Machine learning to interprete lithologies from geophysical logs

Himanshu Bhardwaj

Description of Logs:

1. GR: Natural Gamma Ray log (API)
2. DPHI: Density porosity
3. NPHI: Neutron porosity
4. PEF: Photoelectric factor (barns/sec)

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df = pd.read_csv('DataSet.csv')
df.head()
```

Out[2]:

	GR	NPHI	DPHI	PEF	PICK	TRUE
0	18.4445	0.1263	0.0973	2.5341	7	7
1	18.4814	0.1138	0.0882	2.5683	7	7
2	17.9632	0.1100	0.0890	2.5396	7	7
3	16.0150	0.1213	0.1170	2.3682	7	7
4	14.6361	0.1112	0.1520	2.1106	10	10

In [3]:

```
df['LithCode'] = df['TRUE']
df=df.drop(['PICK','TRUE'], axis=1)
df.head()
```

Out[3]:

	GR	NPHI	DPHI	PEF	LithCode
0	18.4445	0.1263	0.0973	2.5341	7
1	18.4814	0.1138	0.0882	2.5683	7
2	17.9632	0.1100	0.0890	2.5396	7
3	16.0150	0.1213	0.1170	2.3682	7
4	14.6361	0.1112	0.1520	2.1106	10

In [4]:

```
import csv
input_file = csv.DictReader(open("Lithology.csv"))
```

In [5]:

```
import csv
with open('Lithology.csv') as f:
    d = dict(filter(None, csv.reader(f)))
print(d)
```

```
{'1': 'Unkown', '2': 'Halite', '3': 'Gypsum', '4': 'Dolomite', '5': 'Dolomitic Limestone', '6': 'Che
rty Dolomitic Limestone', '7': 'Cherty Dolomite', '8': 'Limestone', '9': 'Cherty Limestone', '10': '
Chert', '11': 'Shale', '12': 'Sand Stone', '13': 'Ironstone', '14': 'Coal'}
```

In [6]:

```
df['Lithology'] = df['LithCode'].apply(lambda x: d[str(x)])
df.head()
```

Out[6]:

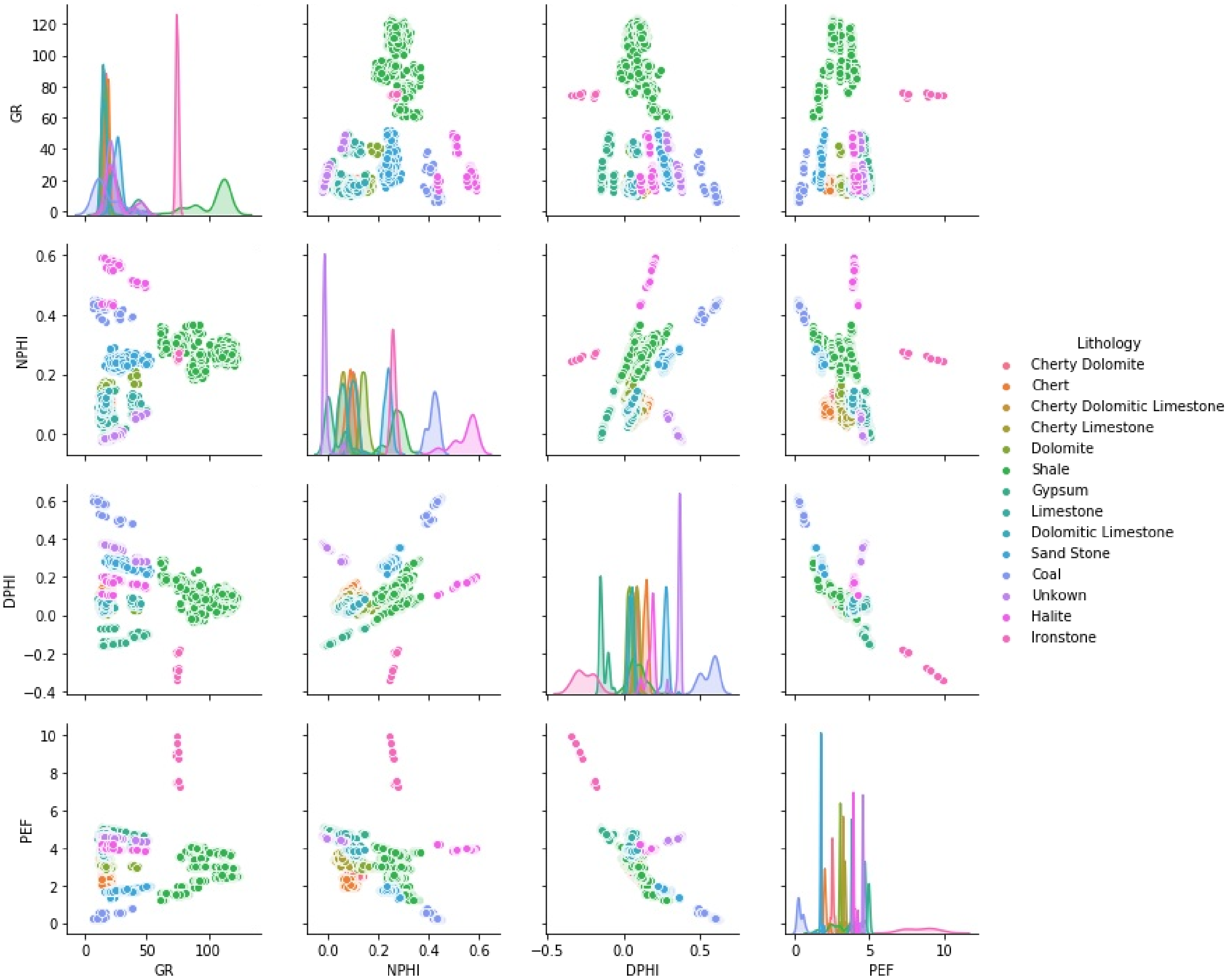
	GR	NPHI	DPHI	PEF	LithCode	Lithology
0	18.4445	0.1263	0.0973	2.5341	7	Cherty Dolomite
1	18.4814	0.1138	0.0882	2.5683	7	Cherty Dolomite
2	17.9632	0.1100	0.0890	2.5396	7	Cherty Dolomite
3	16.0150	0.1213	0.1170	2.3682	7	Cherty Dolomite
4	14.6361	0.1112	0.1520	2.1106	10	Chert

In [7]:

```
sns.pairplot(df.drop('LithCode',axis=1), hue='Lithology')
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1ececea5c48>
```

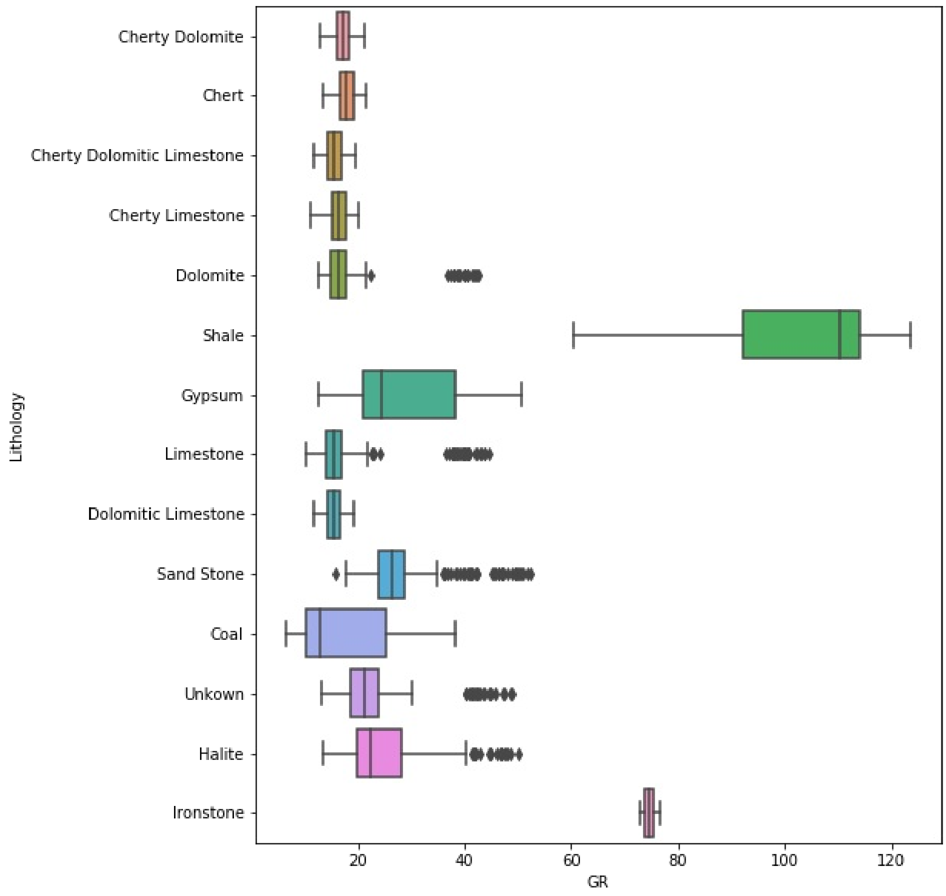


In [8]:

```
plt.figure(figsize=(8,10))
sns.boxplot(y=df['Lithology'], x=df['GR'], orient="h")
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x1ecedd9d648>

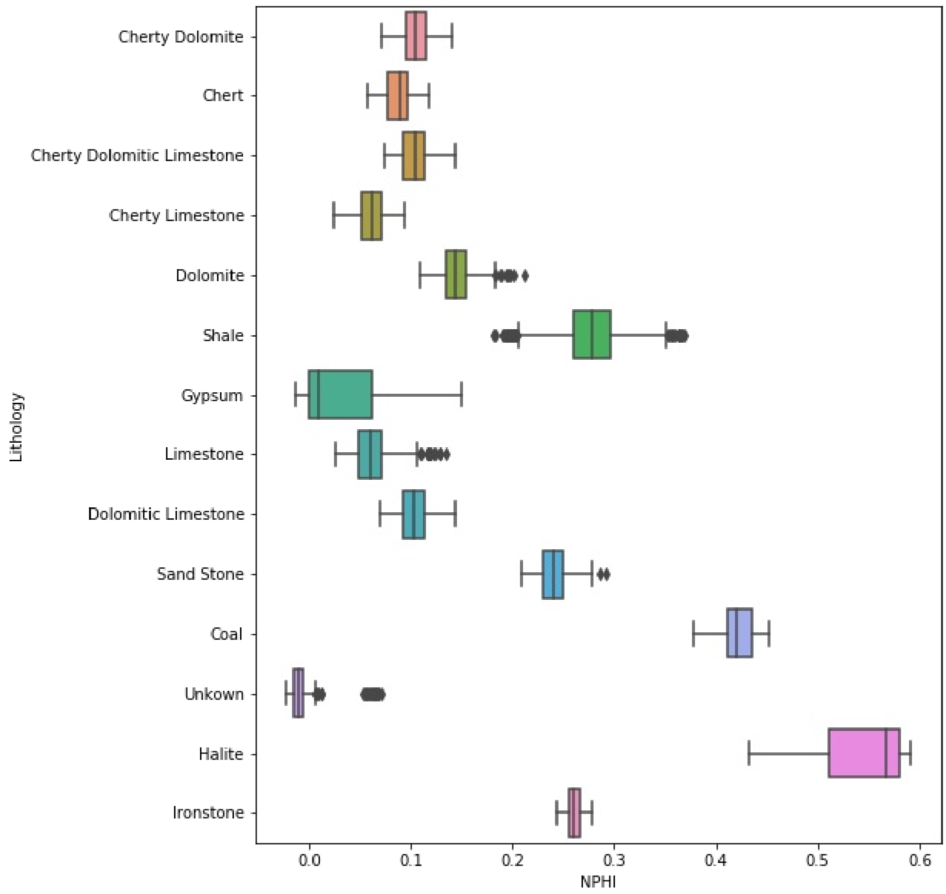


In [9]:

```
plt.figure(figsize=(8,10))
sns.boxplot(y=df['Lithology'], x=df['NPHI'], orient="h")
```

Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x1ecee2cf248>

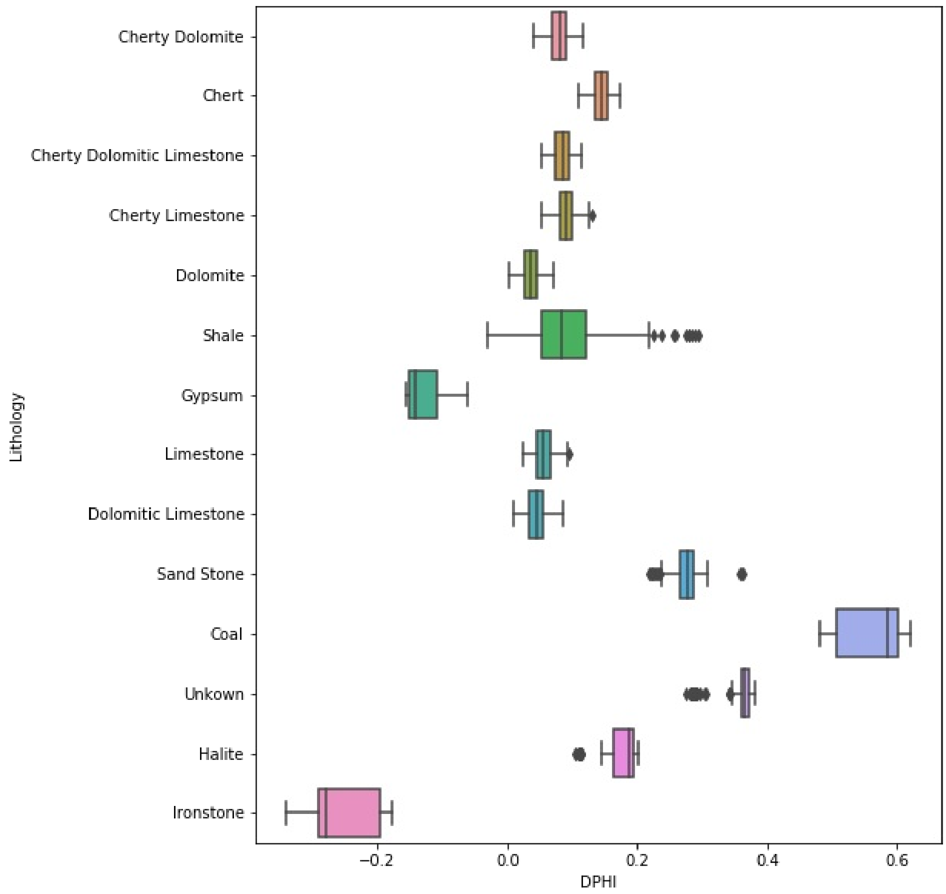


In [10]:

```
plt.figure(figsize=(8,10))
sns.boxplot(y=df['Lithology'], x=df['DPHI'], orient="h")
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ecedadfe48>
```

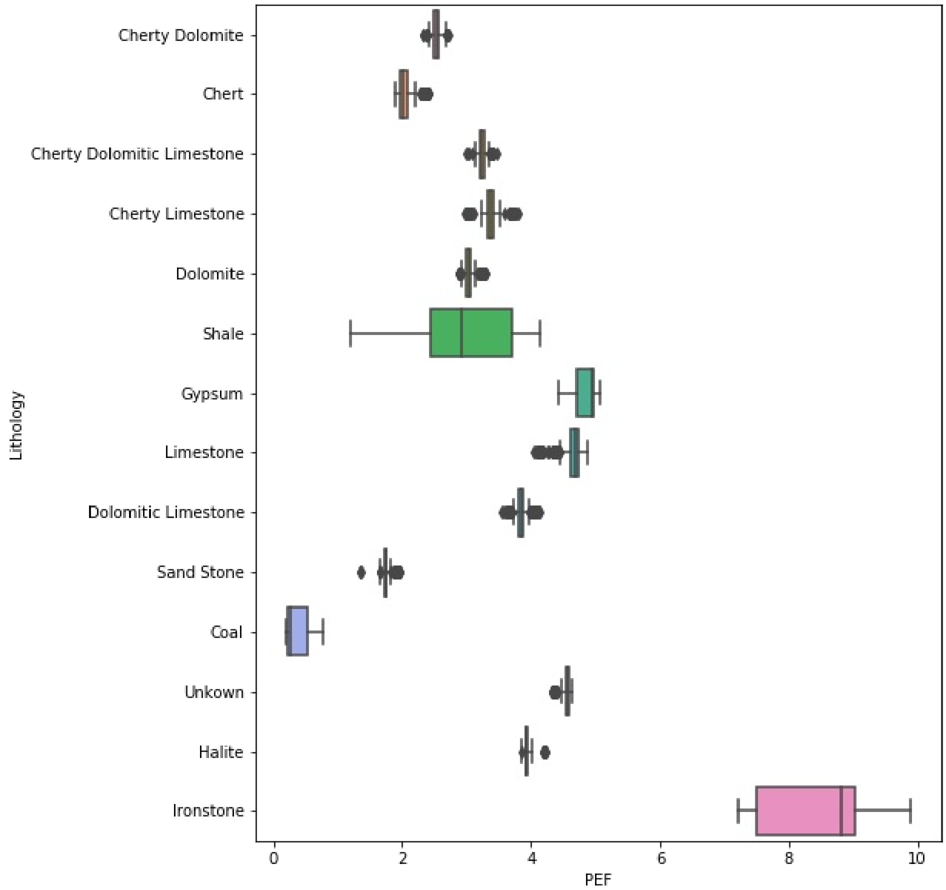


In [11]:

```
plt.figure(figsize=(8,10))
sns.boxplot(y=df['Lithology'], x=df['PEF'], orient="h")
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ecee6947c8>
```



In [12]:

```
df.head()
```

Out[12]:

	GR	NPHI	DPHI	PEF	LithCode	Lithology
0	18.4445	0.1263	0.0973	2.5341	7	Cherty Dolomite
1	18.4814	0.1138	0.0882	2.5683	7	Cherty Dolomite
2	17.9632	0.1100	0.0890	2.5396	7	Cherty Dolomite
3	16.0150	0.1213	0.1170	2.3682	7	Cherty Dolomite
4	14.6361	0.1112	0.1520	2.1106	10	Chert

In [13]:

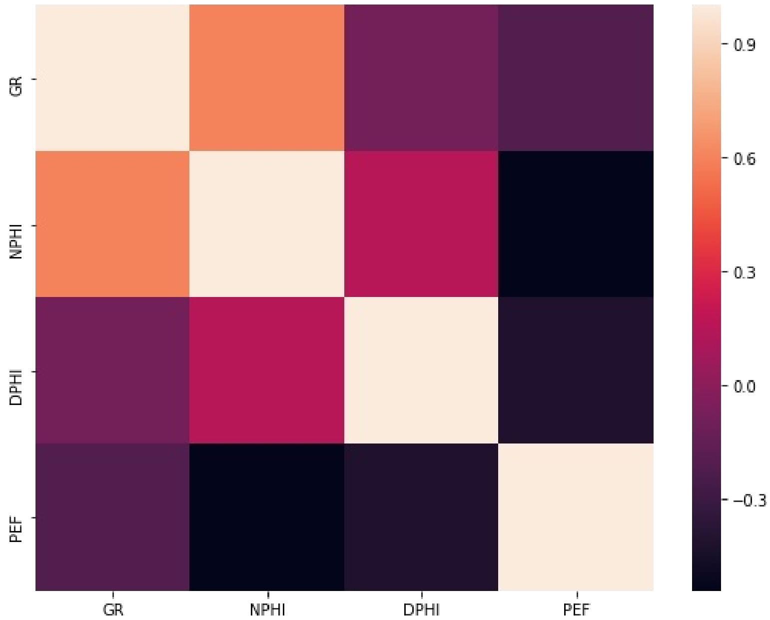
```
dt = df[['GR', 'NPHI', 'DPHI', 'PEF']]
cor_dt = dt.corr()
```

In [14]:

```
plt.figure(figsize=(9,7))
sns.heatmap(cor_dt, label=True)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ecee9c7688>
```



In [15]:

```
dt['LithCode'] = df['LithCode']
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [16]:

```
dt.head()
```

Out[16]:

	GR	NPHI	DPHI	PEF	LithCode
0	18.4445	0.1263	0.0973	2.5341	7
1	18.4814	0.1138	0.0882	2.5683	7
2	17.9632	0.1100	0.0890	2.5396	7
3	16.0150	0.1213	0.1170	2.3682	7
4	14.6361	0.1112	0.1520	2.1106	10

In [17]:

```
from sklearn.preprocessing import StandardScaler
```

In [18]:

```
features = dt.drop('LithCode', axis=1)
targets = dt['LithCode']
```

In [19]:

```
scaler = StandardScaler()
std_features = scaler.fit_transform(features)
```

In [20]:

```
X_features = pd.DataFrame(data=std_features, columns=features.columns)
X_features.head()
```

Out[20]:

	GR	NPHI	DPHI	PEF
0	-0.597337	-0.247507	-0.090083	-0.765823
1	-0.596353	-0.352919	-0.160980	-0.734030
2	-0.610163	-0.384965	-0.154747	-0.760710
3	-0.662079	-0.289672	0.063398	-0.920046
4	-0.698824	-0.374845	0.336080	-1.159514

In [21]:

```
from sklearn.model_selection import train_test_split
```

In [22]:

```
X_train, X_test, y_train, y_test = train_test_split(X_features, targets, train_size=0.7, random_state=122)
```

In [23]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(3500, 4)
(3500,)
(1500, 4)
(1500,)
```

In [24]:

```
X_features.describe()
```

Out[24]:

	GR	NPHI	DPHI	PEF
count	5.000000e+03	5.000000e+03	5.000000e+03	5.000000e+03
mean	-2.953193e-17	-2.857714e-17	5.556111e-17	-3.763656e-17
std	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00
min	-9.148307e-01	-1.502337e+00	-3.500164e+00	-2.934056e+00
25%	-6.621496e-01	-7.172247e-01	-5.162455e-01	-8.135351e-01
50%	-5.595014e-01	-3.031646e-01	-2.770646e-01	4.238435e-02
75%	2.707103e-01	8.211644e-01	3.563364e-01	6.836093e-01
max	2.204341e+00	3.671306e+00	3.984563e+00	6.069262e+00

In [25]:

```
import Machine_Learning as ML
```

In [26]:

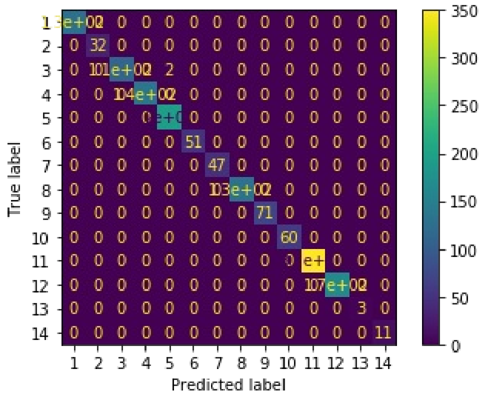
```
models = ML.Models_ret()
```

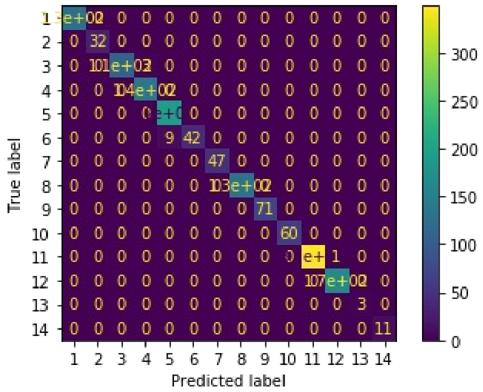
Model fitting with regularized logistic regresseion, LDA, KNN, Gradient Boost, ADA boost, Random forest, Decision tree classifier and corresponding confusion matrix

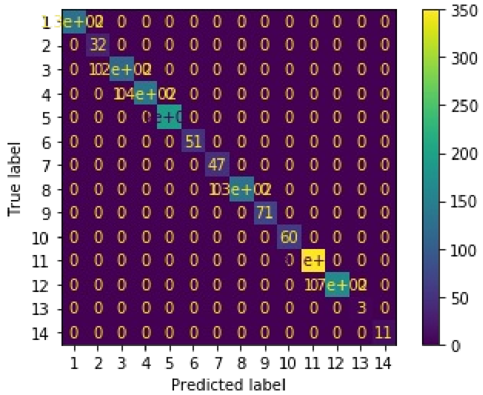
In [27]:

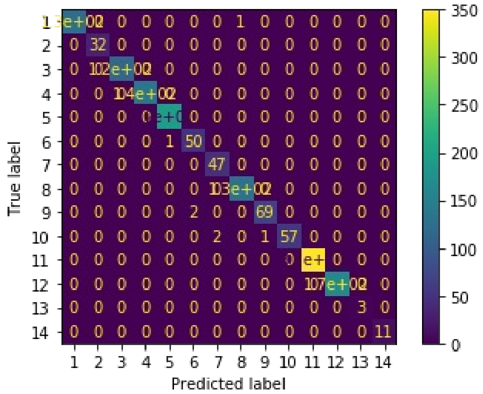
```
models_fit = ML.fit_model(models, X_train, y_train, X_test, y_test)
```

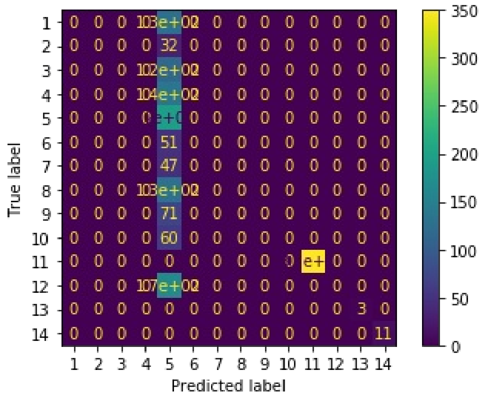
```
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

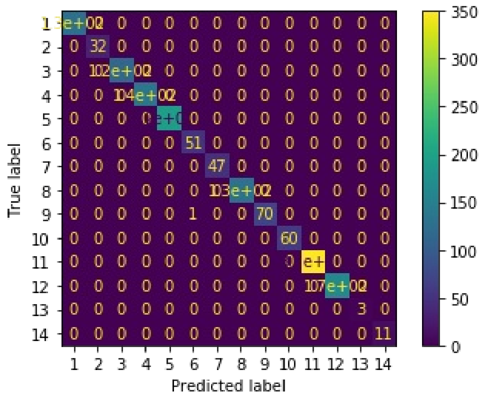


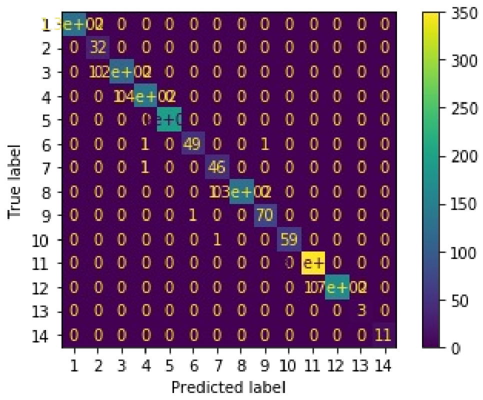












In [28]:

```
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import plot_confusion_matrix
```

In [29]:

```
voting_clf = VotingClassifier(estimators=models_fit, voting='soft')
voting_clf.fit(X_train, y_train)
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[29]:

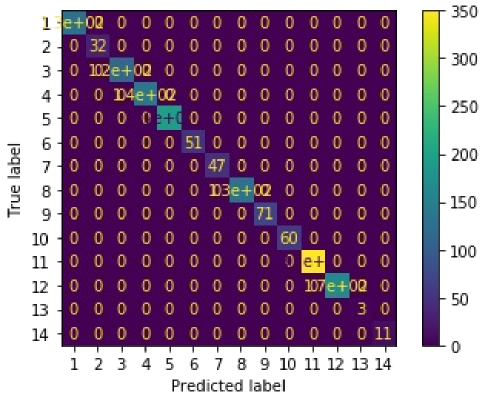
```
VotingClassifier(estimators=[['LR',
                              LogisticRegression(C=5.0, class_weight=None,
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=100,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=None,
                                                 solver='lbfgs', tol=0.0001,
                                                 verbose=0, warm_start=False)],
                             ['LDA',
                              LinearDiscriminantAnalysis(n_components=None,
                                                         priors=None,
                                                         shrinkage=None...
                              DecisionTreeClassifier(ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='gini',
                                                     max_depth=None,
                                                     max_features=None,
                                                     max_leaf_nodes=None,
                                                     min_impurity_decrease=0.0,
                                                     min_impurity_split=None,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_leaf=0.0,
                                                     presort='deprecated',
                                                     random_state=None,
                                                     splitter='best')]],
                 flatten_transform=True, n_jobs=None, voting='soft',
                 weights=None)
```

The confusion matrix of voting classifier to get best classifications

In [30]:

```
plt.figure(figsize=(12,8))
plot_confusion_matrix(voting_clf, X_test,y_test)
plt.show()
```

```
<Figure size 864x576 with 0 Axes>
```



Using the model to interprete Geophysical logs

In [31]:

```
df = pd.read_csv('GPH.csv')
df.head()
```

Out[31]:

	DEPT	MNOR	MINV	NPHS	NPHL	NPHI	NPHD	RHOB	QN	QF	...	RT30	RT20	RT10	RT	RMUD
0	1510.0	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	...	16.0478	4.9457	3.3305	1999.9999	0.7480
1	1510.5	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	...	18.5982	5.8843	3.9779	1999.9999	0.7493
2	1511.0	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	...	18.9170	6.0046	4.0613	1999.9999	0.7462
3	1511.5	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	...	17.3583	5.5102	3.7269	1999.9999	0.7506
4	1512.0	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	-999.25	...	15.9935	5.0936	3.4468	1999.9999	0.7484

5 rows × 30 columns

In [32]:

```
dt = df[['DEPT','GR','NPHI','DPHI','PE']]
dt = dt[dt != -999.25].dropna()
dt['Depth'] = dt['DEPT']
dt = dt.drop('DEPT', axis=1)
dt.head()
```

Out[32]:

	GR	NPHI	DPHI	PE	Depth
3370	74.7029	0.0420	0.1074	4.7732	3195.0
3371	72.7194	0.0371	0.0875	4.7937	3195.5
3372	69.7554	0.0371	0.0756	4.8635	3196.0
3373	74.6137	0.0465	0.0722	4.8484	3196.5
3374	75.5497	0.0658	0.0789	4.7579	3197.0

In [33]:

```
dt = dt[['Depth', 'GR','NPHI','DPHI','PE']]
dt.head()
```

Out[33]:

	Depth	GR	NPHI	DPHI	PE
3370	3195.0	74.7029	0.0420	0.1074	4.7732
3371	3195.5	72.7194	0.0371	0.0875	4.7937
3372	3196.0	69.7554	0.0371	0.0756	4.8635
3373	3196.5	74.6137	0.0465	0.0722	4.8484
3374	3197.0	75.5497	0.0658	0.0789	4.7579

In [34]:

```
dt.index = dt['Depth']
```

In [35]:

```
import hemi_welllogs as hb
```

```
<matplotlib.colors.ListedColormap object at 0x000001ECF3D0F708>
```

Blocking the geophysical logs

In [36]:

```
ddf = hb.block_df(dt)
```

In [37]:

```
ddf[3200:3400].plot(subplots=True, figsize=(12,10), grid=True)
```

Out[37]:

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001ECF3F8FF08>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001ECF43C8208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001ECF43FC508>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001ECF442DB88>],
      dtype=object)
```

In [38]:

```
ddf.tail()
```

Out[38]:

	GR	NPHI	DPHI	PE
Depth				
5014.526870	53.808508	0.128009	0.055633	4.125405
5014.542048	53.808508	0.128009	0.055633	4.125405
5014.545328	53.808508	0.128009	0.055633	4.125405
5014.595846	53.808508	0.128009	0.055633	4.125405
5014.646365	53.808508	0.128009	0.055633	4.125405

In [39]:

```
scale = StandardScaler()
features = scale.fit_transform(ddf)
features
```

Out[39]:

```
array([[-0.02565548, -1.21609564, -0.49924943,  1.48724866],
       [-0.02565548, -1.21609564, -0.49924943,  1.48724866],
       [-0.02565548, -1.21609564, -0.49924943,  1.48724866],
       ...,
       [-0.5260154 , -0.28514321, -0.79436105,  0.61228787],
       [-0.5260154 , -0.28514321, -0.79436105,  0.61228787],
       [-0.5260154 , -0.28514321, -0.79436105,  0.61228787]])
```

In [40]:

```
interpretation = voting_clf.predict(features)
```

In [41]:

```
interpretation
```

Out[41]:

```
array([8, 8, 8, ..., 5, 5, 5], dtype=int64)
```

In [42]:

```
ddf_interpret = ddf
ddf_interpret['Lith_pred'] = interpretation
ddf_interpret.head()
```

Out[42]:

	GR	NPHI	DPHI	PE	Lith_pred
Depth					
3195.000000	76.160038	0.03624	0.071991	4.763356	8
3195.037941	76.160038	0.03624	0.071991	4.763356	8
3195.049475	76.160038	0.03624	0.071991	4.763356	8
3195.075883	76.160038	0.03624	0.071991	4.763356	8
3195.098951	76.160038	0.03624	0.071991	4.763356	8

In [43]:

```
ddf['Lithology'] = ddf['Lith_pred'].apply(lambda x: d[str(x)])
ddf.head()
```

Out[43]:

Depth	GR	NPHI	DPHI	PE	Lith_pred	Lithology
3195.000000	76.160038	0.03624	0.071991	4.763356	8	Limestone
3195.037941	76.160038	0.03624	0.071991	4.763356	8	Limestone
3195.049475	76.160038	0.03624	0.071991	4.763356	8	Limestone
3195.075883	76.160038	0.03624	0.071991	4.763356	8	Limestone
3195.098951	76.160038	0.03624	0.071991	4.763356	8	Limestone

In [44]:

```
ddf[3370:3400]
```

Out[44]:

Depth	GR	NPHI	DPHI	PE	Lith_pred	Lithology
3370.000789	48.885117	0.07428	0.063861	4.403666	8	Limestone
3370.034124	48.885117	0.07428	0.063861	4.403666	8	Limestone
3370.038385	48.885117	0.07428	0.063861	4.403666	8	Limestone
3370.049409	48.885117	0.07428	0.063861	4.403666	8	Limestone
3370.077483	48.885117	0.07428	0.063861	4.403666	8	Limestone
...	...	...	...	...	...	...
3399.950661	34.923290	0.03349	0.032480	4.885684	3	Gypsum
3399.950833	44.316527	0.03349	0.032480	4.885684	3	Gypsum
3399.951212	44.316527	0.03349	0.032480	4.885684	3	Gypsum
3399.973109	44.316527	0.03349	0.032480	4.885684	3	Gypsum
3399.991659	44.316527	0.03349	0.032480	4.885684	3	Gypsum

2561 rows × 6 columns

In [45]:

```
Litho = ddf[['Lith_pred','Lithology']]
Litho['Depth'] = ddf.index
Litho.head()
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
```

Out[45]:

Depth	Lith_pred	Lithology	Depth
3195.000000	8	Limestone	3195.000000
3195.037941	8	Limestone	3195.037941
3195.049475	8	Limestone	3195.049475
3195.075883	8	Limestone	3195.075883
3195.098951	8	Limestone	3195.098951

In [46]:

```
top = 0
bot = 0
width = 0
height = 0
Lithcode = 0
Lithology = 'a'
```

In [47]:

```
Litho_interpret = pd.DataFrame(columns=['top','bottom', 'width', 'height'])
```

In [48]:

```
top = Litho['Depth'].iloc[0]
nn = []
for i in range(len(Litho['Lith_pred'])-1):
    if(Litho.iloc[i+1,0]!=Litho.iloc[i,0]):
        bot = Litho.iloc[i,2]
        Lithcode = Litho.iloc[i,0]
        Lithology = Litho.iloc[i,1]
        height = bot-top
        width = 1
        nn.append([top, bot, width, height, Lithcode, Lithology])
        top = bot
```

Final interpretation

In [49]:

```
Litho_interpret = pd.DataFrame(nn, columns=['top','bottom', 'width', 'height','LithCode','Lithology'])
Litho_interpret.head()
```

Out[49]:

	top	bottom	width	height	LithCode	Lithology
0	3195.000000	3199.311941	1	4.311941	8	Limestone
1	3199.311941	3199.423373	1	0.111432	9	Cherty Limestone
2	3199.423373	3199.683504	1	0.260131	5	Dolomitic Limestone
3	3199.683504	3200.414598	1	0.731094	9	Cherty Limestone
4	3200.414598	3205.904068	1	5.489469	3	Gypsum

In [50]:

```
from matplotlib.patches import Rectangle
```

In [51]:

```
dt['Depth'] = dt.index
ddf['Depth'] = ddf.index
```

In [52]:

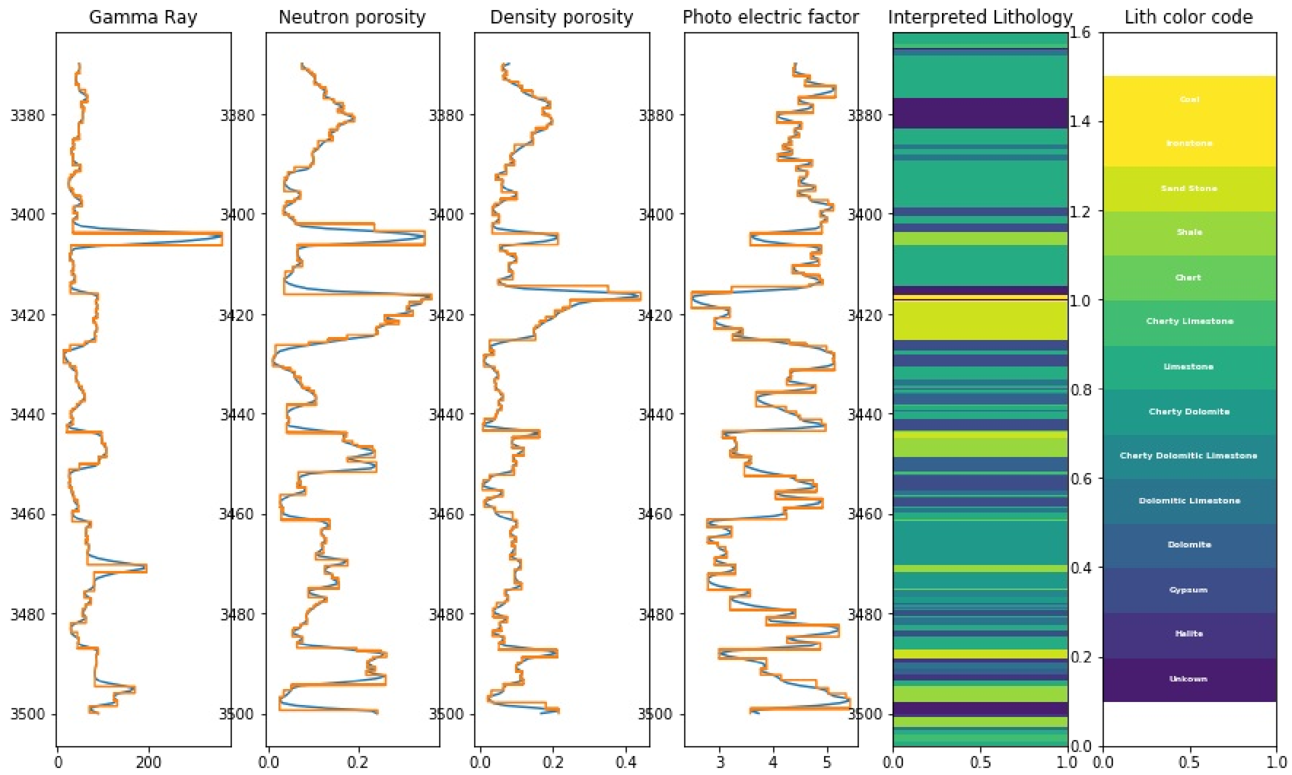
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap

viridis = cm.get_cmap('viridis', 14)
print(viridis)
```

```
<matplotlib.colors.ListedColormap object at 0x000001ECF46F0D08>
```

Interpretation and Geophysical logs plot with interpreted Lithologies

In [56]:

```
hb.plot_interpretation(dt,ddf,Litho_interpret,d)
```



In [57]:

```
ddf
```

Out[57]:

Depth	GR	NPHI	DPHI	PE	Lith_pred	Lithology	Depth
3195.000000	76.160038	0.036240	0.071991	4.763356	8	Limestone	3195.000000
3195.037941	76.160038	0.036240	0.071991	4.763356	8	Limestone	3195.037941
3195.049475	76.160038	0.036240	0.071991	4.763356	8	Limestone	3195.049475
3195.075883	76.160038	0.036240	0.071991	4.763356	8	Limestone	3195.075883
3195.098951	76.160038	0.036240	0.071991	4.763356	8	Limestone	3195.098951
...	...	...	...	...	...	...	...
5014.526870	53.808508	0.128009	0.055633	4.125405	5	Dolomitic Limestone	5014.526870
5014.542048	53.808508	0.128009	0.055633	4.125405	5	Dolomitic Limestone	5014.542048
5014.545328	53.808508	0.128009	0.055633	4.125405	5	Dolomitic Limestone	5014.545328
5014.595846	53.808508	0.128009	0.055633	4.125405	5	Dolomitic Limestone	5014.595846
5014.646365	53.808508	0.128009	0.055633	4.125405	5	Dolomitic Limestone	5014.646365

141477 rows × 7 columns

In [ ]: