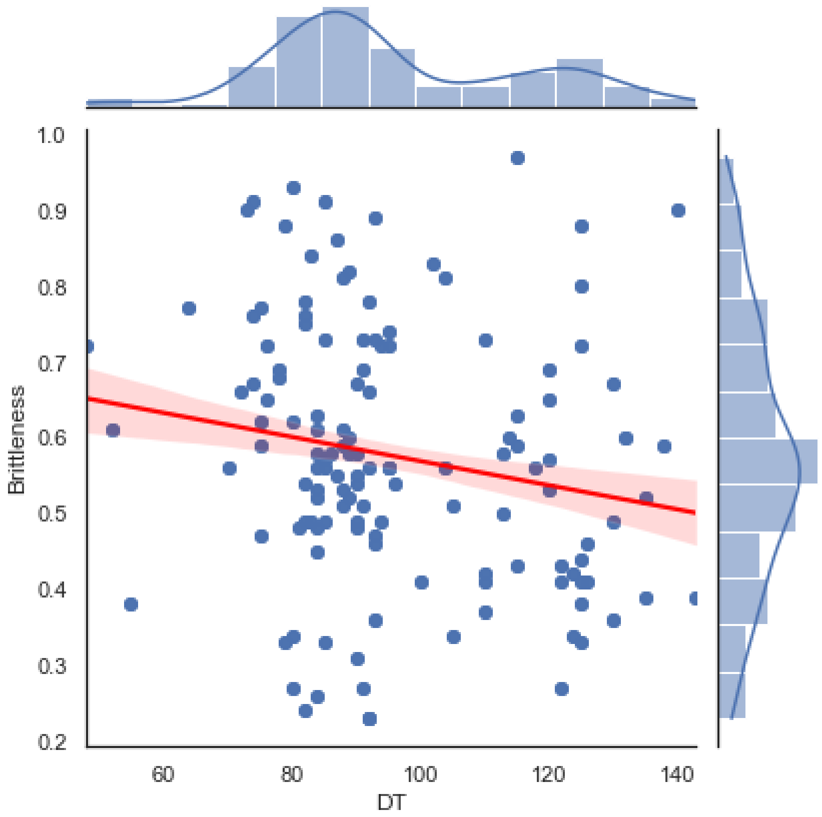
```
<Figure size 10000x2500 with 0 Axes>
```



In [12]:
```
sns.set_theme(style="white")
plt.figure(figsize = (20,5), dpi = (500))
sns.jointplot(x = DadasBV1['DT'], y = DadasBV1['Brittleness'], kind='reg', line_kws={"color": "red"})
```

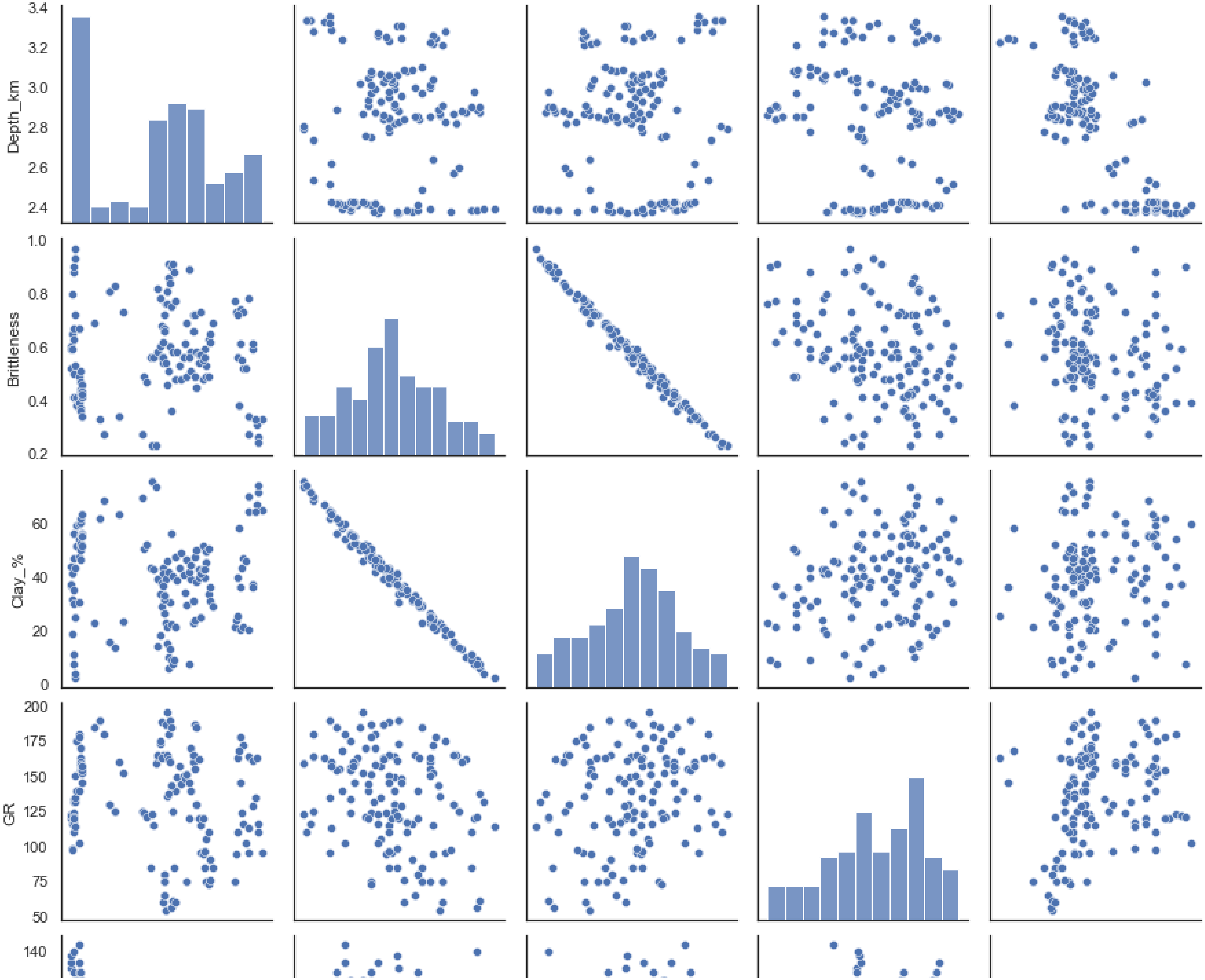
Out[12]:
```
<seaborn.axisgrid.JointGrid at 0x262560f1af0>
```

```
<Figure size 10000x2500 with 0 Axes>
```

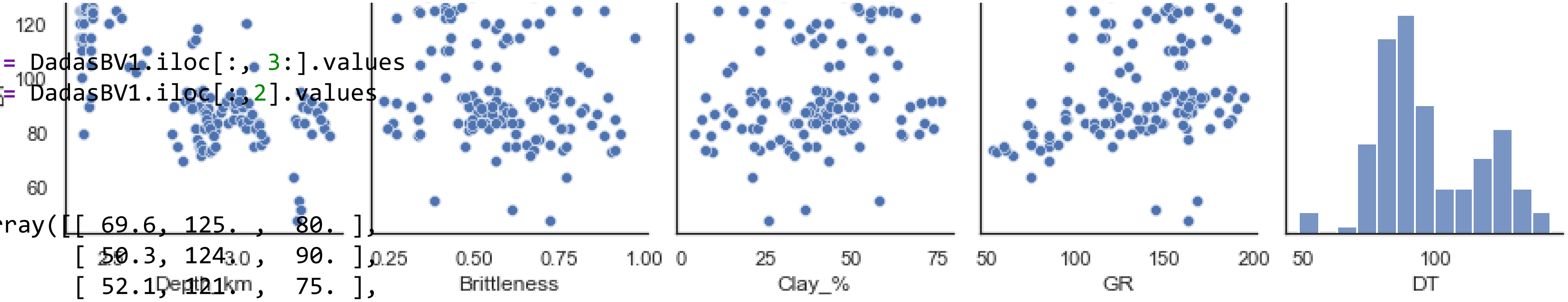


In [13]:
```
sns.pairplot(DadasBV1)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x262561b75b0>
```



Depth_km
Brittleness
Clay_%
GR
3.4
3.2
3.0
2.8
2.6
2.4
1.0
0.8
0.6
0.4
0.2
60
40
20
0
200
175
150
125
100
75
50
140



In [14]:
```python
X = DadasBV1.iloc[:, 3:].values
y = DadasBV1.iloc[:, 2].values
```

In [15]:
```python
X
```

Out[15]:
```
array([[ 69.6, 125. ,  80. ],
       [ 50.3, 124. ,  90. ],
       [ 52.1, 121. ,  75. ],
       ...,
       [ 55.3, 154. , 122. ],
       [ 55.2, 145. , 122. ],
       [ 55.2, 157. , 124. ]])
```

In [16]:
```
y
```

Out[16]:
```
array([0.27, 0.49, 0.47, 0.56, 0.23, 0.56, 0.23, 0.58, 0.6 , 0.56, 0.54,
       0.56, 0.54, 0.46, 0.51, 0.6 , 0.36, 0.69, 0.89, 0.58, 0.72, 0.72,
       0.58, 0.57, 0.73, 0.48, 0.59, 0.53, 0.63, 0.65, 0.69, 0.58, 0.48,
       0.58, 0.53, 0.48, 0.56, 0.61, 0.49, 0.56, 0.51, 0.66, 0.49, 0.45,
       0.54, 0.57, 0.55, 0.56, 0.49, 0.49, 0.62, 0.82, 0.78, 0.68, 0.62,
       0.67, 0.72, 0.66, 0.76, 0.76, 0.81, 0.86, 0.91, 0.84, 0.75, 0.9 ,
       0.91, 0.88, 0.77, 0.69, 0.33, 0.27, 0.81, 0.83, 0.34, 0.73, 0.77,
       0.73, 0.56, 0.74, 0.73, 0.27, 0.59, 0.31, 0.26, 0.24, 0.33, 0.72,
       0.38, 0.61, 0.55, 0.52, 0.52, 0.78, 0.34, 0.61, 0.33, 0.6 , 0.52,
       0.59, 0.65, 0.59, 0.8 , 0.5 , 0.63, 0.41, 0.41, 0.88, 0.9 , 0.67,
       0.97, 0.93, 0.72, 0.53, 0.38, 0.41, 0.39, 0.39, 0.49, 0.67, 0.44,
       0.47, 0.43, 0.42, 0.51, 0.37, 0.36, 0.46, 0.41, 0.34, 0.41, 0.43,
       0.42, 0.27, 0.49, 0.47, 0.56, 0.23, 0.56, 0.23, 0.58, 0.6 , 0.56,
       0.54, 0.56, 0.54, 0.46, 0.51, 0.6 , 0.36, 0.69, 0.89, 0.58, 0.72,
       0.72, 0.58, 0.57, 0.73, 0.48, 0.59, 0.53, 0.63, 0.65, 0.69, 0.58,
       0.48, 0.58, 0.53, 0.48, 0.56, 0.61, 0.49, 0.56, 0.51, 0.66, 0.49,
       0.45, 0.54, 0.57, 0.55, 0.56, 0.49, 0.49, 0.62, 0.82, 0.78, 0.68,
       0.62, 0.67, 0.72, 0.66, 0.76, 0.76, 0.81, 0.86, 0.91, 0.84, 0.75,
       0.9 , 0.91, 0.88, 0.77, 0.69, 0.33, 0.27, 0.81, 0.83, 0.34, 0.73,
       0.77, 0.73, 0.56, 0.74, 0.73, 0.27, 0.59, 0.31, 0.26, 0.24, 0.33,
       0.72, 0.38, 0.61, 0.55, 0.52, 0.52, 0.78, 0.34, 0.61, 0.33, 0.6 ,
       0.52, 0.59, 0.65, 0.59, 0.8 , 0.5 , 0.63, 0.41, 0.41, 0.88, 0.9 ,
       0.67, 0.97, 0.93, 0.72, 0.53, 0.38, 0.41, 0.39, 0.39, 0.49, 0.67,
       0.44, 0.47, 0.43, 0.42, 0.51, 0.37, 0.36, 0.46, 0.41, 0.34, 0.41,
       0.43, 0.42, 0.27, 0.49, 0.47, 0.56, 0.23, 0.56, 0.23, 0.58, 0.6 ,
       0.56, 0.54, 0.56, 0.54, 0.46, 0.51, 0.6 , 0.36, 0.69, 0.89, 0.58,
       0.72, 0.72, 0.58, 0.57, 0.73, 0.48, 0.59, 0.53, 0.63, 0.65, 0.69,
       0.58, 0.48, 0.58, 0.53, 0.48, 0.56, 0.61, 0.49, 0.56, 0.51, 0.66,
       0.49, 0.45, 0.54, 0.57, 0.55, 0.56, 0.49, 0.49, 0.62, 0.82, 0.78,
       0.68, 0.62, 0.67, 0.72, 0.66, 0.76, 0.76, 0.81, 0.86, 0.91, 0.84,
       0.75, 0.9 , 0.91, 0.88, 0.77, 0.69, 0.33, 0.27, 0.81, 0.83, 0.34,
       0.73, 0.77, 0.73, 0.56, 0.74, 0.73, 0.27, 0.59, 0.31, 0.26, 0.24,
       0.33, 0.72, 0.38, 0.61, 0.55, 0.52, 0.52, 0.78, 0.34, 0.61, 0.33,
       0.6 , 0.52, 0.59, 0.65, 0.59, 0.8 , 0.5 , 0.63, 0.41, 0.41, 0.88,
       0.9 , 0.67, 0.97, 0.93, 0.72, 0.53, 0.38, 0.41, 0.39, 0.39, 0.49,
       0.67, 0.44, 0.47, 0.43, 0.42, 0.51, 0.37, 0.36, 0.46, 0.41, 0.34,
       0.41, 0.43, 0.42])
```

In [17]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [18]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [19]:
```
X_train
```

Out[19]:
```
array([[ 0.09862093, -1.10849019, -0.51514649],
       [-0.04269373,  0.42253484, -0.56435697],
       [ 0.6226628 ,  1.4432182 , -0.56435697],
       [-0.4607496 , -0.51309157,  1.2072203 ],
       [ 0.20460693,  0.79111494, -0.66277793],
       [-0.49019015, -1.25025177, -0.95804081],
       [-0.61972859, -1.39201335, -0.85961985],
       [-0.31354683,  0.39418253, -0.56435697],
       [-1.97399409,  0.11065937, -0.51514649],
       [-0.97301524,  0.73441031, -0.07225218],
       [-1.52649766, -0.25792073,  0.32143166],
       [-2.19185419, -0.5697962 ,  0.9611679 ],
       [-0.14867972, -0.39968231,  1.79774606],
       [-0.97301524,  1.4432182 ,  1.2072203 ],
       [-0.60795237,  0.81946726, -0.85961985],
       [-0.95535091,  0.50759179,  0.7151155 ],
       [ 2.00636885,  0.706058  , -0.17067314],
       [ 0.22227126, -1.39201335, -1.25330369],
       [-0.86702925, -0.54144388, -0.21988362],
```

In [20]:
```
X_test
```

Out[20]:
```
array([[ 0.87585157,  1.30145662,  1.9453775 ],
       [ 0.45190759, -0.51309157, -0.56435697],
       [-0.26644194, -0.48473925,  0.9611679 ],
       [ 0.26348804, -0.31462536,  1.9453775 ],
       [-0.5137426 ,  1.58497978, -0.31830457],
       [-0.18400839, -0.14451147, -0.36751505],
       [-2.09764442, -0.05945452, -0.76119889],
       [-1.41462356, -0.11615915,  0.41985262],
       [ 0.00441116, -1.39201335, -1.00725129],
       [-0.97301524,  0.73441031, -0.07225218],
       [ 0.20460693, -1.13684251, -0.41672553],
       [ 0.43424325, -0.05945452,  0.86274694],
       [ 0.97594945,  0.62100105,  1.4532727 ],
       [ 0.58733414, -1.6755365 , -0.66277793],
       [ 1.41166966,  0.73441031,  0.4690631 ],
       [-0.57851182, -1.6755365 , -0.21988362],
       [ 1.41166966,  0.73441031,  0.4690631 ],
       [-0.48430204,  0.87617189, -0.17067314],
       [ 0.94062079,  0.56429642,  1.30564126],
```

In [21]:
```python
from tensorflow.keras.layers import Input, Dense, Activation,Dropout
from tensorflow.keras.models import Model
```

In [22]:
```python
input_layer = Input(shape=(X.shape[1],))
dense_layer_1 = Dense(512, activation='relu')(input_layer)
dense_layer_2 = Dense(256, activation='relu')(dense_layer_1)
dense_layer_3 = Dense(128, activation='relu')(dense_layer_2)
dense_layer_4 = Dense(64, activation='relu')(dense_layer_3)
dense_layer_5 = Dense(64,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 [23]:
```
my_model = model
```

In [24]:
```
my_model.summary()
```

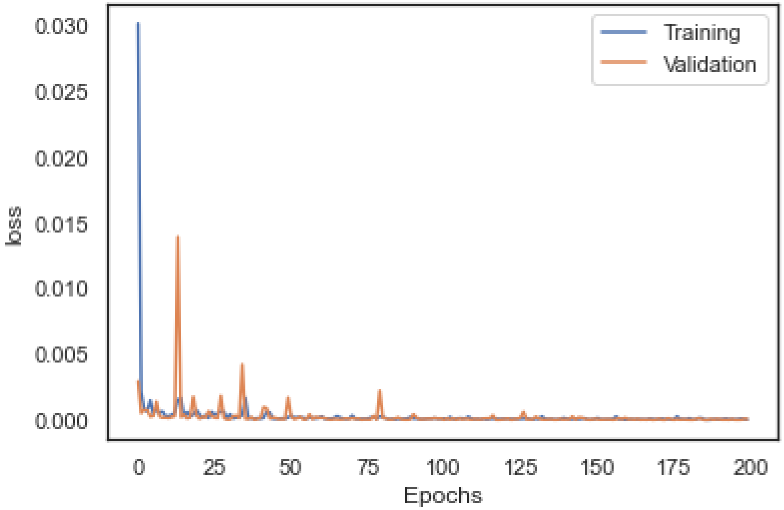
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 3)]               0
_________________________________________________________________
dense (Dense)                (None, 512)               2048
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 174,593
Trainable params: 174,593
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
history = model.fit(X_train, y_train, batch_size=1, epochs=200, verbose=1, validation_split=0.2)
```

```
ss: 2.1792e-04 - val_mean_squared_error: 2.1792e-04
Epoch 24/200
239/239 [==============================] - 1s 6ms/step - loss: 1.8133e-04 - mean_squared_error: 1.8133e-04 - val_lo
ss: 6.9417e-04 - val_mean_squared_error: 6.9417e-04
Epoch 25/200
239/239 [==============================] - 1s 6ms/step - loss: 6.5375e-04 - mean_squared_error: 6.5375e-04 - val_lo
ss: 3.1490e-04 - val_mean_squared_error: 3.1490e-04
Epoch 26/200
239/239 [==============================] - 2s 7ms/step - loss: 4.0434e-04 - mean_squared_error: 4.0434e-04 - val_lo
ss: 2.0707e-04 - val_mean_squared_error: 2.0707e-04
Epoch 27/200
239/239 [==============================] - 2s 8ms/step - loss: 5.3846e-04 - mean_squared_error: 5.3846e-04 - val_lo
ss: 1.8300e-04 - val_mean_squared_error: 1.8300e-04
Epoch 28/200
239/239 [==============================] - 2s 8ms/step - loss: 6.8490e-04 - mean_squared_error: 6.8490e-04 - val_lo
ss: 0.0019 - val_mean_squared_error: 0.0019 - ETA: 1s - l
Epoch 29/200
239/239 [==============================] - 2s 7ms/step - loss: 6.5797e-04 - mean_squared_error: 6.5797e-04 - val_lo
ss: 1.7114e-04 - val_mean_squared_error: 1.7114e-04
Epoch 30/200
```

In [26]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.legend(["Training","Validation"])
plt.figure(figsize = (20,5), dpi = (500))
```

Out[26]:
```
<Figure size 10000x2500 with 0 Axes>
```




```
<Figure size 10000x2500 with 0 Axes>
```

In [44]:
```
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)))
```

```
0.008253118868915506
0.008617677228441815
```

In [45]:
```
score = model.evaluate(X_test, y_test, verbose=1)

print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

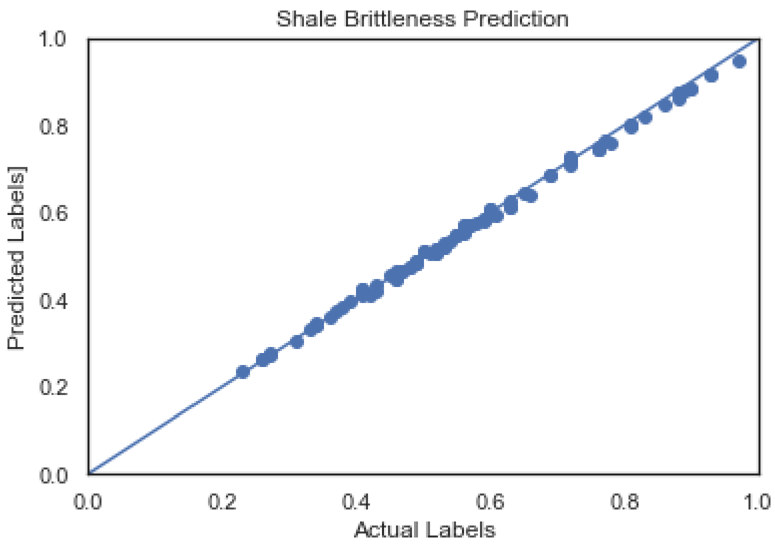
```
4/4 [==============================] - 0s 6ms/step - loss: 7.4264e-05 - mean_squared_error: 7.4264e-05
Test Score: 7.42642005207017e-05
Test Accuracy: 7.42642005207017e-05
```

In [62]:
```
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:  [[0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
Actual labels   :  [0.39 0.52 0.63 0.52 0.6  0.61 0.93 0.81 0.59 0.72]
```

In [48]:
```
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels]')
plt.title('Shale Brittleness Prediction')
lims = [0, 1]
plt.xlim(lims)
plt.ylim(lims)
_ = plt.plot(lims, lims)
plt.figure(figsize = (30,5), dpi = (500))
```

Out[48]:
```
<Figure size 15000x2500 with 0 Axes>
```




```
<Figure size 15000x2500 with 0 Axes>
```

In [49]:
```
my_model = model
```

In [50]:
```
my_model.save('./saved_models/my_tf_model')
```

```
INFO:tensorflow:Assets written to: ./saved_models/my_tf_model\assets
```

In [51]:
```
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, 3)]               0
_________________________________________________________________
dense (Dense)                (None, 512)               2048
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 174,593
Trainable params: 174,593
Non-trainable params: 0
_________________________________________________________________
```

In [52]:
```
from tensorflow.keras.models import save_model, load_model
import pandas as pd
```

In [53]:
```
model = load_model('./saved_models/my_tf_model',
        custom_objects=None,
    compile=True)
```

In [54]:
```
raw_dataset=pd.read_csv("DadasBPV1.csv",sep=",")
```

In [55]:
```
DadasBPV1 = raw_dataset.copy()
DadasBPV1.head()
```

Out[55]:

	Well	Depth_km	Clay_%	GR	DT
0	Akcay_1	3.675	54.07	70	65
1	Akcay_1	3.680	36.39	75	70
2	Akcay_1	3.685	74.07	90	60
3	Akcay_1	3.690	33.52	160	60
4	Akcay_1	3.695	51.66	190	85

In [56]:
```
X_new =DadasBPV1.iloc[:, 2:].values
```

In [57]:
```
X_new
```

Out[57]:
```
array([[ 54.07,  70.  ,  65.  ],
       [ 36.39,  75.  ,  70.  ],
       [ 74.07,  90.  ,  60.  ],
       [ 33.52, 160.  ,  60.  ],
       [ 51.66, 190.  ,  85.  ],
       [ 40.48, 195.  ,  91.  ],
       [ 37.89, 165.  ,  90.  ],
       [ 22.49,  95.  ,  84.  ],
       [ 25.41,  75.  ,  88.  ],
       [ 36.35, 160.  ,  81.  ],
       [ 42.5 ,  95.  ,  82.  ],
       [ 46.14,  85.  ,  80.  ],
       [ 42.47,  90.  ,  75.  ],
       [ 38.58,  70.  ,  80.  ],
       [ 39.36,  78.  ,  80.  ],
       [ 42.29,  45.  ,  60.  ],
       [ 41.84,  85.  ,  65.  ],
       [ 24.66,  80.  ,  66.  ],
       [ 31.99,  82.  ,  59.  ],
       [  5.21,  84.  ,  70.  ],
       [  2.93,  81.  ,  80.  ],
       [ 34.18,  95.  ,  60.  ],
       [ 35.62, 160.  ,  62.  ],
       [ 43.1 , 140.  ,  80.  ],
       [ 53.75, 120.  ,  75.  ],
       [ 60.78, 125.  ,  85.  ],
       [ 37.18, 122.  ,  70.  ],
       [ 34.9 , 124.  ,  95.  ],
       [ 62.4 , 120.  ,  95.  ],
       [ 42.75, 100.  , 100.  ],
       [ 54.25, 126.  ,  75.  ],
       [ 50.52, 128.  ,  80.  ]])
```

In [58]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_new = sc.fit_transform(X_new)
```

In [59]:
```
X_new
```

Out[59]:
```
array([[ 0.98231945, -1.07869874, -1.0038023 ],
       [-0.25129812, -0.94279969, -0.56736651],
       [ 2.37781444, -0.53510253, -1.44023808],
       [-0.45155165,  1.36748423, -1.44023808],
       [ 0.8141623 ,  2.18287856,  0.74194083],
       [ 0.0340806 ,  2.31877761,  1.26566376],
       [-0.146636  ,  1.50338329,  1.17837661],
       [-1.22116714, -0.39920347,  0.65465367],
       [-1.01742487, -0.94279969,  1.0038023 ],
       [-0.25408911,  1.36748423,  0.3927922 ],
       [ 0.1750256 , -0.39920347,  0.48007936],
       [ 0.42900569, -0.67100158,  0.30550505],
       [ 0.17293236, -0.53510253, -0.13093073],
       [-0.09849142, -1.07869874,  0.30550505],
       [-0.04406712, -0.86126026,  0.30550505],
       [ 0.1603729 , -1.75819402, -1.44023808],
       [ 0.12897426, -0.67100158, -1.0038023 ],
       [-1.06975593, -0.80690063, -0.91651514],
       [-0.55830702, -0.75254101, -1.52752523],
       [-2.42687481, -0.69818139, -0.56736651],
       [-2.58596124, -0.77972082,  0.30550505],
       [-0.40550032, -0.39920347, -1.44023808],
       [-0.30502468,  1.36748423, -1.26566376],
       [ 0.21689045,  0.82388802,  0.30550505],
       [ 0.95999153,  0.2802918 , -0.13093073],
       [ 1.45050802,  0.41619085,  0.74194083],
       [-0.19617607,  0.33465142, -0.56736651],
       [-0.3552625 ,  0.38901104,  1.61481239],
       [ 1.56354311,  0.2802918 ,  1.61481239],
       [ 0.19246929, -0.26330442,  2.05124817],
       [ 0.99487891,  0.44337066, -0.13093073],
       [ 0.73461909,  0.49773029,  0.30550505]])
```

In [60]:
```
print(model.predict(X_new))
```

```
[[0.43101504]
 [0.61161435]
 [0.23568521]
 [0.602528  ]
 [0.36327404]
 [0.5051749 ]
 [0.5645151 ]
 [0.77834404]
 [0.7567872 ]
 [0.5853956 ]
 [0.55522275]
 [0.50714743]
 [0.5570553 ]
 [0.5998928 ]
 [0.5859057 ]
 [0.57006675]
 [0.551545  ]
 [0.74878705]
 [0.66768813]
```

In [ ]:

In [ ]: