

```
In [1]: import lasio
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsRegressor # for KNN regression

import matplotlib.pyplot as plt # for data visualization
import plotly.express as px # for data visualization
```

```
In [2]: las = lasio.read("ColvilleUnit1.LAS")
well = las.df()
df = well.dropna(how="any")
print(df.shape)
```

```
(6055, 8)
```

```
In [3]: df.isnull().sum()
```

```
Out[3]: CALI      0
GR          0
ILD         0
ILM         0
LL8         0
RHOB        0
SP          0
SWL         0
dtype: int64
```

```
In [4]: from sklearn.model_selection import train_test_split
train , test = train_test_split(df, test_size = 0.3)

x_train = train.drop('GR', axis=1)
y_train = train['GR']

x_test = test.drop('GR', axis = 1)
y_test = test['GR']
```

```
In [5]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)
```

```
In [6]: #import required packages
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:

```
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

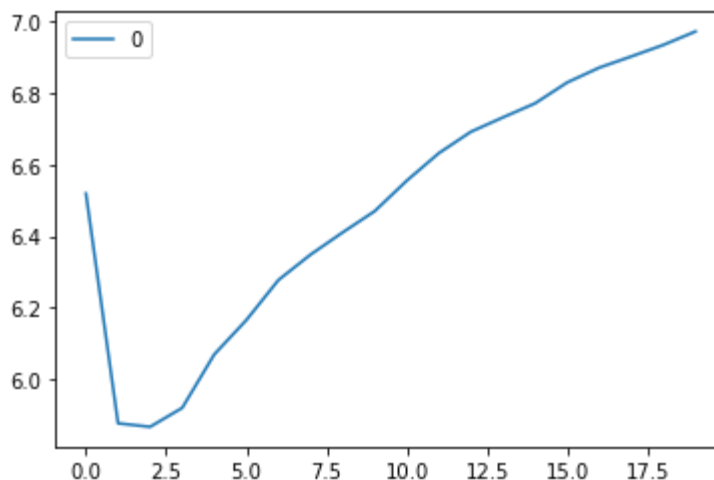
    model.fit(x_train, y_train) #fit the model
    pred=model.predict(x_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ', K , 'is:', error)
```

```
RMSE value for k= 1 is: 6.519317827135389
RMSE value for k= 2 is: 5.8759443369196305
RMSE value for k= 3 is: 5.866107225811023
RMSE value for k= 4 is: 5.919093694756719
RMSE value for k= 5 is: 6.0688633820424265
RMSE value for k= 6 is: 6.165032298710593
RMSE value for k= 7 is: 6.277147871293805
RMSE value for k= 8 is: 6.348198430960938
RMSE value for k= 9 is: 6.41025601225977
RMSE value for k= 10 is: 6.4698670596886325
RMSE value for k= 11 is: 6.555274682804502
RMSE value for k= 12 is: 6.632277999174153
RMSE value for k= 13 is: 6.692191057158776
RMSE value for k= 14 is: 6.732478875281051
RMSE value for k= 15 is: 6.77174368502252
RMSE value for k= 16 is: 6.830192652040286
RMSE value for k= 17 is: 6.871645141579368
RMSE value for k= 18 is: 6.903040291319884
RMSE value for k= 19 is: 6.935320445199133
RMSE value for k= 20 is: 6.972919487527774
```

In [8]:

```
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```

Out[8]: &lt;AxesSubplot:&gt;



```
In [9]: from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}

knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(x_train,y_train)
model.best_params_
```

```
Out[9]: {'n_neighbors': 2}
```

```
In [10]: modelR = KNeighborsRegressor(n_neighbors=2, #default=2
                                     weights='uniform', #{'uniform', 'distance'} or callable,
                                     algorithm='auto', #{'auto', 'ball_tree', 'kd_tree', 'br
                                     #leaf_size=30, #default=30, Leaf size passed to BallTree
                                     #p=2, #default=2, Power parameter for the Minkowski metr
                                     #metric='minkowski', #default='minkowski', with p=2 is e
                                     metric_params=None, #dict, default=None, Additional keyv
                                     n_jobs=-1 #default=None, The number of parallel jobs to
                                     )
```

```
In [11]: reg = modelR.fit(x_train, y_train)
```

```
In [12]: # Predict on training data

pred_values_tr = modelR.predict(x_train)

# Predict on a test data

pred_values_te = modelR.predict(x_test)
```

```
In [13]: # Basic info about the model
print("")
print('***** KNN Regression *****')

print("")
scoreR_te = modelR.score(x_test, y_test)
print('Test Accuracy Score: ', scoreR_te)
scoreR_tr = modelR.score(x_train, y_train)
print('Training Accuracy Score: ', scoreR_tr)

print('-----')
```

```
***** KNN Regression *****
```

```
Test Accuracy Score: 0.8754898272808345
Training Accuracy Score: 0.9696118262736437
```

```
-----
```

In [14]:

```
# Create a copy of each dataframe before modifying
df_train_new=train.copy()
df_test_new=test.copy()

# ----- Training DataFrame -----
# Attach predicted class labels and values

df_train_new['Predicted GR']=pred_values_tr

# ----- Test DataFrame -----
# Attach predicted class labels and values

df_test_new['Predicted GR']=pred_values_te

# ----- Combined DataFrame -----
# Combine training and testing dataframes back into one
df_new=pd.concat([df_train_new, df_test_new], ignore_index=False, axis=0, sort=False)
df_new
```

Out[14]:

	CALI	GR	ILD	ILM	LL8	RHOB	SP	SWL	Predicted GR
DEPTH									
2450.0	8.371	72.871	14.984	16.373	19.698	2.553	54.188	24.116	73.1925
1989.0	8.290	59.409	32.702	36.681	37.297	2.634	55.445	11.519	59.5205
3006.5	8.464	67.354	33.907	36.167	35.683	2.659	60.417	11.144	60.2440
3311.0	8.042	32.544	45.822	50.419	38.891	2.453	66.370	13.173	31.7090
3373.0	8.219	49.611	25.729	26.421	26.948	2.465	56.610	16.550	52.8445
...	...	...	...	...	...	...	...	...	...
2637.5	8.350	65.750	26.711	31.032	31.199	2.619	58.796	11.532	63.7450
4116.5	8.244	66.181	30.151	39.143	40.813	2.605	82.725	13.654	67.2790
1397.5	8.298	73.459	18.471	19.014	17.840	2.566	43.572	16.760	69.6450
1395.0	8.328	104.638	14.583	14.929	13.505	2.520	43.517	21.343	73.0400
1262.5	8.095	61.523	41.517	46.780	41.335	2.609	34.887	11.307	61.2685

6055 rows × 9 columns

In [15]:

```
data= df_new.sort_index(ascending=True)
data
```

Out[15]:

	CALI	GR	ILD	ILM	LL8	RHOB	SP	SWL	Predicted GR
DEPTH									

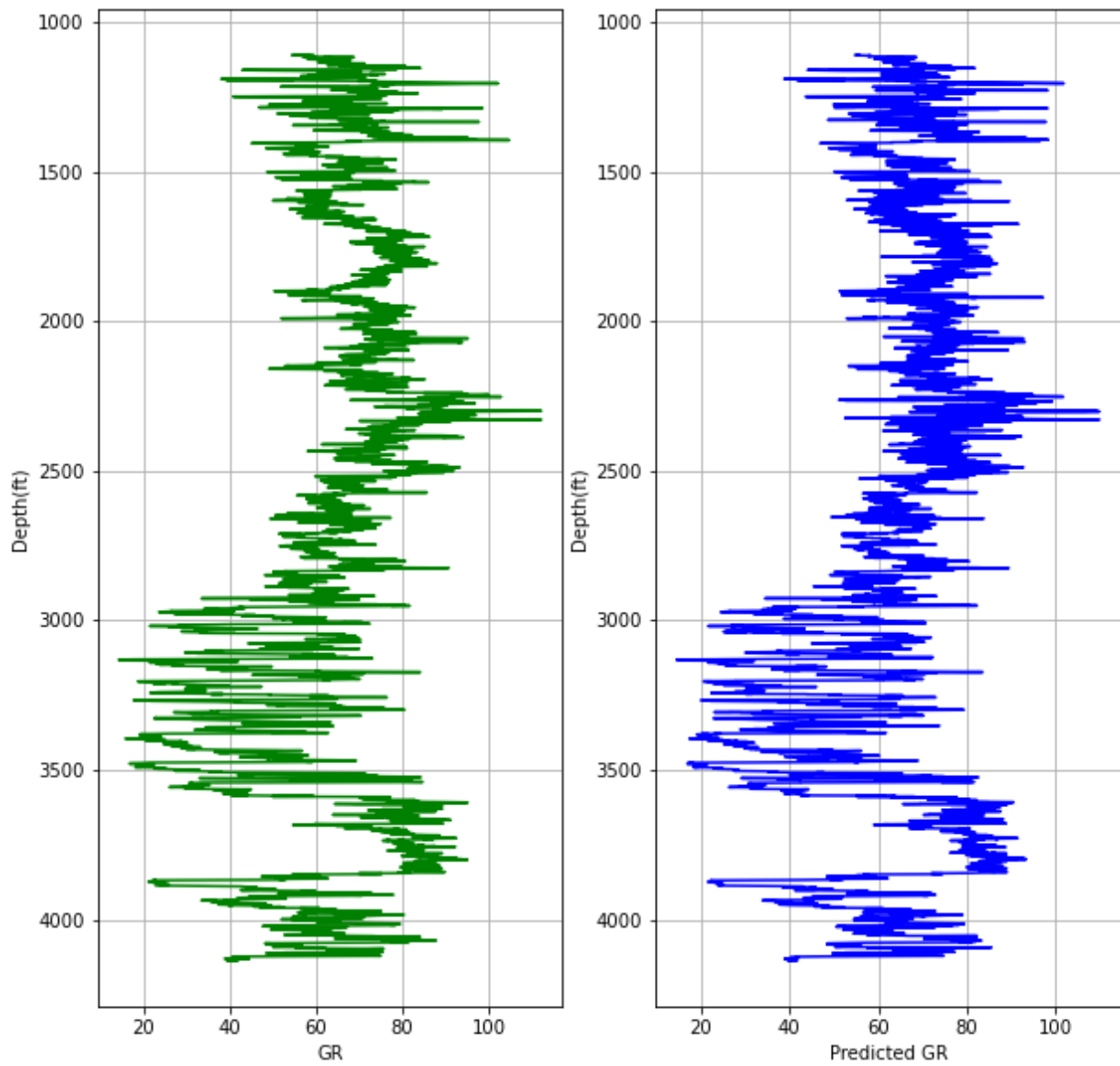
	CALI	GR	ILD	ILM	LL8	RHOB	SP	SWL	Predicted GR
DEPTH									
1110.0	13.907	59.206	2.996	2.572	3.121	1.852	28.073	29.676	55.4650
1110.5	14.354	56.543	2.893	2.574	3.047	1.802	28.022	31.547	55.4650
1111.0	14.037	54.387	2.952	2.613	2.992	1.748	28.033	33.854	54.7025
1111.5	13.877	55.018	3.013	2.675	2.973	1.709	28.078	36.162	55.4745
1112.0	14.156	55.931	3.082	2.769	2.954	1.712	28.126	38.469	55.4745
...	...	...	...	...	...	...	...	...	...
4135.0	8.123	39.606	22.608	28.664	27.949	2.481	64.309	11.841	40.4935
4135.5	8.127	39.293	22.738	28.939	28.138	2.470	64.294	12.295	39.5670
4136.0	8.131	39.841	22.871	29.216	28.303	2.462	64.258	12.579	39.5670
4136.5	8.129	40.555	22.971	29.440	28.409	2.483	64.211	12.332	39.9240
4137.0	8.131	41.694	23.066	29.578	28.506	2.461	64.169	11.887	40.4935

In [17]:

```

def plotter():
    f, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,10))
    logs = ['GR', 'Predicted GR']
    colors = ['green', 'blue']
    for i, log, color in zip(range(2), logs, colors):
        ax[i].plot(data[log], data.index, color=color)
        ax[i].invert_yaxis()
        ax[i].set_xlabel(log)
        ax[i].set_ylabel("Depth(ft)")
        ax[i].grid()
    plotter()

```



In [ ]:

In [ ]:

In [ ]: