UNIVERSITY OF LOUISIANA Lafayette

GROUP PROJECT REPORT

Machine Learning Applications (INFX 598)

Project title: Identifying the Most Effective Fracking Zones using ML and Deep Learning

Project category: ML and Oil & Gas

Group members: Silas Adeoluwa Samuel (C00440966)
Shamsul Hoque (C00448470)
Jamal Ahmadov (C00423516)

Submission date: November 20, 2020

Contents

Project Summary

This project attempts to estimate one of the most important rock physical properties, a fracability index, by exploring numerous machine learning algorithms, where established laboratory techniques would not be cost-effective. The fracability index of an unconventional hydrocarbon reservoir was predicted using models such as Linear Regression, Support Vector Regression, Lasso Regression, Ridge Regression, Elastic Net, Decision Trees, and Random Forests. The shortlisted models were fine-tuned by using a Grid Search method with different sets of hyperparameters. Consequently, an error analysis was performed on each model using the Root Means Square Error (RMSE) to determine the best predictive model. The model was tested on the new data to determine if it generalizes well. The purpose of this study is to establish a machine learning technique that accurately predicts intrinsic rock properties for newly drilled oil and gas wells.

Statement of Problem & Big Picture View

Hydraulic fracturing is one of the most important measures for developing low-porosity and low-permeability oil and gas reservoirs. Determining the most effective fracturing zones without physically tampering with the formation is still considered a tough task. Fracability is a key parameter that has been used to evaluate whether the reservoir can be easily fractured. In recent years, there have been many reports on fracability evaluation. However, the majority of the techniques to estimate fracability include parameters that should be obtained from the experimental tests in the laboratory. It is a time-consuming and costly procedure and thus, an alternative solution is required. In this project, we propose to utilize easily available conventional well logs to describe fracability using machine learning techniques.

Originally, rock brittleness was adopted to evaluate fracability. However, it has been proved that some reservoirs with high brittleness are not fractured easily. This means that brittleness alone is not enough to describe fracability. Fracture toughness is also an important factor affecting the level of fracability because a higher fracture toughness indicates the rock can better resist fracture initiation and propagation. Another factor to be considered is the minimum horizontal stress of the formation. The lower the stress, the smaller is the fracture-closure stress, resulting in easier fracture propagation and higher fracture conductivity. As a result, our fracability index calculation will be based on these 3 properties. Since fracability has a direct relationship with brittleness and an inverse relationship with fracture toughness and minimum horizontal stress, the equation to calculate fracability index is defined as below.

$$FI = \frac{BI}{(K_{IC} \times minimum\ horizontal\ stress)}$$

where all 3 parameters will be calculated using empirical correlations.

BI is a Brittleness Index and estimated using the equation below:

$$BI = \frac{E_n + V_n}{2} \times 100\ \%$$

$$E_n = \frac{E - E_{min}}{E_{max} - E_{min}}$$

$$\nu_n = \frac{\nu - \nu_{min}}{\nu_{max} - \nu_{min}}$$

$$E_d = \rho V_s^2 \left[\frac{3 V_p^2 - 4 V_s^2}{V_p^2 - V_s^2} \right]$$

$$\nu = \frac{V_p^2 - 2V_s^2}{2 (V_p^2 - V_s^2)}$$

$K_{IC}$is a fracture toughness and can be calculated using Young's modulus.

$$K_{IC} = 3.672 \times 10^{-6} E_d + 0.451$$

Minimum horizontal stress can be estimated by Eaton's equation.

$$\sigma_{min} - P_p = \frac{v}{1 - v}(\sigma_v - P_p)$$

As it is seen the target values of the fracability index will be estimated using empirical equations, whereas we aim to predict fracability directly from well logs without a need to know all these parameters. For a long time, the petroleum industry has been seeking a solution to this problem. As machine learning models can understand complex problems, this study can pave the way to produce long-awaited solutions.

Data Acquisition

The Bakken formation is about 200,000 square miles (520,000 km2) underlying parts of Montana, North Dakota, Saskatchewan, and Manitoba. We collected data from 60 wells that are located in the main producing counties: Burke, Mountrail, Williams, Mckenzie, Billings, and Dunn.

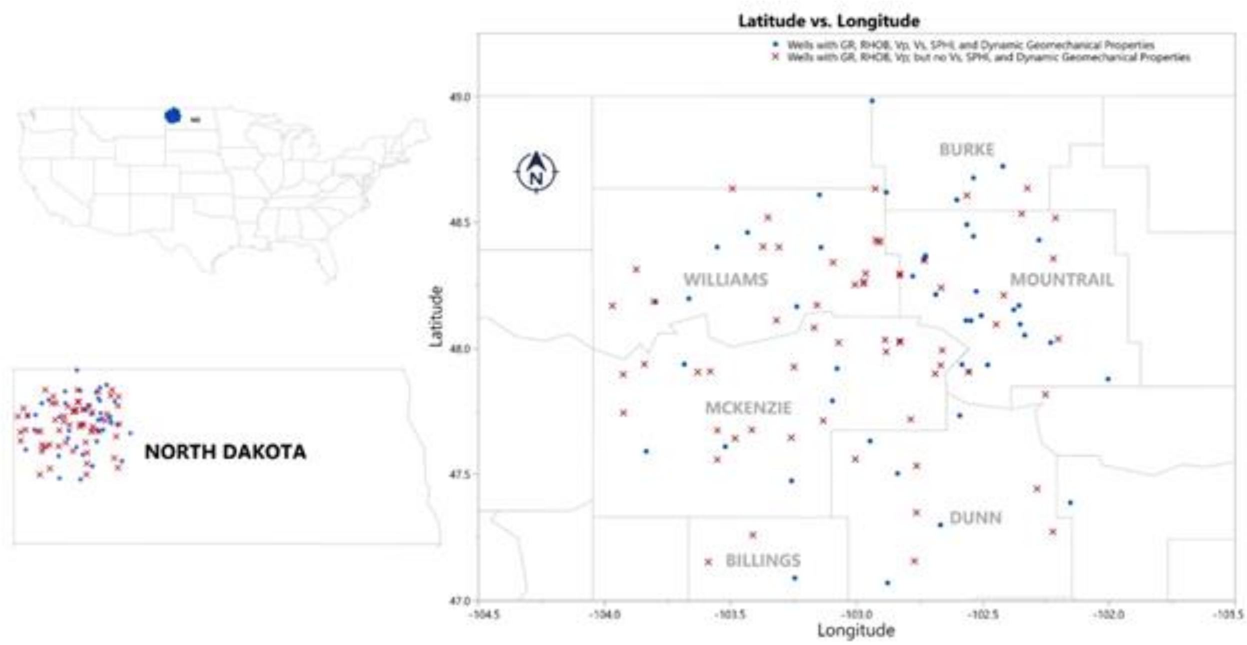


Figure 1. County map of North Dakota.

Density, gamma-ray, resistivity, porosity, and sonic log data from the North Dakota Industrial Commission Oil and Gas Division website were collected. The aim was to analyze the Upper Bakken formation from North Dakota. The Upper and Lower sections of Bakken Formation are easily distinguishable by significantly high Gamma Ray log values, which result from the high organic matter content contained within them required for hydrocarbon generation. Hence, to effectively select the target formation (Upper Bakken), depth and Gamma-Ray value parameters were set to restrict our selection of training data to the target formation. The minimum Gamma Ray log value of the Upper Bakken Formation is approximately 150 API, while the estimated depth of the target formation in each well was obtained from well core sample data and previous well log correlations carried out by Oil and Gas companies. Hence, in the Python code, we have restricted our source of training data to specific start and stop depths, as well as Gamma Ray values equal to or above 150 API.

Jupyter Notebook was used to prepare the data, build, and run promising models. The well log data are ‘las’ files which were read using the ‘lasio’ python package. Creating a data frame makes handling the data a lot easier. Scikit-learn libraries come in handy for various ‘python’ related tasks.

```
In [5]: def logs(las_file):

            df = las_file.df()
            Dens=pd.DataFrame(df[Dens_name])
            GR=pd.DataFrame(df[GR_name])
            Res=pd.DataFrame(df[Res_name])
            Por_dens=pd.DataFrame(df[Por_dens_name])
            Por_neut=pd.DataFrame(df[Por_neut_name])
            Vp=pd.DataFrame(0.3048*10**6/(df[Vp_name]))
            Vs=pd.DataFrame(0.3048*10**6/(df[Vs_name]))
            Logs_comb=pd.concat([Dens,GR,Res,Por_dens,Por_neut,Vp,Vs],axis=1)
            return Logs_comb
```

```
In [9]: def bakken(df,form_top):
            df_bakken=df[form_top:]
            Vel_ub=df_bakken[(df_bakken[GR_name]>=150)]
            return Vel_ub
```

```
In [14]: def frac_index(Vel_ub):

             YM=10**3*Vel_ub[Dens_name]*Vel_ub[Vs_name]**2*(3*Vel_ub[Vp_name]**2-4*Vel_ub[Vs_name]**2)/(Vel_ub[Vp_name]**2
                                                                                                     -Vel_ub[Vs_name]**2)  #Pa
             YM=YM*10**-9 #GPa
             PR=(Vel_ub[Vp_name]**2-2*Vel_ub[Vs_name]**2)/(2*Vel_ub[Vp_name]**2-2*Vel_ub[Vs_name]**2)
             YM_norm=((YM-YM.min())/(YM.max()-YM.min()))
             PR_norm=((PR-PR.min())/(PR.max()-PR.min()))
             BI=100*(YM_norm+PR_norm)/2 #Brittleness index

             Frac_toughness=0.003672*YM+0.45034 #Fracture toughness from Chen et al. (1997)

             overburden_st=1.067 #psi/ft
             pore_pr=0.65 #psi/ft
             Minhor_stress_psi=((PR/(1-PR))*(overburden_st-pore_pr)+pore_pr)*PR.index #psi
             Minhor_stress=Minhor_stress_psi*0.00689476 #MPa

             FI=BI/(Frac_toughness*Minhor_stress)

             df=pd.concat([YM,PR,BI,Frac_toughness,Minhor_stress,FI],axis=1,keys=["YM","PR","BI","Fracture toughness",
                                                                                  "Minimum horizontal stress","FI"])
             return df
```

Finally, we imported the raw data, identified the Upper Bakken interval, calculated the required parameters for fracability index estimation (i.e. target variable), and exported the data of each well in .xlsx format.

In [15]:
```
BI=frac_index(Vel_ub)
final_table=pd.concat([Vel_ub,BI],axis=1)
final_table.head()
```

Out[15]:

DEPT	RHOZ	GR_EDTC	AHORT	DPHZ	NPHI	DTCO	DTSM	YM	PR	BI	Fracture toughness	Minimum horizontal stress	FI
9995.0	2.5040	206.1783	42.5779	0.1204	0.2498	2584.608183	1792.876844	16.681562	0.036267	8.161194	0.511595	45.874935	0.347738
9995.5	2.3327	301.9115	68.8700	0.2206	0.3501	2584.380270	1717.496602	15.200142	0.104503	16.297411	0.506155	48.149481	0.668719
9996.0	2.1989	374.7152	86.1447	0.2989	0.3950	2586.803905	1662.831462	13.958512	0.147908	20.912597	0.501596	49.786717	0.837415
9996.5	2.1435	410.1278	104.3522	0.3313	0.4175	2594.128656	1611.972599	13.205936	0.185497	25.513277	0.498832	51.345814	0.996109
9997.0	2.1412	409.7719	151.3408	0.3326	0.4316	2626.635184	1529.602936	12.458496	0.243429	33.375970	0.496088	54.050488	1.244732

In [16]:
```
writer = pd.ExcelWriter('Well-'+Well_name+'.xlsx', engine='xlsxwriter')
final_table.to_excel(writer,sheet_name=Well_name)
writer.save()
```

The Machine Learning project can now begin, but by first combining all the excel files previously exported.

In [200]:
```
# Change directory
os.chdir("C:/Users/Jamal/Desktop/ULL/Fall 20/INFX 598/INFX 598-Project/Codes/All excel files")
# Create a list with all the files
path = os.getcwd()
files = os.listdir(path)
# Select only xlsx files
files_xlsx = [f for f in files if f[-4:] == "xlsx"]
```

In [201]:
```
cols = ['DEPT','RHOZ','GR_EDTC','AORT','DPHZ','NPHI','DTCO','DTSM','YM','PR','BI','Fracture toughness',
        'Minimum horizontal stress','FI']
lst = []
i=0
# Loop over list of Excel files
for f in files_xlsx:
    df=pd.read_excel(f,header=None,skiprows=1)
    lst.append(df)

df2=pd.concat(lst,axis=0)
df2.columns=cols
```

Data Exploration

Next, we explored the data to identify outliers and missing values. But before this, we shuffled the dataset.

```
In [202]: df2.reset_index(drop=True,inplace=True)
          df2.head()
Out[202]:
```

	DEPT	RHOZ	GR_EDTC	AORT	DPHZ	NPHI	DTCO	DTSM	YM	PR	BI	Fracture toughness	Minimum horizontal stress	FI
0	10712.0	2.5477	225.2252	120.6288	0.0949	0.1439	3092.438339	2158.377308	24.332668	0.025078	9.413415	0.539690	48.799045	0.357431
1	10712.5	2.4275	327.9264	133.7183	0.1652	0.4590	3094.425668	2104.132466	22.999520	0.069999	13.338805	0.534794	50.327295	0.495595
2	10713.0	2.3373	388.6479	142.0983	0.2180	0.4224	3352.021661	2060.865861	23.748102	0.196148	36.445674	0.537543	55.527107	1.221034
3	10713.5	2.2909	378.2459	187.9962	0.2451	0.3028	3382.727668	1990.143306	22.417466	0.235326	39.417476	0.532657	57.492959	1.287142
4	10714.0	2.2905	363.8178	239.3053	0.2453	0.2764	3262.845312	1962.800803	21.468825	0.216454	33.710115	0.529174	56.525352	1.126987

```
In [203]: df2 = df2.sample(frac=1,random_state=42).reset_index(drop=True) #shuffle the data
```

As observed, the important attributes which include compressional slowness (DTCO), Gamma Ray (GR_EDTC), Resistivity (AORT), Neutron Porosity (DPHI), shear slowness (DTSM), and Density Porosity (DPHZ) are float values. We also noticed that the dataset has some missing values.

```
In [178]: df2.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1787 entries, 0 to 1786
Data columns (total 14 columns):
DEPT                         1787 non-null float64
RHOZ                         1787 non-null float64
GR_EDTC                      1787 non-null float64
AORT                         1787 non-null float64
DPHZ                         1787 non-null float64
NPHI                         1787 non-null float64
DTCO                         1650 non-null float64
DTSM                         1787 non-null float64
YM                           1650 non-null float64
PR                           1650 non-null float64
BI                           1650 non-null float64
Fracture toughness           1650 non-null float64
Minimum horizontal stress    1650 non-null float64
FI                           1650 non-null float64
dtypes: float64(14)
memory usage: 195.5 KB
```

Furthermore, the features had outliers, especially compressional slowness (DTCO) and resistivity (AORT). This is confirmed by the skewness values, histograms, and box plots for these parameters as shown below.

```
In [179]: df2.describe()
Out[179]:
```

	DEPT	RHOZ	GR_EDTC	AORT	DPHZ	NPHI	DTCO	DTSM
count	1787.000000	1787.000000	1787.000000	1787.000000	1787.000000	1787.00000	1650.000000	1787.000000
mean	10295.198657	2.283486	395.532790	305.571822	0.249039	0.32581	3313.242660	1847.892796
std	619.072375	0.121418	122.202634	590.313083	0.070934	0.08610	6534.924058	247.999913
min	8147.000000	1.960300	151.806700	1.452810	-0.007800	0.00880	2433.016222	-65.211352
25%	9877.750000	2.207300	305.313050	37.382050	0.221350	0.27860	2834.124627	1684.857107
50%	10550.500000	2.263200	397.142600	121.859000	0.260900	0.32935	3006.555536	1806.465439
75%	10686.750000	2.330750	468.544900	316.490065	0.293600	0.38130	3201.801445	1941.173159
max	11276.500000	2.723300	937.895700	7190.411100	0.414200	0.62780	267673.662949	2864.287466

```
In [182]: df2.skew()
Out[182]: DEPT                        -0.698354
          RHOZ                         1.116874
          GR_EDTC                      0.495561
          AORT                         5.616226
          DPHZ                        -1.123025
          NPHI                        -0.426963
          DTCO                        40.195786
          DTSM                         1.155550
          YM                           1.055002
          PR                          -5.841437
          BI                           1.184996
          Fracture toughness           1.055002
          Minimum horizontal stress   -2.574010
          FI                           2.310649
          dtype: float64
```

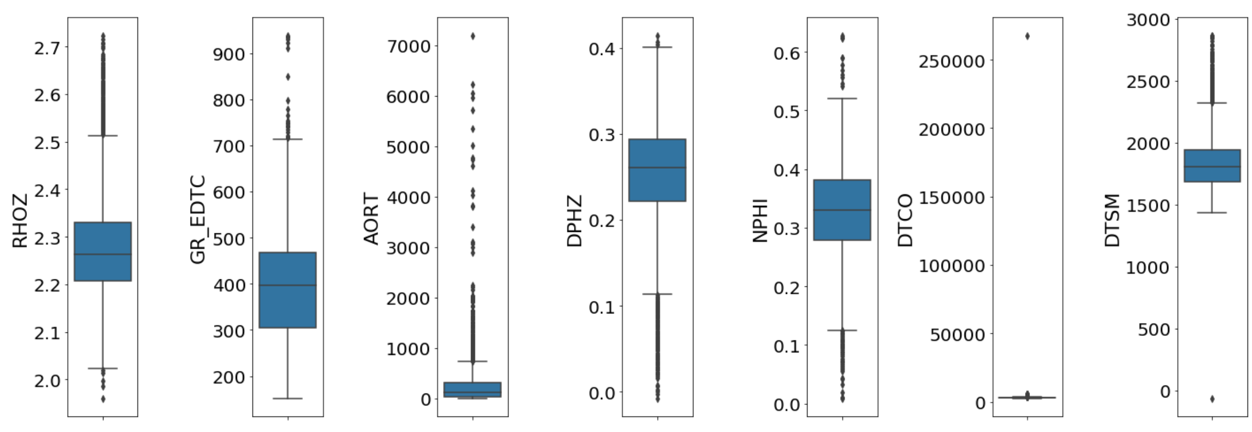


Figure 2. Boxplots of variables from the data collected.

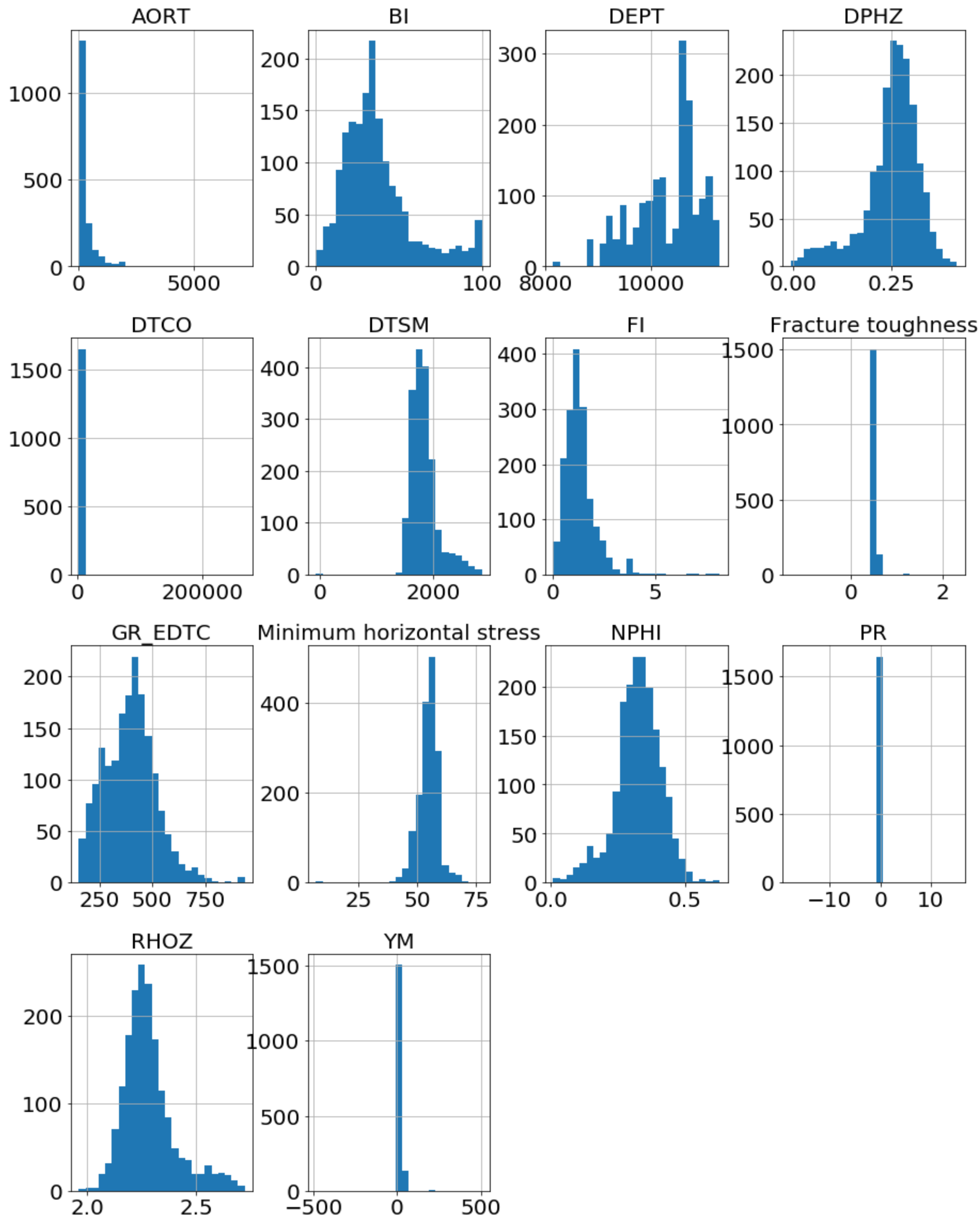


Figure 3. Histograms of all the features in the dataset.

Data Preparation

We removed the outliers using percentiles by viewing each variable separately and choosing different upper and lower percentiles. For instance, the porosities below the 1st percentile and above the 99th percentile are removed.

In [273]:
```
def remove_outliers_index(property,lower_quant,upper_quant):
    if lower_quant>0 and upper_quant>0:
        dat=df2[property]
        P_uq=dat.quantile(upper_quant)
        P_lq=dat.quantile(lower_quant)
        index = df2[(dat >= P_uq)|(dat<=P_lq)].index
    elif lower_quant>0 and upper_quant==0:
        dat=df2[property]
        P_lq=dat.quantile(lower_quant)
        index = df2[dat<=P_lq].index
    else:
        dat=df2[property]
        P_uq=dat.quantile(upper_quant)
        index = df2[dat >= P_uq].index
    return index
```

In [339]:
```
def remove_outliers_df(df):
    index_all=np.concatenate((remove_outliers_index("RHOZ",0.01,0.99),
    remove_outliers_index("AORT",0.01,0.99),
    remove_outliers_index("DPHZ",0.01,0.99),
    remove_outliers_index("NPHI",0.01,0.99),
    remove_outliers_index("DTCO",0,0.999),
    remove_outliers_index("DTSM",0.001,0),
    remove_outliers_index("YM",0.005,0.995),
    remove_outliers_index("PR",0.03,0.99)))
    unique_index=np.unique(index_all)
    df_f=df.drop(unique_index)
    return df_f
```

After removing outliers, the skewness values decreased significantly, and the box plots below describe a reduced variance of the variables. We also applied log transformation to the AORT variable since it still had a skewed distribution.

```
In [342]: df_mod.skew()

Out[342]: DEPT                          -0.664807
          RHOZ                           1.088581
          GR_EDTC                        0.282587
          AORT                           2.679069
          DPHZ                          -1.076884
          NPHI                          -0.305718
          DTCO                           2.099945
          DTSM                           1.468808
          YM                             2.132177
          PR                             0.946523
          BI                             1.270665
          Fracture toughness             2.132177
          Minimum horizontal stress      0.056079
          FI                             1.399167
          dtype: float64
```

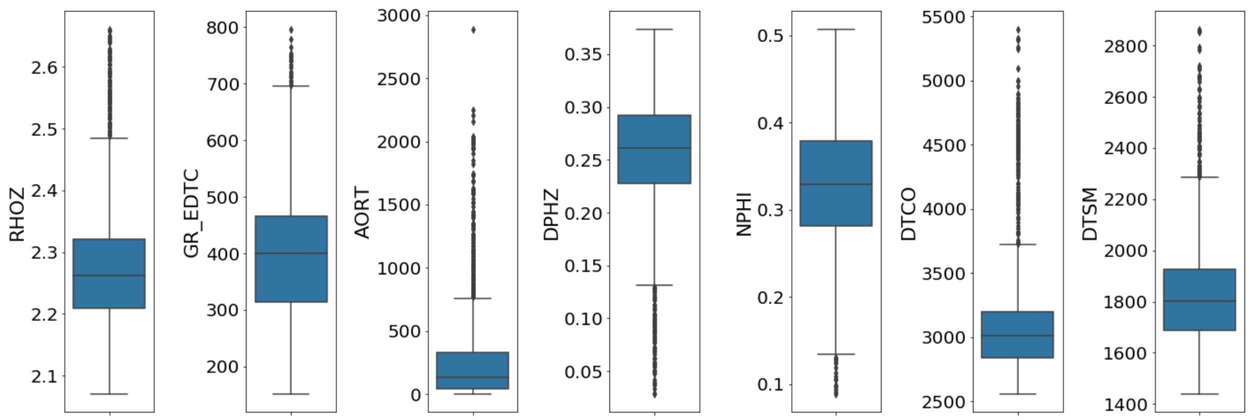


Figure 4. Boxplots of variables from the data collected after outlier removal.

In the end, we removed the rows that contained missing values and obtained the final dataset that was fed to the machine learning algorithms.

```
In [393]: df_f=df_mod.dropna()
          df_f.reset_index(drop=True,inplace=True)
```

Before applying the algorithms to the data, we divided it into the training and test sets. Our predictor features are density (RHOZ), gamma-ray (GR_EDTC), resistivity (AORT), density porosity (DPHZ), neutron porosity (NPHI), and compressional slowness (DTCO). Since our target variable, fracability index is not normally distributed some values are more common than others. That was why we split the fracability index into 4 categories and divided the dataset using stratified sampling. We also compared stratified sampling to random sampling. You can see that the proportion of each category in the test set obtained by stratified sampling is almost identical to the proportion of each category in the whole dataset. On the other hand, the random sampling results deviate from others significantly, especially for the 2nd and 4th categories. The test size is 20 % of the whole dataset.

```
In [437]: def strat_split(df_f,test):
              df_f["fi_cat"] = pd.cut(df_f["FI"],bins=[0, 1, 2, 3, np.inf],labels=[1, 2, 3, 4])
              df_f.dropna(inplace=True)
              df_f.reset_index(drop=True,inplace=True)
              split = StratifiedShuffleSplit(n_splits=1, test_size=test, random_state=42)
              for train_index, test_index in split.split(df_f, df_f["fi_cat"]):
                  strat_train_set = df_f.loc[train_index]
                  strat_test_set = df_f.loc[test_index]

              strat_prop=strat_test_set["fi_cat"].value_counts() / len(strat_test_set)
              full_data=df_f["fi_cat"].value_counts() / len(df_f)
              train_set, test_set = train_test_split(df_f, test_size=0.2, random_state=42)
              random=test_set["fi_cat"].value_counts() / len(test_set)
              compare=pd.concat([full_data,strat_prop,random],axis=1)
              compare.columns=['Full dataset','Stratified sampling','Random sampling']

              return strat_train_set, strat_test_set,compare
```

```
In [438]: train_set, test_set,sampling_table=strat_split(df_f,0.2)
```

Table 1. Comparison of splitting mechanisms.

FI category	Full dataset	Stratified sampling	Random sampling
1 (0-1)	**0.37**	**0.37**	**0.39**
2 (1-2)	**0.54**	**0.54**	**0.52**
3 (2-3)	**0.09**	**0.09**	**0.09**

Lastly, the features were scaled using scikit-learn's StandardScaler() function that subtracts the mean and divides by a standard deviation.

```
[ ]  from sklearn.preprocessing import MinMaxScaler, StandardScaler
     scaler=MinMaxScaler()
     X_train_scaled=pd.DataFrame(scaler.fit_transform(X_train))
     X_test_scaled=pd.DataFrame(scaler.fit_transform(X_test))
```

Shortlisting Promising Models

We trained several regression algorithms using 5-fold cross-validation and obtained the average root mean squared error of validation sets. The error on the whole training set for each model was also measured.

Table 2. Performance of several ML models on training set and validation set.

Algorithms	Model parameters	Training	Average CV
Linear Regression		0.46	0.46
Ridge Regression	alpha=1	0.46	0.46
Lasso Regression	alpha=0.01	0.46	0.47
Decision Tree	tree depth=31 (default)	0.00	0.54
SVM (kernel=Linear)	C=100	0.46	0.47
SVM (kernel=Polynomial)	degree=2 & C=10	0.46	0.47
SVM (kernel=RBF)	degree=3 & gamma='scale'	0.39	0.43
Random Forests	number of estimators=1000 tree depth=6	0.36	0.43

As the training set error is very similar to the average validation set error it can be concluded that Linear Regression, Ridge Regression, and Lasso Regression underfit the training data. These models are too simple to capture the relationships in our data. Since we cannot add more training data and cannot make these models more complex, we will omit them from further use. On the other hand, the complexity of the Decision Tree, SVM, and Random Forest models can be increased by fine-tuning the hyperparameters.

System Fine-Tuning

Table 3. Performance of shortlisted models with best model parameters on training set and validation set.

Algorithms	Model parameters	Best model parameters	Training	Average CV
Decision Tree	tree depth: 6-10 maximum leaf nodes: 2-50 minimum samples split: 1-3	tree depth: 6 maximum leaf nodes: 5 minimum samples split: 2	0.45	0.46
SVM (kernel=Linear)	C=1, 10, 100, 1000	C=1	0.46	0.47
SVM (kernel=RBF)	C=0.1, 1, 10, 100, 1000 gamma=1, 0.1, 0.01, 0.001, 0.0001	C=1 Gamma=1	0.27	0.40
Random Forests	number of estimators=30, 50, 70, 90, 110 tree depth=5, 7, 9 minimum samples leaf=1, 3, 5	number of estimators=90 tree depth=9 minimum samples leaf=1	0.28	0.41

We use a Grid Search method to evaluate all the possible combinations of specified hyperparameter values. The best model parameters for each algorithm correspond to the combination with the lowest average cross-validation error. Overall, the SVM model with RBF kernel and Random Forest outperformed Decision Tree and Linear SVM algorithms. The SVM model with the RBF kernel is slightly better than the Random Forest model. Therefore, the SVM model with the RBF kernel and parameters of C=1 and gamma=1 was chosen to evaluate the test set. The lower values of both C and gamma made sure that the model did not overfit the training data and can generalize well. Also, the smaller tree depth for both Decision Tree and Random Forests helped models to generalize well.

Solution Presentation

The RMSE obtained from the test set is 0.36. All the models can be considered as a good fit where the cross-validation error is slightly higher than the training error and they both are relatively low. However, several limitations can prevent reducing the training error and validation error further. First and foremost, the sample size for this study was fairly small. Adding more instances will lead to better training and prediction of the models. Secondly, there might be a need for the removal of outliers further. Especially, the prediction for the higher values of the fracability index can be difficult as it was seen in the test set. Moreover, complex models such as ensemble methods and neural networks can be used to capture more complex relationships in the dataset. Techniques, such as bagging and boosting, can also be applied to improve the model's results.

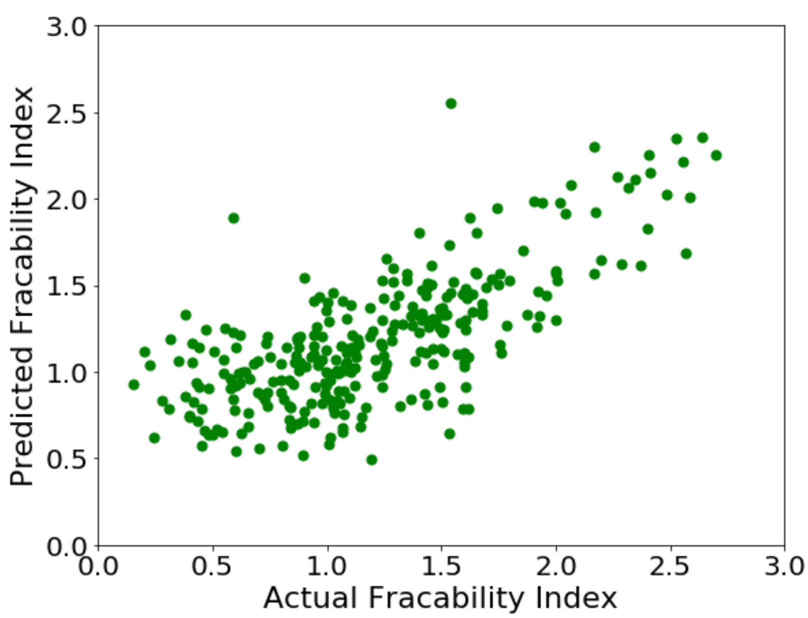
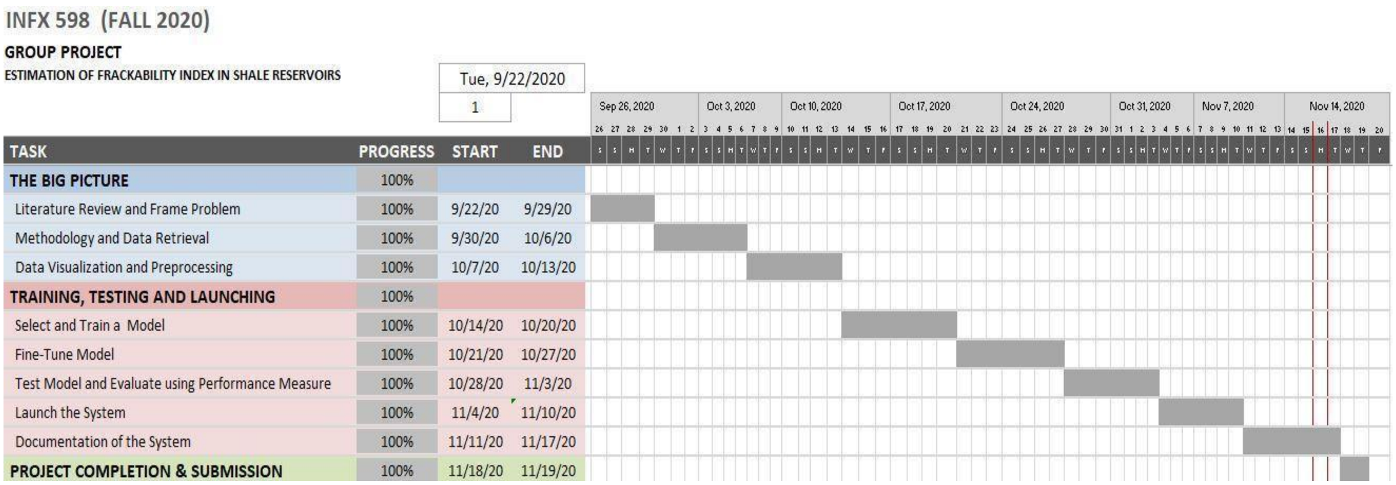


Figure 5. Actual vs. Predicted Fracability Index.

Appendix

Gantt Chart



INFX 598 (FALL 2020)

GROUP PROJECT

ESTIMATION OF FRACKABILITY INDEX IN SHALE RESERVOIRS

Tue, 9/22/2020

1

TASK	PROGRESS	START	END
THE BIG PICTURE	100%		
Literature Review and Frame Problem	100%	9/22/20	9/29/20
Methodology and Data Retrieval	100%	9/30/20	10/6/20
Data Visualization and Preprocessing	100%	10/7/20	10/13/20
TRAINING, TESTING AND LAUNCHING	100%		
Select and Train a Model	100%	10/14/20	10/20/20
Fine-Tune Model	100%	10/21/20	10/27/20
Test Model and Evaluate using Performance Measure	100%	10/28/20	11/3/20
Launch the System	100%	11/4/20	11/10/20
Documentation of the System	100%	11/11/20	11/17/20
PROJECT COMPLETION & SUBMISSION	100%	11/18/20	11/19/20

Link to GitHub Repository

https://github.com/jamalahmad1996/Machine-Learning-Applications-INFX-598/blob/master/Project.ipynb

Contribution Made by Each Group Member

Jamal Ahmadov:

- Completed literature review on the estimation of fracability index from geomechanical properties, such as brittleness index, fracture toughness, and minimum horizontal stress.
- Collected well log data for 44 wells from NDIC Oil and Gas website.
- Wrote a Python code in Jupyter Notebook to read the raw data, choose the required well logs, identify Upper Bakken zone, estimate geomechanical properties and fracability index and export the data frame as an excel file.
- Wrote a Python code in Jupyter Notebook to build the whole dataset and perform data preprocessing that included data visualization, outlier removal, and fixing missing values.
- Wrote a Python code to train several ML models, measure their performance, shortlist 3 algorithms, fine-tune their hyperparameters, choose the best model based on Grid Search, and evaluate it on the test set.
- Documented “Shortlist Promising Models”, “System Fine-Tuning” and “Solution Presentation” sections of the project.
- Uploaded all the source code to the GitHub account.

Silas Adeoluwa Samuel:

- Completed literature review on the estimation of fracability index from geomechanical properties, such as brittleness index, fracture toughness, and minimum horizontal stress.
- Collected well log data for 28 wells from NDIC Oil and Gas website.
- Selected 11 wells with the required well logs and modified Jamal’s Python code to read the raw data, identify Upper Bakken zone, estimate geomechanical properties and fracability index as well as export the data frame as an excel file.
- Created a Gantt Chart using an online template as well as updating it as the project progressed.
- Wrote codes for the following models: SVR with different kernels, Linear Regression, Decision Trees, and Random Forest. My codes were compared with those written by other teammates and modifications were made.

- Wrote portions of the final report (Data Acquisition & Preparation) and carried out final editing of the entire group report.

Shamsul Hoque:

- Completed literature review on the estimation of the fracability index from geomechanical properties, such as brittleness index, fracture toughness, and minimum horizontal stress.
- Collected well log data for 24 wells from NDIC Oil and Gas website.
- Modified Jamal's Python code to read the raw data, identify the Upper Bakken zone, estimate geomechanical properties and fracability index as well as export the data frame as an excel file.
- Wrote the summary of accomplishments on different phases of the project.
- Wrote codes with different training models (e.g. Linear Regression, Support Vector Regression, RandomForest, etc.) to evaluate their performance, ran GridSearch for choosing the best parameters, and applied the best model on the test set. Compare results with other group members.
- Wrote several portions of the final report- Statement of the Problem and Big Picture View, Data Acquisition, and Data Visualization.

References

- He, Rui, Zhaozhong Yang, Xiaogang Li, Zhanling Li, Ziyuan Liu, and Fei Chen. "A comprehensive approach for fracability evaluation in naturally fractured sandstone reservoirs based on analytical hierarchy process method." *Energy Science & Engineering* 7, no. 2 (2019): 529-545.
- Ibrahim Mohamed, Mohamed, Mohamed Salah, Yakup Coskuner, Mazher Ibrahim, Chester Pieprzica, and Erdal Ozkan. "Integrated Approach to Evaluate Rock Brittleness and Fracability for Hydraulic Fracturing Optimization in Shale Gas." In *SPE Oklahoma City Oil and Gas Symposium*. Society of Petroleum Engineers, 2019.
- Jin, Xiaochun, Subhash N. Shah, Jean-Claude Roegiers, and Bo Zhang. "An integrated petrophysics and geomechanics approach for fracability evaluation in shale reservoirs." *SPE Journal* 20, no. 03 (2015): 518-526.
- Li, Jing, Xiao-Rong Li, Hong-Bin Zhan, Ming-Shui Song, Chen Liu, Xiang-Chao Kong, and Lu-Ning Sun. "Modified method for fracability evaluation of tight sandstones based on interval transit time." *Petroleum Science* 17, no. 2 (2020): 477-486.
- Parapuram, George K., Mehdi Mokhtari, and Jalel Ben Hmida. "Prediction and Analysis of Geomechanical Properties of the Bakken Shale Using Artificial Intelligence and Data Mining." In *Unconventional Resources Technology Conference, Austin, Texas, 24-26 July 2017*, pp. 2815-2833. Society of Exploration Geophysicists, American Association of Petroleum Geologists, Society of Petroleum Engineers, 2017.
- Steptoe, Anne. "Petrofacies and depositional systems of the Bakken Formation in the Williston Basin, North Dakota." (2012).
- Sui, Haoyue, Wei Gao, and Ruilin Hu. "A New Evaluation Method for the Fracability of a Shale Reservoir Based on the Structural Properties." *Geofluids* 2019 (2019).
- Sui, Lili, Jiangtao Zheng, Jian Yu, and Xianxia Wang. "Comprehensive evaluation on shale fracability using principal component analysis." *Electron. J. Geotech. Eng* 20 (2015): 5965-5976.
- Wood, David A. "Bakken stratigraphic and type well log learning network exploited to predict and data mine shear wave acoustic velocity." *Journal of Applied Geophysics* 173 (2020): 103936.

- Yuan, Junliang, Jianliang Zhou, Shujie Liu, Yongcun Feng, Jingen Deng, Qingming Xie, and Zhaohui Lu. "An improved fracability-evaluation method for shale reservoirs based on new fracture toughness-prediction models." *SPE Journal* 22, no. 05 (2017): 1-704.
- Zhixi, Chen, Chen Mian, Jin Yan, and Huang Rongzun. "Determination of rock fracture toughness and its relationship with acoustic velocity." *International Journal of Rock Mechanics and Mining Sciences* 34, no. 3-4 (1997):49-e1.