Part 1

In [302…
```
#Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib.lines import Line2D
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style("darkgrid")
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [303…
```
# Load the data
facies_vectors = pd.read_excel('Exam2_facies.xlsx', sheet_name='facies_vecto
well_nolan = pd.read_excel('Exam2_facies.xlsx', sheet_name='Well Nolan')

# Interpolate missing data
facies_vectors.interpolate(method='cubic', inplace=True)

# Select only rows with Well Name 'NOLAN'
well_nolan = well_nolan[well_nolan['Well Name'] != 'BAD']
```

In [304…
```
#Check facies_vectors sheet
facies_vectors.head()
print(facies_vectors.dtypes)
```

```
Well Name     object
Depth        float64
GR           float64
ILD_log10    float64
DeltaPHI     float64
PHIND        float64
PE           float64
NM_M           int64
RELPOS       float64
Facies         int64
dtype: object
```

In [305…
```
##Check well nolan sheet
well_nolan.head()
```

Out[305]:

	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS
0	NOLAN	2853.5	106.813	0.533	9.339	15.222	3.500	1	1.000
1	NOLAN	2854.0	100.938	0.542	8.857	15.313	3.416	1	0.977
2	NOLAN	2854.5	94.375	0.553	7.097	14.583	3.195	1	0.955

	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS
3	NOLAN	2855.0	89.813	0.554	7.081	14.110	2.963	1	0.932
4	NOLAN	2855.5	91.563	0.560	6.733	13.189	2.979	1	0.909

In [306…
```
#check any missing value
facies_vectors.isna().any()
```

Out[306]:
```
Well Name     False
Depth         False
GR            False
ILD_log10     False
DeltaPHI      False
PHIND         False
PE            False
NM_M          False
RELPOS        False
Facies        False
dtype: bool
```

In [307…
```
well_nolan.isna().any()
```

Out[307]:
```
Well Name     False
Depth         False
GR            False
ILD_log10     False
DeltaPHI      False
PHIND         False
PE            False
NM_M          False
RELPOS        False
dtype: bool
```

In [308…
```
# Create dummy variables for the facies
facies_vectors = pd.concat([facies_vectors, pd.get_dummies(facies_vectors['F

# Split the data into X and y
X = facies_vectors[['Depth','GR', 'ILD_log10', 'DeltaPHI', 'PHIND', 'PE', 'N
y = facies_vectors[['Facies_1', 'Facies_2', 'Facies_3', 'Facies_4', 'Facies_

#Split "Well Nolan" dataset
X_nolan = well_nolan[['Depth','GR', 'ILD_log10', 'DeltaPHI', 'PHIND', 'PE',

# Split the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Scale the data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_nolan_scaled = scaler.transform(X_nolan)
```

Refine the most important hyperparameters for 2 models

In [309…

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to tune
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance']
}

# Create a grid search object
knn_grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

# Fit the grid search object to the training data
knn_grid.fit(X_train_scaled, y_train)

# Print the best parameters and score
print("Best parameters:", knn_grid.best_params_)
print("Best score:", knn_grid.best_score_)
```

```
Best parameters: {'n_neighbors': 3, 'weights': 'distance'}
Best score: 0.8369888934594817
```

In [310…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to tune
param_grid = {
    'n_estimators': [100, 300, 500],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 4, 6]
}

# Create a grid search object
rfc_grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid,

# Fit the grid search object to the training data
rfc_grid.fit(X_train_scaled, y_train)

# Print the best parameters and score
print("Best parameters:", rfc_grid.best_params_)
print("Best score:", rfc_grid.best_score_)
```

```
Best parameters: {'max_depth': 7, 'min_samples_split': 2, 'n_estimators': 50
0}
Best score: 0.6820649938296997
```

In [311…

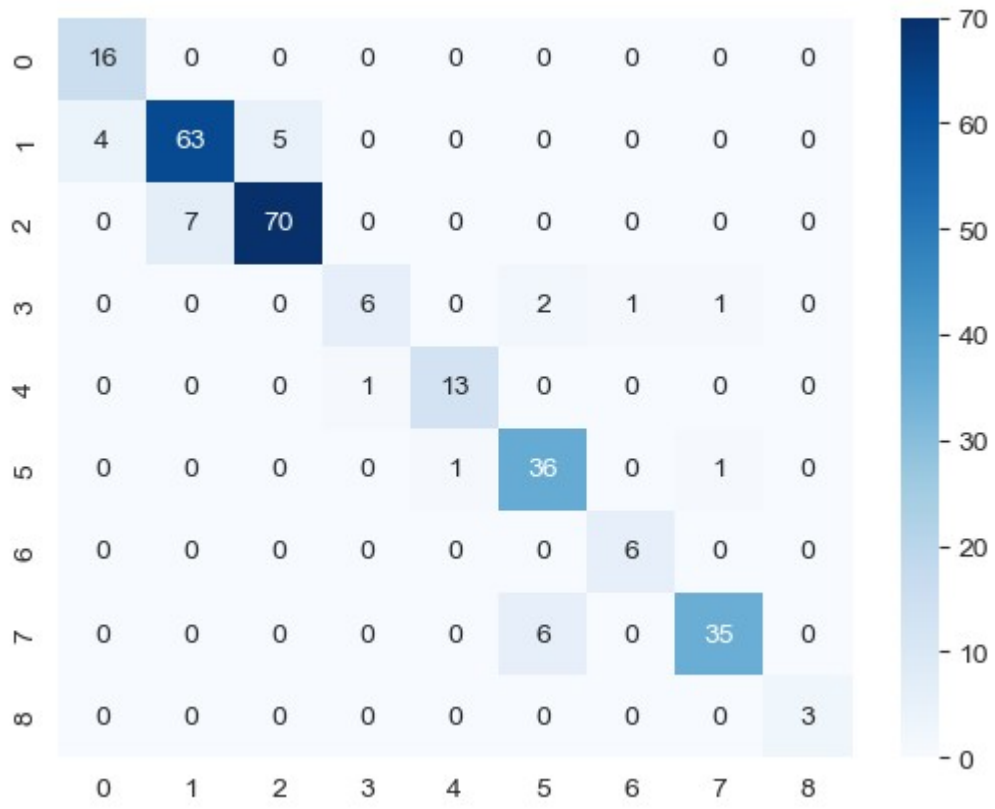
```
#train and evaluate a KNN classifier
knn = KNeighborsClassifier(n_neighbors=3, weights='distance')
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
print('KNN Classifier')
print(classification_report(y_test, y_pred_knn))
# Get confusion matrix
cm_knn = confusion_matrix(y_test.values.argmax(axis=1), y_pred_knn.argmax(ax

# Create heatmap
sns.heatmap(cm_knn, annot=True, cmap='Blues')
```

```
KNN Classifier
              precision    recall  f1-score   support
```

0	0.94	0.94	0.94	16
1	0.90	0.88	0.89	72
2	0.93	0.91	0.92	77
3	0.86	0.60	0.71	10
4	0.93	0.93	0.93	14
5	0.82	0.95	0.88	38
6	0.86	1.00	0.92	6
7	0.95	0.85	0.90	41
8	1.00	1.00	1.00	3
micro avg	0.90	0.89	0.90	277
macro avg	0.91	0.89	0.90	277
weighted avg	0.91	0.89	0.90	277
samples avg	0.89	0.89	0.89	277

```
C:\Users\nguye\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in samples with no predicted labels. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[311]: <AxesSubplot:>



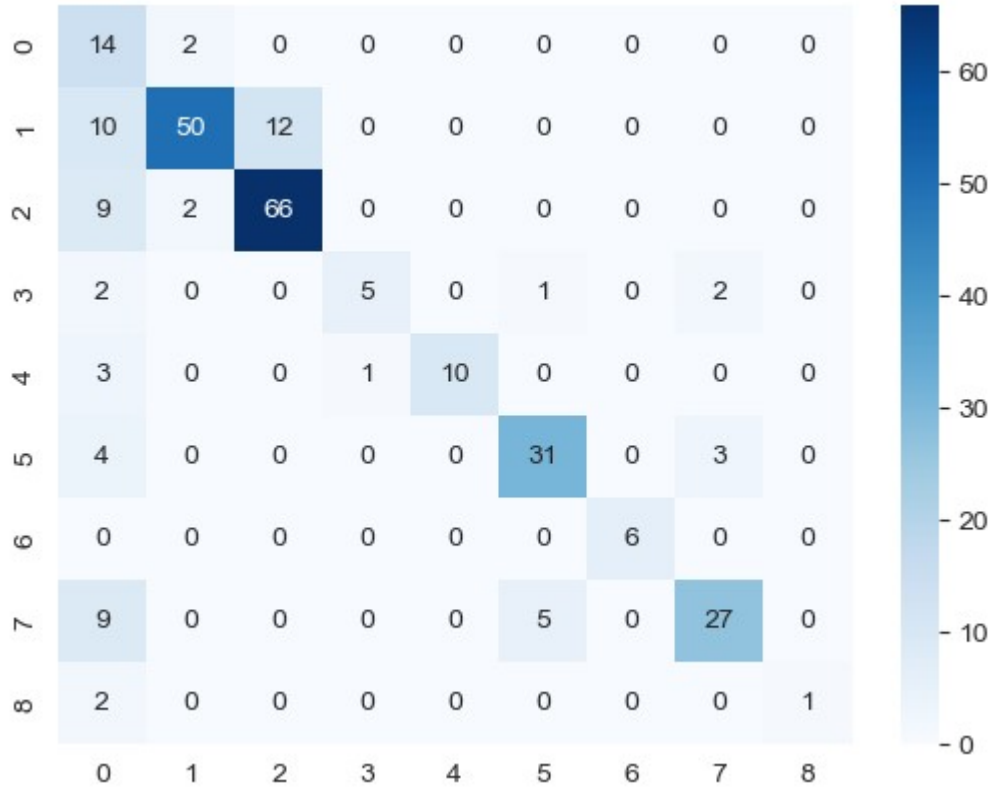
In [312…

```
#train and evaluate a RandomForest classifier
rfc = RandomForestClassifier(n_estimators=500, max_depth=7, min_samples_spli
rfc.fit(X_train_scaled, y_train)
y_pred_rfc = rfc.predict(X_test_scaled)
print('Random Forest Classifier')
print(classification_report(y_test, y_pred_rfc))
cm_rfc = confusion_matrix(y_test.values.argmax(axis=1), y_pred_rfc.argmax(ax

# Create heatmap
sns.heatmap(cm_rfc, annot=True, cmap='Blues')
```

```
Random Forest Classifier
              precision    recall  f1-score   support

           0       1.00      0.69      0.81        16
           1       0.93      0.69      0.79        72
           2       0.85      0.86      0.85        77
           3       0.83      0.50      0.62        10
           4       1.00      0.71      0.83        14
           5       0.84      0.82      0.83        38
           6       1.00      1.00      1.00         6
           7       0.84      0.66      0.74        41
           8       1.00      0.33      0.50         3

   micro avg       0.88      0.75      0.81       277
   macro avg       0.92      0.70      0.78       277
weighted avg       0.89      0.75      0.80       277
 samples avg       0.75      0.75      0.75       277

C:\Users\nguye\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in samples with no predicted labels. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Out[312]:

```
<AxesSubplot:>
```



In [313…

```
# Make predictions using the trained models
y_pred_nolan_knn = knn.predict(X_nolan_scaled)
y_pred_nolan_rfc = rfc.predict(X_nolan_scaled)

# Convert the predicted facies to a dataframe
df_pred_nolan_knn = pd.DataFrame(y_pred_nolan_knn, columns=['Facies_1_knn',
df_pred_nolan_rfc = pd.DataFrame(y_pred_nolan_rfc, columns=['Facies_1_rfc',

# Add the predicted facies to the Well Nolan dataset
well_nolan = pd.concat([well_nolan, df_pred_nolan_knn, df_pred_nolan_rfc], a
well_nolan
```

Out[313]:

	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS	Facies_1_knn
0	NOLAN	2853.5	106.813	0.533	9.339	15.222	3.500	1.0	1.000	0.0
1	NOLAN	2854.0	100.938	0.542	8.857	15.313	3.416	1.0	0.977	0.0
2	NOLAN	2854.5	94.375	0.553	7.097	14.583	3.195	1.0	0.955	0.0
3	NOLAN	2855.0	89.813	0.554	7.081	14.110	2.963	1.0	0.932	0.0
4	NOLAN	2855.5	91.563	0.560	6.733	13.189	2.979	1.0	0.909	0.0
...	...	...	...	...	...	...	...	...	...	...
157	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
158	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
159	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
160	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0

	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS	Facies_1_knn
161	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0

439 rows × 27 columns

In [314…

```
# Filter rows where any of the 9 Facies have a value of 1 for either the KNN
facies_1_to_9 = well_nolan[(well_nolan.loc[:, 'Facies_1_knn':'Facies_9_rfc']
facies_1_to_9
```

Out[314]:

	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS	Facies_1_knn
0	NOLAN	2853.5	106.813	0.533	9.339	15.222	3.500	1.0	1.000	0.0
1	NOLAN	2854.0	100.938	0.542	8.857	15.313	3.416	1.0	0.977	0.0
2	NOLAN	2854.5	94.375	0.553	7.097	14.583	3.195	1.0	0.955	0.0
3	NOLAN	2855.0	89.813	0.554	7.081	14.110	2.963	1.0	0.932	0.0
4	NOLAN	2855.5	91.563	0.560	6.733	13.189	2.979	1.0	0.909	0.0
…	…	…	…	…	…	…	…	…	…	…
157	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
158	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
159	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
160	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
161	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0

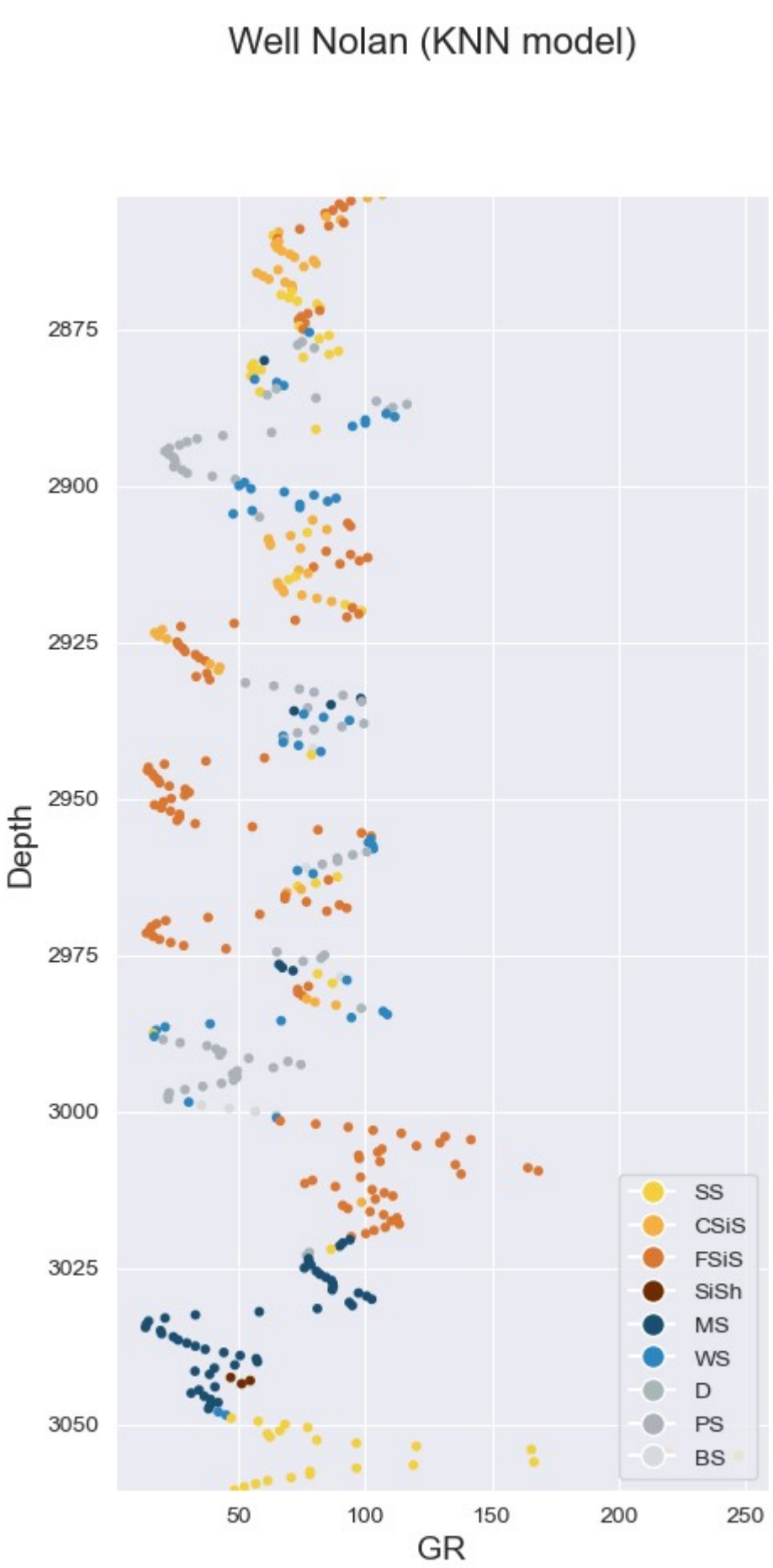
396 rows × 27 columns

In [315…

```
#Create a function to plot the predicted facies
facies_labels_names = ['SS', 'CSiS', 'FSiS', 'SiSh', 'MS', 'WS', 'D', 'PS',

def plot_facies(well_data, well_name, model_name):
    facies_cols = [f'Facies_{i}_{model_name.lower()}' for i in range(1, 10)]
    facies_labels = np.argmax(well_data[facies_cols].values, axis=1) + 1

    fig, ax = plt.subplots(figsize=(5, 10))
    fig.suptitle(f"Well {well_name} ({model_name} model)", fontsize=16)
    ax.scatter(well_data['GR'], well_data['Depth'], c=[facies_colors[i-1] fo
    ax.set_xlabel('GR', fontsize=14)
    ax.set_ylim(well_data['Depth'].max(), well_data['Depth'].min())
    ax.set_ylabel('Depth', fontsize=14)

    #handles = [Line2D([0], [0], marker='o', color='w', label=f'Facies {i}',
    handles = [Line2D([0], [0], marker='o', color='w', label=facies_labels_n

    ax.legend(handles=handles, loc='lower right')
```

In [316…

```
# Create separate plots for the KNN and RFC models
plot_facies(well_nolan, 'Nolan', 'KNN')
plot_facies(well_nolan, 'Nolan', 'RFC')
```


Well Nolan (KNN model)
Depth
2875
2900
2925
2950
2975
3000
3025
3050
50
100
150
200
250
GR
SS
CSiS
FSiS
SiSh
MS
WS
D
PS
BS


Well Nolan (RFC model)
Depth
2875
2900
2925
2950
2975
3000
3025
3050
50
100
150
200
250
GR
SS
CSiS
FSiS
SiSh
MS
WS
D
PS
BS

In [ ]:

Part 2

In [93]:
```
#Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [82]:
```
# Load the data from the "well 14" and "well 15" sheets of the VolveData.xls
df = pd.read_excel("VolveData.xlsx", sheet_name=["well 14", "well 15"])
```

In [83]:
```
# Concatenate the data from the two sheets
df = pd.concat(df)
df.head()
```

Out[83]:

		Depth	Well	GR	RT	RHOB	NPHI	Facies
well 14	**0**	3178.5	14	50.2190	0.5888	2.3296	0.3657	SH
	1	3179.0	14	47.2468	0.7768	2.3170	0.3776	UN
	2	3179.5	14	49.5247	1.0707	2.2960	0.5390	SH
	3	3180.0	14	44.9124	1.4460	2.2514	0.5482	UN
	4	3180.5	14	47.0048	0.9542	2.2733	0.5076	UN

In [84]:
```
#check any missing value
df.isna().any()
```

Out[84]:
```
Depth     False
Well      False
GR        False
RT        False
RHOB      False
NPHI      False
Facies    False
dtype: bool
```

In [85]:
```
# Split the data into X and y
X = df[['Depth', 'GR', 'RT', 'RHOB', 'NPHI']]
y = df['Facies']
```

In [86]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

In [87]:
```
# Apply the MinMaxScaler to the features
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [89]:
```
# Train a logistic regression model
log_reg = LogisticRegression(C=100,max_iter=1000)
log_reg.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_log_reg = log_reg.predict(X_test)
print('Logistic Regression')
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_log_reg))
print('Classification Report:')
print(classification_report(y_test, y_pred_log_reg))
print('Accuracy Score:', accuracy_score(y_test, y_pred_log_reg))
```

```
Logistic Regression
Confusion Matrix:
[[235   1  35   0]
 [  6 158   3   0]
 [ 32   1 151   0]
 [  4  10  13   0]]
Classification Report:
              precision    recall  f1-score   support

          CB       0.85      0.87      0.86       271
          SH       0.93      0.95      0.94       167
          SS       0.75      0.82      0.78       184
          UN       0.00      0.00      0.00        27

    accuracy                           0.84       649
   macro avg       0.63      0.66      0.64       649
weighted avg       0.81      0.84      0.82       649

Accuracy Score: 0.8382126348228043
```

```
C:\Users\nguye\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\nguye\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\nguye\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [90]:
```
# Train a K-Nearest Neighbor model
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_knn = knn.predict(X_test)
print('\nK-Nearest Neighbor')
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_knn))
print('Classification Report:')
print(classification_report(y_test, y_pred_knn))
print('Accuracy Score:', accuracy_score(y_test, y_pred_knn))
```

```
K-Nearest Neighbor
Confusion Matrix:
[[244   1  25   1]
 [  8 153   1   5]
 [ 20   4 157   3]
 [  1  10   6  10]]
Classification Report:
              precision    recall  f1-score   support

          CB       0.89      0.90      0.90       271
          SH       0.91      0.92      0.91       167
          SS       0.83      0.85      0.84       184
          UN       0.53      0.37      0.43        27

    accuracy                           0.87       649
   macro avg       0.79      0.76      0.77       649
weighted avg       0.86      0.87      0.87       649

Accuracy Score: 0.8690292758089369
```

In [91]:
```
# Train a support vector machine model
svm = SVC(kernel='rbf',C=100, gamma=1)
svm.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_svm = svm.predict(X_test)
print('\nSupport Vector Machine')
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_svm))
print('Classification Report:')
print(classification_report(y_test, y_pred_svm))
print('Accuracy Score:', accuracy_score(y_test, y_pred_svm))
```

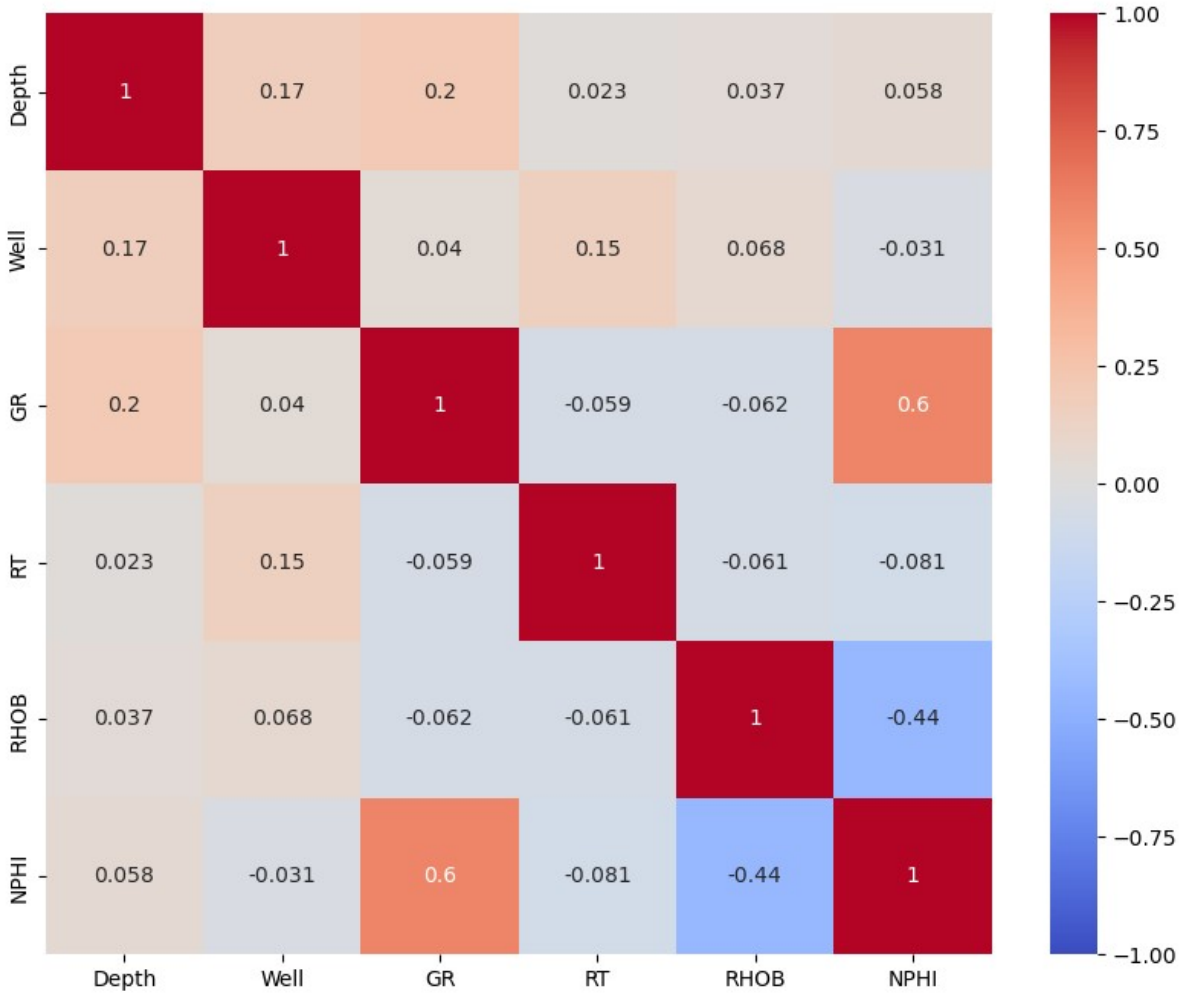
```
Support Vector Machine
Confusion Matrix:
[[244   0  27   0]
 [  6 157   2   2]
 [ 25   1 156   2]
 [  2   7  10   8]]
Classification Report:
              precision    recall  f1-score   support

          CB       0.88      0.90      0.89       271
          SH       0.95      0.94      0.95       167
          SS       0.80      0.85      0.82       184
          UN       0.67      0.30      0.41        27
```

```
    accuracy                           0.87       649
   macro avg       0.82      0.75      0.77       649
weighted avg       0.87      0.87      0.87       649

Accuracy Score: 0.8705701078582434
```

In [92]:
```
from sklearn.ensemble import RandomForestClassifier

# Train a random forest model
rf = RandomForestClassifier(n_estimators=500, random_state=42,max_depth=None
rf.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred_rf = rf.predict(X_test)
print('\nRandom Forest')
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_rf))
print('Classification Report:')
print(classification_report(y_test, y_pred_rf))
print('Accuracy Score:', accuracy_score(y_test, y_pred_rf))
```

```
Random Forest
Confusion Matrix:
[[246   1  23   1]
 [  0 161   1   5]
 [ 19   1 163   1]
 [  1   8   7  11]]
Classification Report:
              precision    recall  f1-score   support

          CB       0.92      0.91      0.92       271
          SH       0.94      0.96      0.95       167
          SS       0.84      0.89      0.86       184
          UN       0.61      0.41      0.49        27

    accuracy                           0.90       649
   macro avg       0.83      0.79      0.81       649
weighted avg       0.89      0.90      0.89       649

Accuracy Score: 0.8952234206471494
```

In [94]:
```
#Check features correlation

# Calculate the correlations
correlations = df.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.show()
```


Depth
Well
GR
RT
RHOB
NPHI
1 0.17 0.2 0.023 0.037 0.058
0.17 1 0.04 0.15 0.068 -0.031
0.2 0.04 1 -0.059 -0.062 0.6
0.023 0.15 -0.059 1 -0.061 -0.081
0.037 0.068 -0.062 -0.061 1 -0.44
0.058 -0.031 0.6 -0.081 -0.44 1
Depth
Well
GR
RT
RHOB
NPHI
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [95]:
```
#Create a function to check the best combination features
from sklearn.feature_selection import SelectKBest, mutual_info_classif

def select_features(model, X, y, k):
    selector = SelectKBest(score_func=mutual_info_classif, k=k)
    selector.fit(X, y)
    return selector

best_features = {}
k = 3 # Select top k features

for model_name, model in [('Logistic Regression', log_reg),
                          ('K-Nearest Neighbors', knn),
                          ('Support Vector Machine', svm),
                          ('Random Forest', rf)]:
    selector = select_features(model, X_train, y_train, k)
    best_features[model_name] = list(X.columns[selector.get_support()])

print("Best features for each model:")
for model_name, features in best_features.items():
    print(f"{model_name}: {features}")
```

```
Best features for each model:
Logistic Regression: ['Depth', 'GR', 'NPHI']
K-Nearest Neighbors: ['Depth', 'GR', 'NPHI']
Support Vector Machine: ['Depth', 'GR', 'NPHI']
Random Forest: ['Depth', 'GR', 'NPHI']
```

In [96]:
```
#Find values of hyperparameters ensure the models ensure high generalization
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter search space for each classifier
param_grid_log_reg = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
param_grid_knn = {'n_neighbors': list(range(1, 21))}
param_grid_svm = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001]}
param_grid_rf = {'n_estimators': [100, 200, 500], 'max_depth': [None, 10, 20

# Create a dictionary to store the best hyperparameters for each classifier
best_hyperparameters = {}

# Perform grid search for each classifier
for model_name, model, param_grid in [('Logistic Regression', log_reg, param
                                       ('K-Nearest Neighbors', knn, param_gri
                                       ('Support Vector Machine', svm, param_
                                       ('Random Forest', rf, param_grid_rf)]:
    grid_search = GridSearchCV(model, param_grid, scoring='accuracy', cv=5)
    grid_search.fit(X_train, y_train)
    best_hyperparameters[model_name] = grid_search.best_params_

print("Best hyperparameters for each model:")
for model_name, hyperparameters in best_hyperparameters.items():
    print(f"{model_name}: {hyperparameters}")
```

```
Best hyperparameters for each model:
Logistic Regression: {'C': 100}
K-Nearest Neighbors: {'n_neighbors': 8}
Support Vector Machine: {'C': 100, 'gamma': 1}
Random Forest: {'max_depth': None, 'min_samples_split': 10, 'n_estimators':
500}
```

Note: Based on the evaluation metrics above, RandomForest the best model

In [98]:
```
# Load the data from the "well 13" sheets of the VolveData.xlsx file
well_13_data = pd.read_excel("VolveData.xlsx", sheet_name="well 13")
well_13_data.head()
```

Out[98]:

	Depth	Well	GR	RT	RHOB	NPHI
0	4175.5	13	20.6032	4.1812	2.6117	0.0770
1	4176.0	13	21.4990	4.5516	2.6131	0.0798
2	4176.5	13	22.4472	4.4804	2.6334	0.0801
3	4177.0	13	29.6713	4.3859	2.6328	0.1005
4	4177.5	13	34.7014	4.8566	2.6183	0.1001

In [99]:
```
# Deploy the model on Well 13

well_13_X = well_13_data[['Depth', 'GR', 'RT', 'RHOB', 'NPHI']]
well_13_X_scaled = scaler.transform(well_13_X)
well_13_predictions = rf.predict(well_13_X_scaled)
```

In [100…
```
# Export the predictions for Well 13 along with depth and GR log to a separa
well_13_export_data = well_13_data[['Depth', 'GR']].copy()
well_13_export_data['Facies_Predictions'] = well_13_predictions
```

In [101…
```
well_13_export_data.to_excel("Well_13_Predictions.xlsx", index=False)
```

In [ ]:

Part 3

In [47]:
```
#Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
from sklearn.ensemble import RandomForestRegressor
```

In [48]:
```
# Load the data
df = pd.read_csv('ROP_DataSet.csv')
```

In [49]:
```
#Perform Summary Statistics
summary_stats = df.describe()
print(summary_stats)
```

```
         Hole Depth    Hook Load   Rotary RPM  Rotary Torque  Weight on Bit
\
count   7935.000000  7931.000000  7935.000000    7935.000000    7935.000000
mean   10484.787524   129.673106    65.835791      11.460365      19.827309
std     2306.885010     7.720494    27.062600       3.386803       5.611646
min   -14454.000000   107.200000  -999.000000       2.701000       0.000000
25%     8499.500000   123.800000    49.000000       9.096000      16.300000
50%    10487.000000   129.500000    70.000000      11.373000      20.400000
75%    12469.500000   134.400000    90.000000      14.198000      23.900000
max    14454.000000   156.400000   101.000000      20.050000      39.400000

       Differential Pressure  Gamma at Bit  Rate Of Penetration
count            7935.000000   7935.000000          7935.000000
mean              520.270573    211.782641           143.112973
std               142.475611     81.531795            55.736881
min                 2.900000     54.120000             1.610000
25%               429.400000    148.240000           100.160000
50%               565.900000    204.710000           161.160000
75%               627.700000    235.290000           185.240000
max               783.300000    600.000000           259.290000
```

There seems to be some negative values in the Hole Depth and Rotary RPM columns, which is not physically possible. It could be due to data entry errors or some other issue in data collection.

The Differential Pressure column has a relatively high standard deviation compared to its mean, which could suggest that there are some extreme values or outliers present in the data.

The Gamma at Bit column also has a wide range of values, with a minimum of 54.12 and a maximum of 600. This suggests that there may be outliers present in this column as well.

In [50]:
```
#Unique values of column Rotary RPM
unique_rpm = df['Rotary RPM'].unique()
print(unique_rpm)
```

```
[  29 -999   30   40   32   39   50   49   44   51   58    9   10   24
   25   14   15  100  101   99   60   41   79   80   86   81   52   53
   31   45   69   70   71   35   54   65   90   91   89]
```

In [51]:
```
#Check missing data in Depth
missing_data_mask = df.isnull().any(axis=1)
missing_depths = df.loc[missing_data_mask, 'Hole Depth']

print("Depths with missing data:")
print(missing_depths)
```

```
Depths with missing data:
10       6534
367      6891
413      6937
7898    14419
Name: Hole Depth, dtype: int64
```

In [52]:
```
#Check for other missing data
missing_data = df.isnull().sum()
print(missing_data)
```

```
Hole Depth              0
Hook Load               4
Rotary RPM              0
Rotary Torque           0
Weight on Bit           0
Differential Pressure   0
Gamma at Bit            0
Rate Of Penetration     0
dtype: int64
```

There are only a few missing values in the Hook Load olumn, so dropping the affected rows is a viable option

In [53]:
```
#Replace [np.inf, -np.inf, np.inf, -999., -999, 999, '', " ", 'inf', 'NaN']
values_to_replace = [np.inf, -np.inf, np.inf, -999., -999, 999, '', " ", 'in
df.replace(values_to_replace, np.nan, inplace=True)
```

In [54]:
```
#Dropping null rows
df.dropna(inplace=True)
```

In [55]:
```
#Recheck dataset
missing_data = df.isnull().sum()
print(missing_data)
```

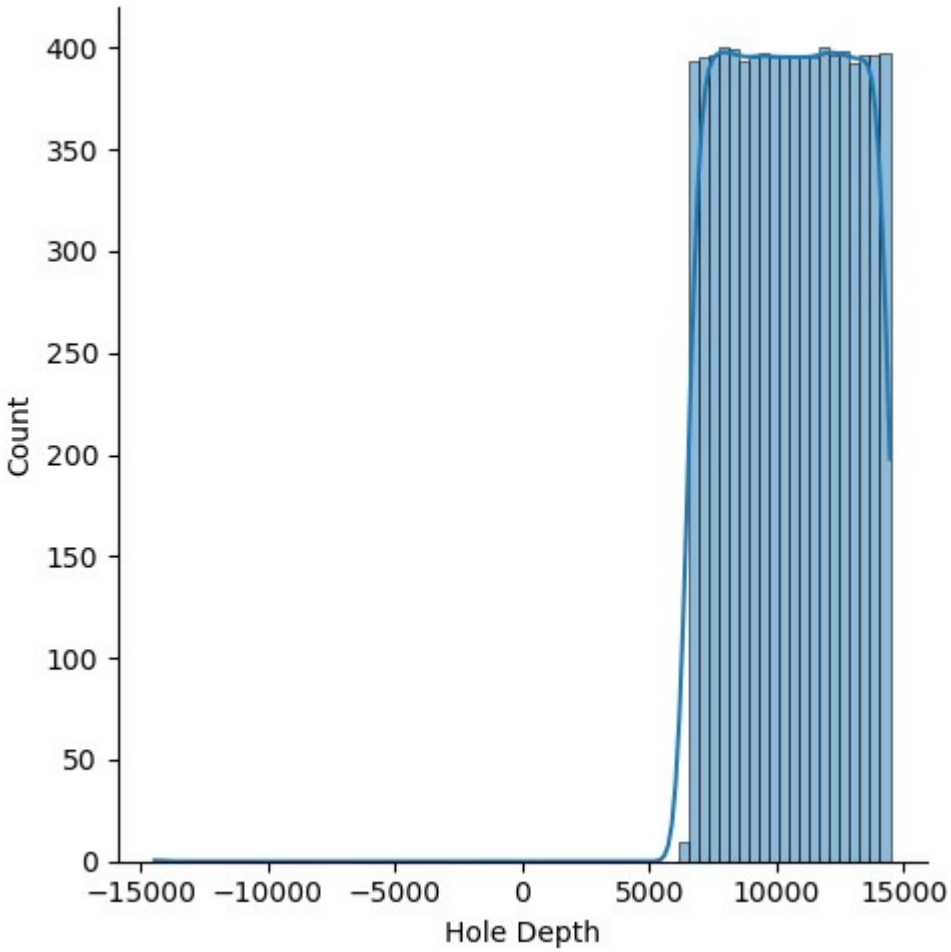
```
Hole Depth              0
Hook Load               0
Rotary RPM              0
Rotary Torque           0
Weight on Bit           0
Differential Pressure   0
Gamma at Bit            0
Rate Of Penetration     0
```

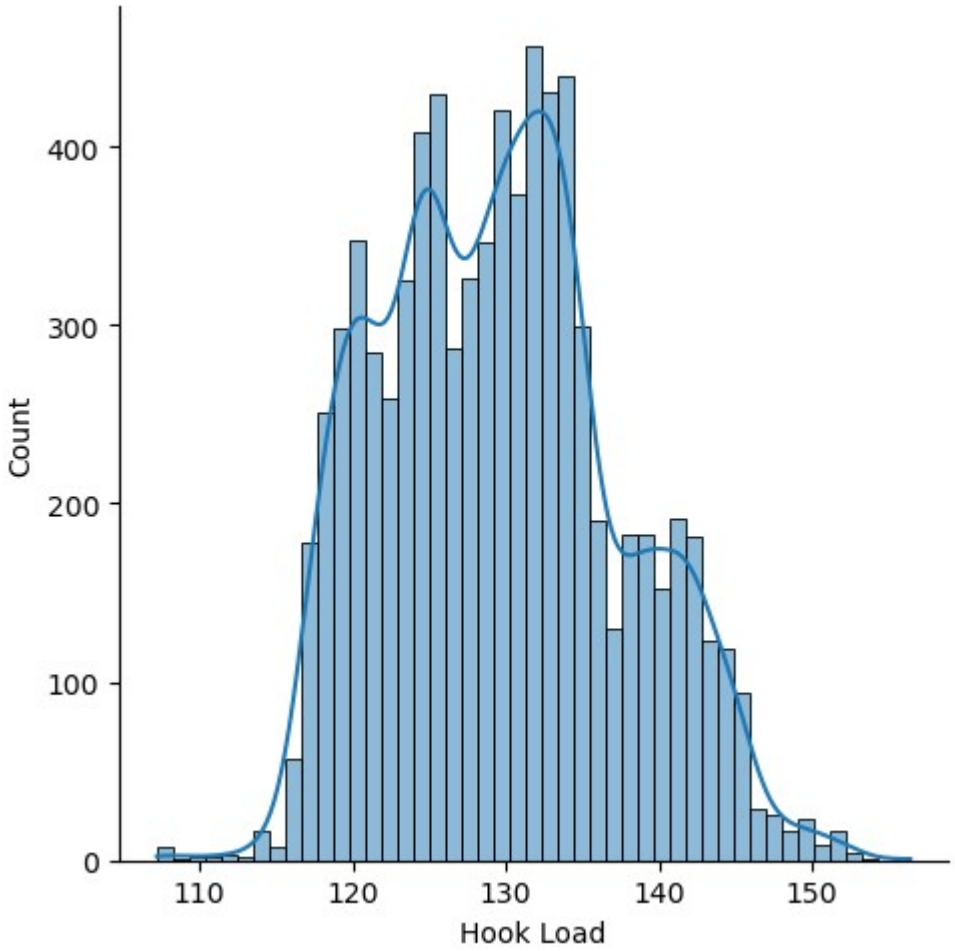
```
dtype: int64
```

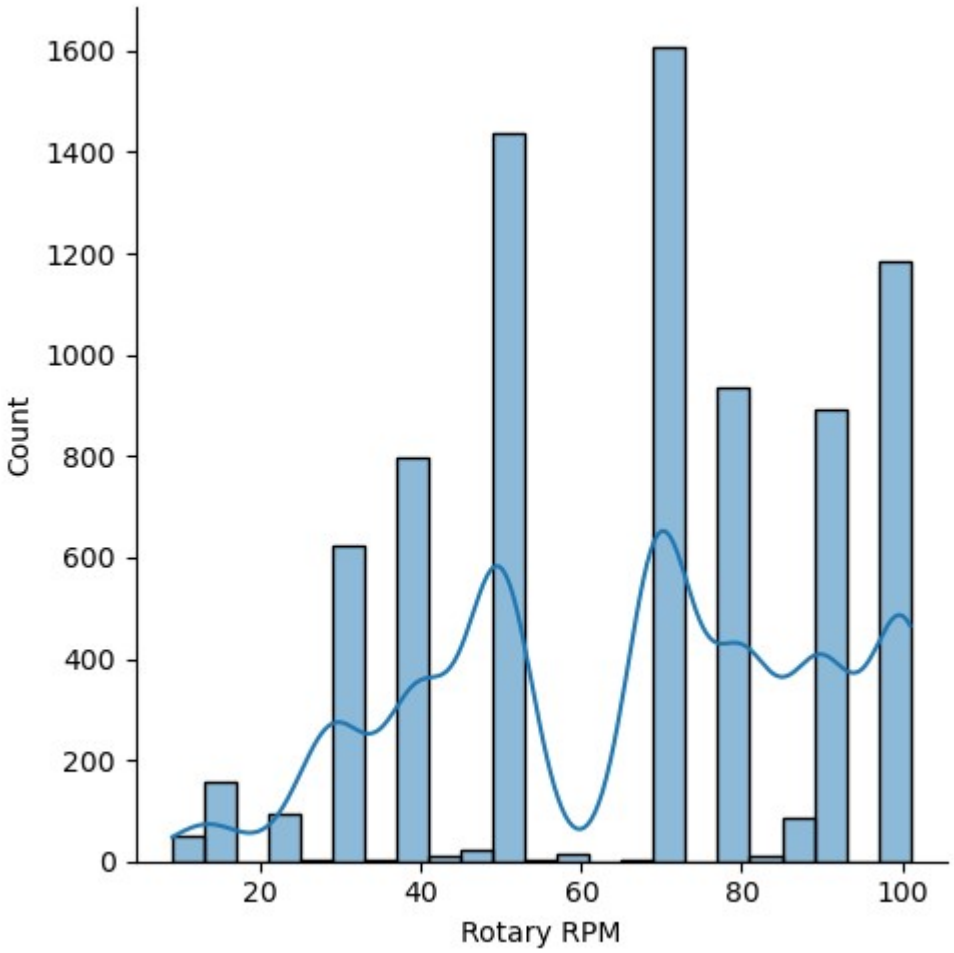
In [56]:
```
#Check data dimensions
print("Cleaned dataset dimensions:", df.shape)
```

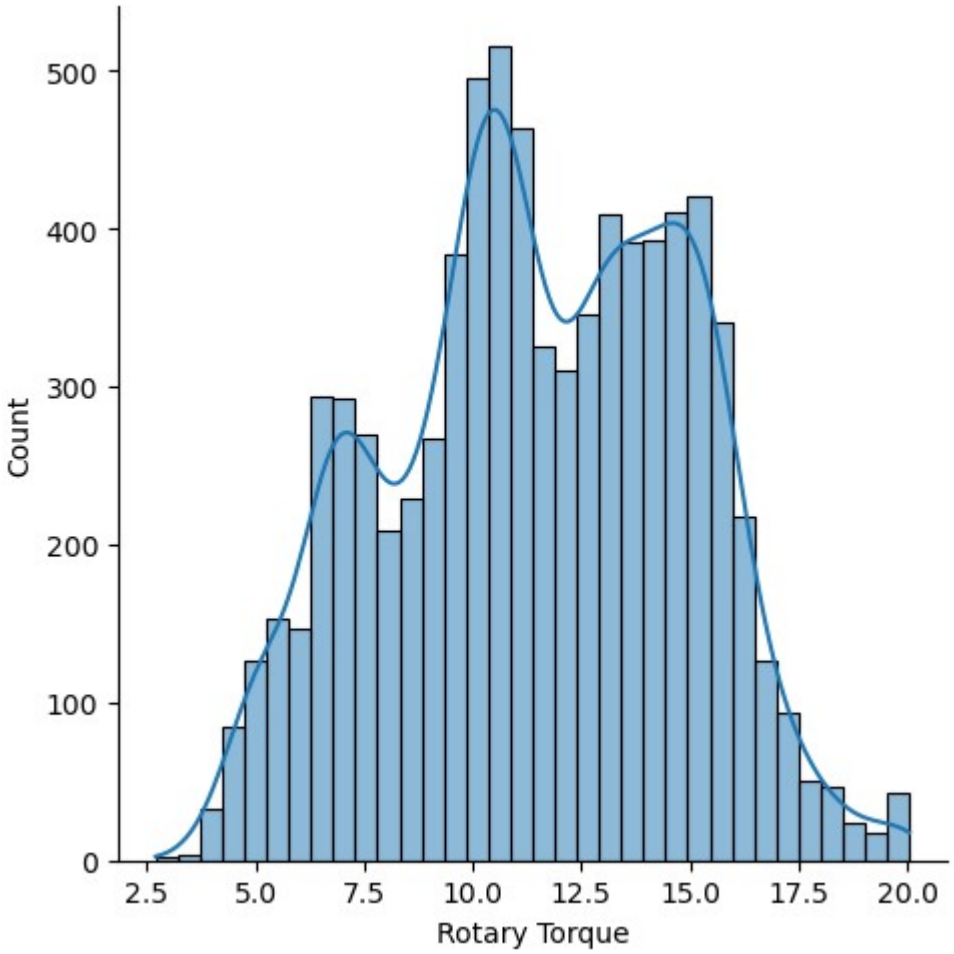
```
Cleaned dataset dimensions: (7930, 8)
```

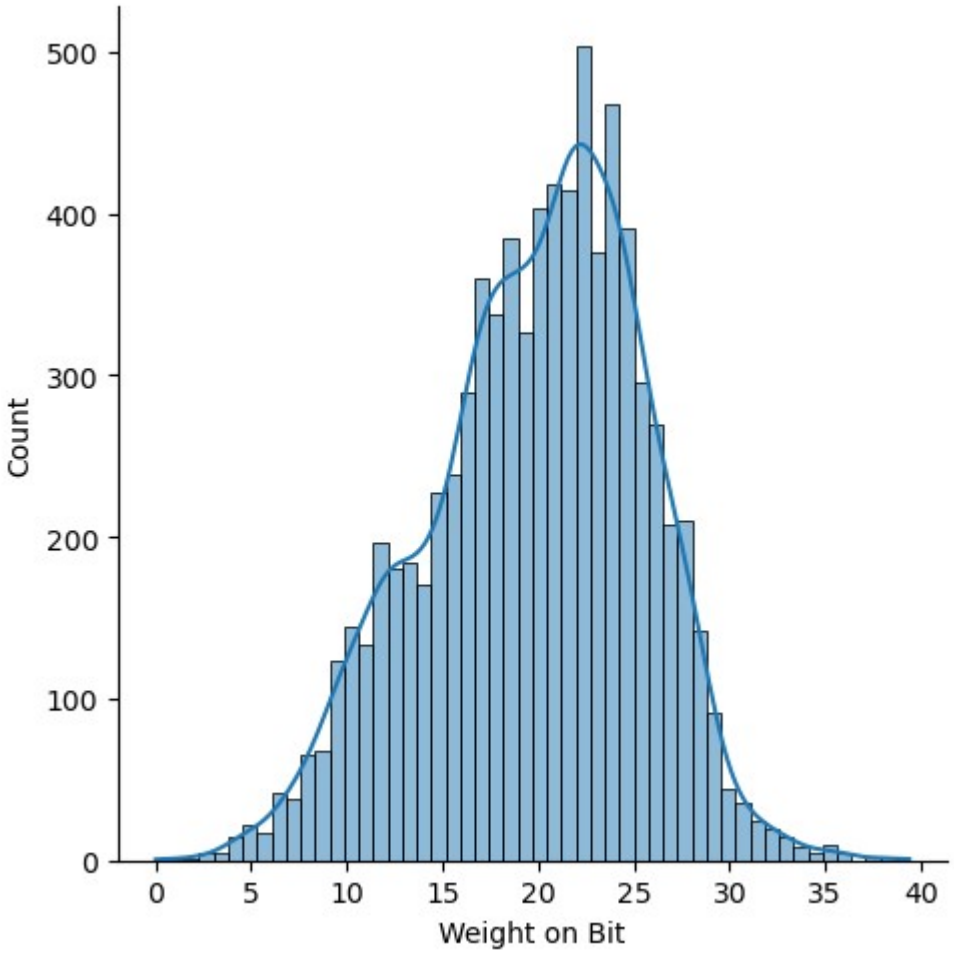
In [57]:
```
#Distribution plots
import matplotlib.pyplot as plt
import seaborn as sns

sns.displot(data=df, x='Hole Depth', kde=True)
plt.show()

sns.displot(data=df, x='Hook Load', kde=True)
plt.show()

sns.displot(data=df, x='Rotary RPM', kde=True)
plt.show()

sns.displot(data=df, x='Rotary Torque', kde=True)
plt.show()

sns.displot(data=df, x='Weight on Bit', kde=True)
plt.show()

sns.displot(data=df, x='Differential Pressure', kde=True)
plt.show()

sns.displot(data=df, x='Gamma at Bit', kde=True)
plt.show()

sns.displot(data=df, x='Rate Of Penetration', kde=True)
plt.show()
```

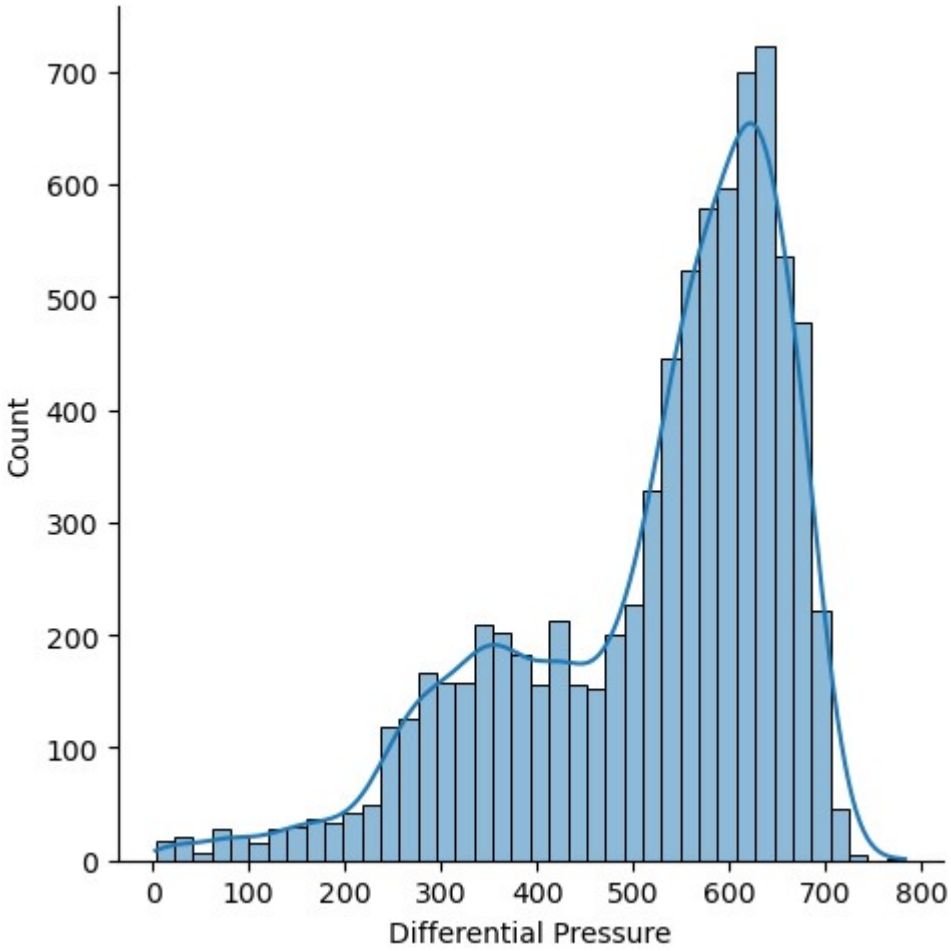

400
350
300
250
200
150
100
50
0
Count
−15000
−10000
−5000
0
5000
10000
15000
Hole Depth

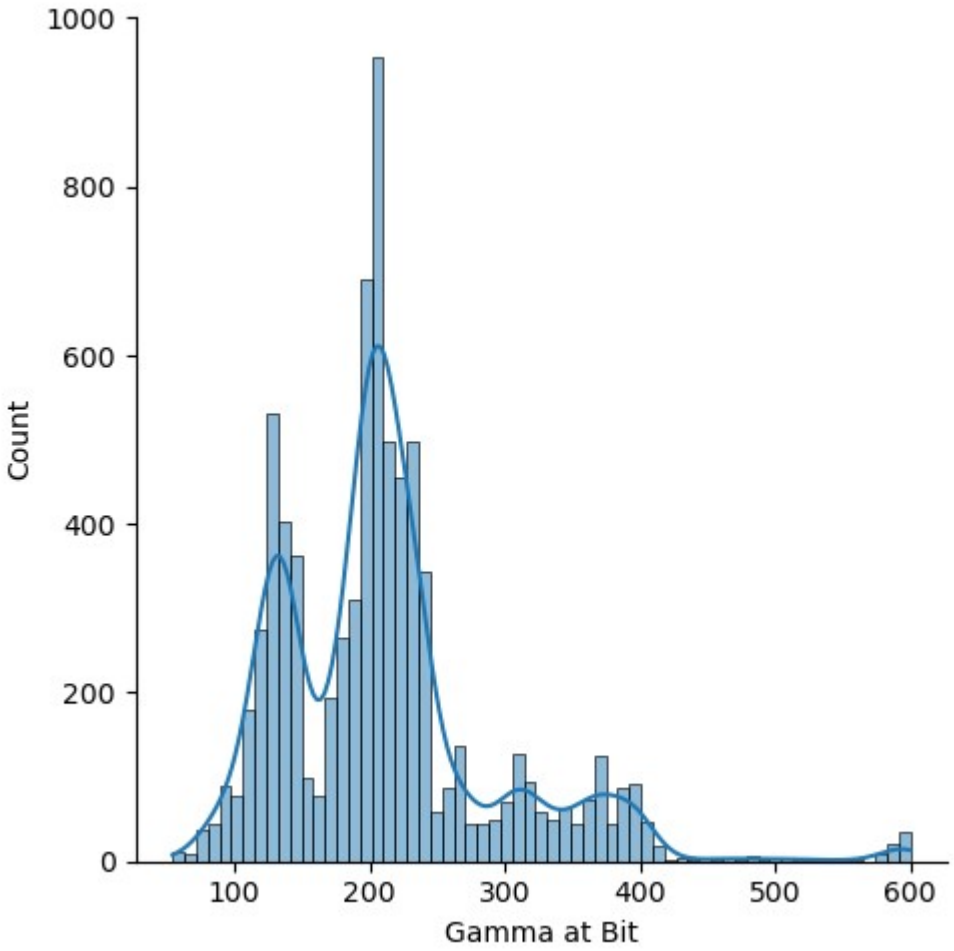

400
300
200
100
0
Count
110
120
130
140
150
Hook Load

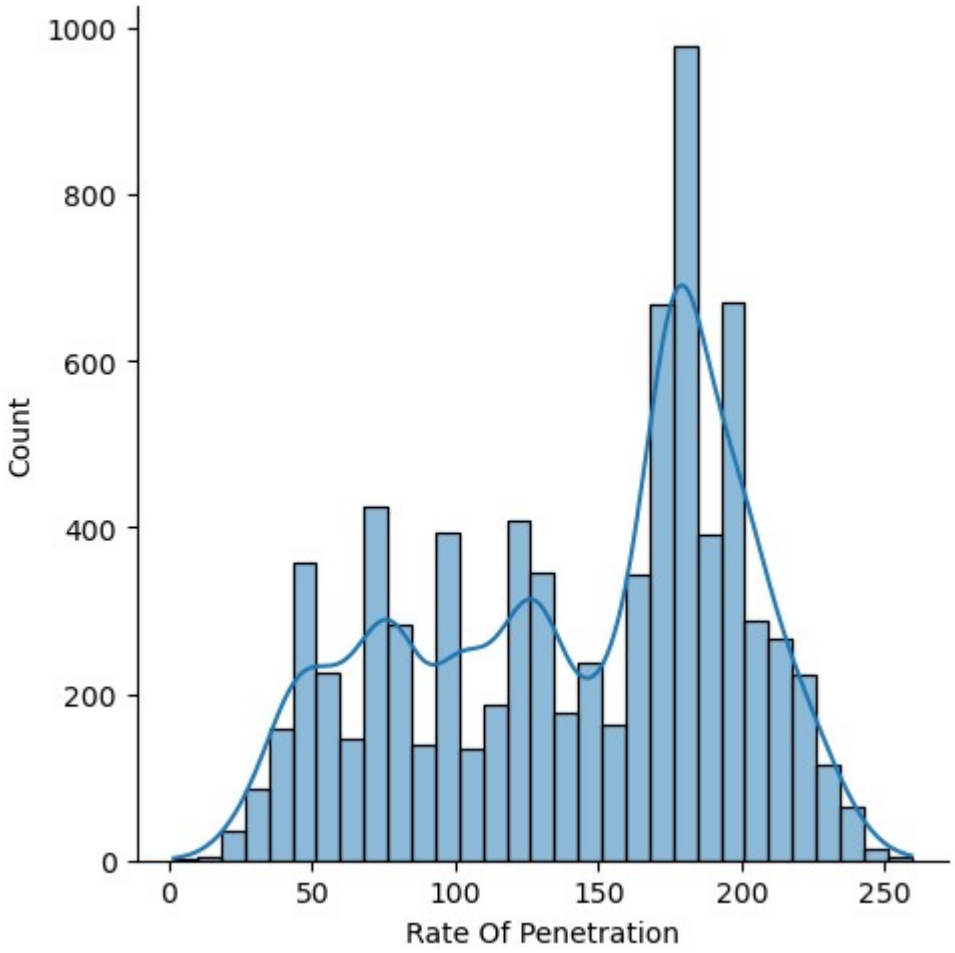

1600
1400
1200
1000
800
600
400
200
0
Count
20
40
60
80
100
Rotary RPM


500
400
300
200
100
0
Count
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Rotary Torque


500
400
300
200
100
0
Count
0
5
10
15
20
25
30
35
40
Weight on Bit

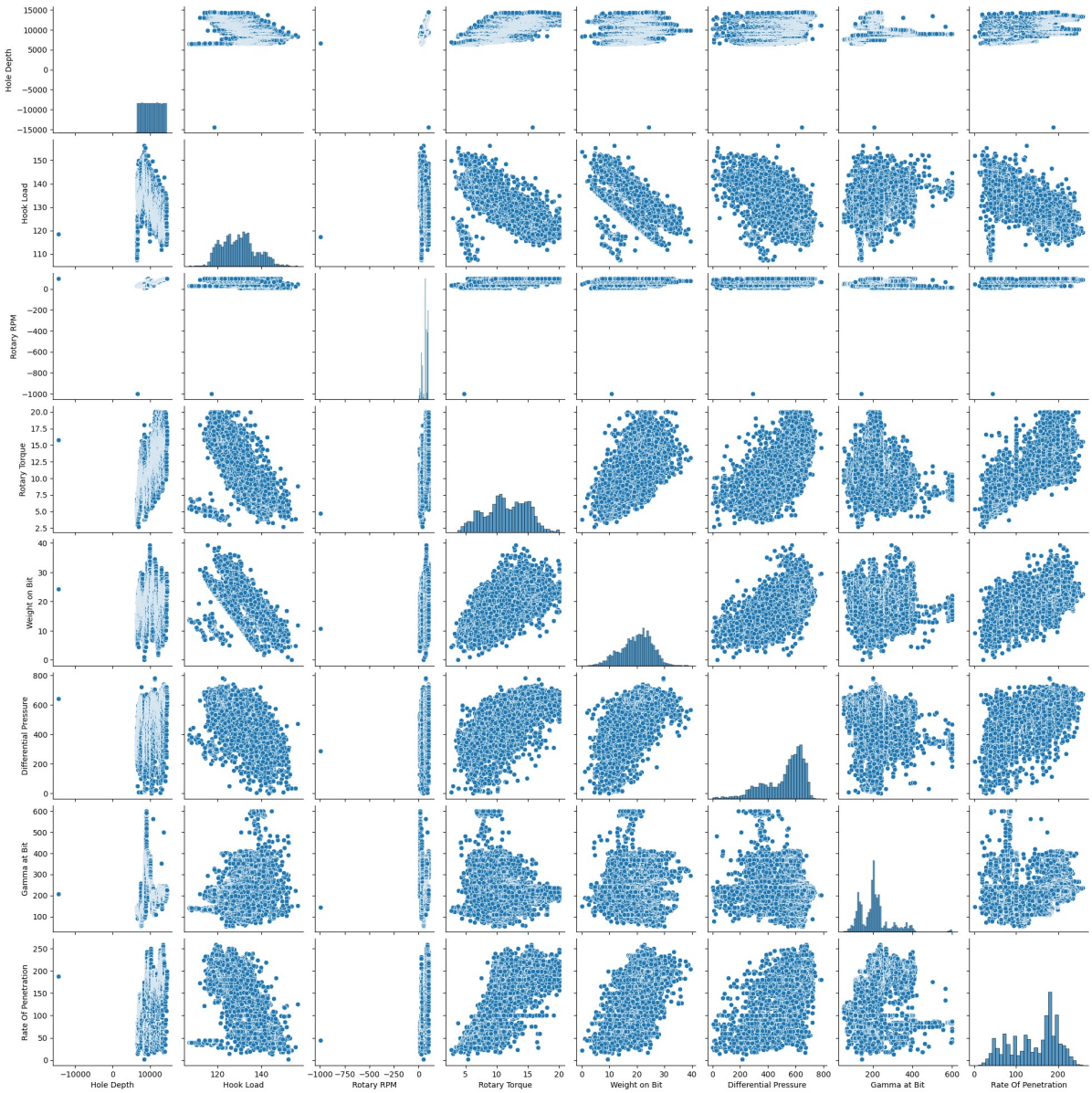

Count
0
100
200
300
400
500
600
700
0
100
200
300
400
500
600
700
800
Differential Pressure

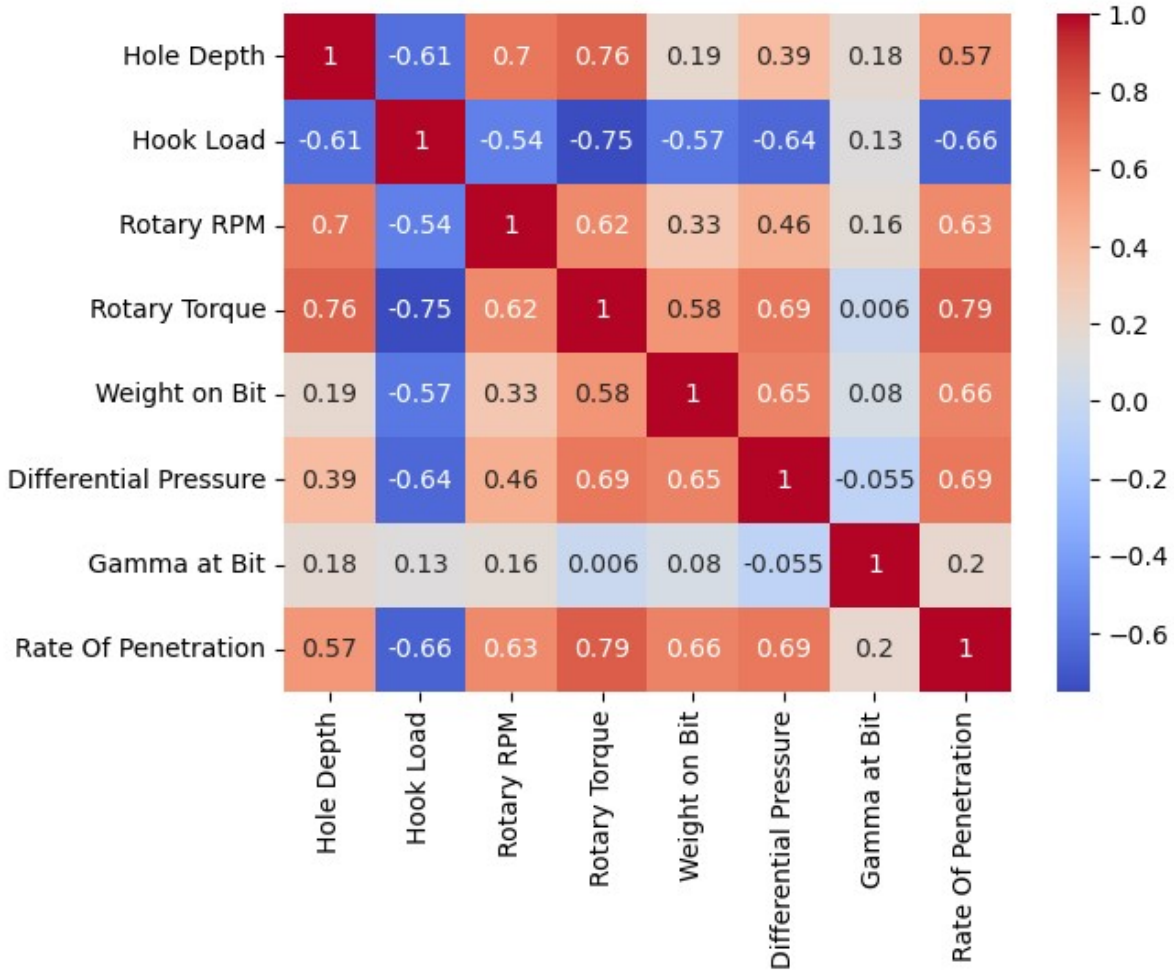

1000
800
600
400
200
0
Count
100
200
300
400
500
600
Gamma at Bit



The Hook Load column has a roughly normal distribution, centered around 130 klbs.
The Rotary RPM column has a roughly normal distribution, centered around 70 rpm.
The Rotary Torque column has a roughly normal distribution.
The Weight on Bit column has a roughly normal distribution, centered around 20 klbs.
The Differential Pressure column has a roughly normal distribution, with a few extreme values on the upper end of the range.
The Gamma at Bit column has a skewed distribution, with one peak around 200.
The Rate of Penetration column has a roughly normal distribution, centered around 150.

In [19]:
```
sns.pairplot(data=df)
plt.show()

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```


Hole Depth
Hook Load
Rotary RPM
Rotary Torque
Weight on Bit
Differential Pressure
Gamma at Bit
Rate Of Penetration



From the correlation matrix, we can see that the features with the highest positive correlations with ROP are Rotary Torque (0.794), Rotary RPM (0.687), and Differential Pressure (0.694). This suggests that these features are important for predicting ROP and should be included in the regression model.

On the other hand, Hook Load (-0.661) and Weight on Bit (-0.656) have the strongest negative correlations with ROP, indicating that as these features increase, ROP tends to decrease. Hole Depth also has a moderate positive correlation with ROP (0.574).

Gamma at Bit has a weak positive correlation with ROP (0.197), suggesting that it may not be as important for predicting ROP as the other features.

Overall, the EDA and correlation analysis suggest that the Rotary Torque, Rotary RPM, and Differential Pressure are important features for predicting ROP, while Hook Load and Weight on Bit may have a negative impact on ROP

In [59]:
```
# Apply the MinMaxScaler to the features
scaler = MinMaxScaler()
numerical_cols = ['Hole Depth', 'Hook Load', 'Rotary RPM', 'Rotary Torque',
df_scaled = pd.DataFrame(scaler.fit_transform(df[numerical_cols]), columns=n
```

In [60]:
```
# Split the data into training and testing sets
X = df_scaled.drop('Rate Of Penetration', axis=1)
y = df_scaled['Rate Of Penetration']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran
```

In [63]:
```
#train an Extra Tree regressor model with following hyperparameters
model_etr = ExtraTreesRegressor(n_estimators=100, criterion='mse', max_depth
model_etr.fit(X_train, y_train)
y_pred_etr = model_etr.predict(X_test)
```

```
C:\Users\nguye\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:396:
FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in
version 1.2. Use `criterion='squared_error'` which is equivalent.
  warn(
```

In [64]:
```
#Evaluate the model
mae = mean_absolute_error(y_test, y_pred_etr)
mse = mean_squared_error(y_test, y_pred_etr)
r2 = r2_score(y_test, y_pred_etr)
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('R^2 Score:', r2)
```

```
Mean Absolute Error: 0.028535904137139106
Mean Squared Error: 0.0025839565825221907
R^2 Score: 0.9446813206927512
```

In [65]:
```
#Train an random forest regression model
model_rf = RandomForestRegressor(n_estimators=100, max_features='sqrt', min_
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)
```

In [66]:
```
#Evaluate the model
mae = mean_absolute_error(y_test, y_pred_rf)
mse = mean_squared_error(y_test, y_pred_rf)
r2 = r2_score(y_test, y_pred_rf)
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('R^2 Score:', r2)
```

```
Mean Absolute Error: 0.042568751845163205
Mean Squared Error: 0.004436287967156051
R^2 Score: 0.9050256521221541
```

In [44]:
```
```

In [67]:
```
#Evaluate importance features
importances = model.feature_importances_
feature_importances = dict(zip(X_train.columns, importances))
print(feature_importances)
```

```
{'Hole Depth': 0.07377974298685983, 'Hook Load': 0.10654404615531163, 'Rotar
y RPM': 0.252233839604786, 'Rotary Torque': 0.27405911213549583, 'Weight on
Bit': 0.09461329847731952, 'Differential Pressure': 0.1479910653668878, 'Gam
ma at Bit': 0.05077889527333958}
```

Most important feature for predicting ROP is 'Rotary Torque', with an importance score of 0.274. The next most important features are 'Rotary RPM' (0.252). 'Differential Pressure' (0.148), , and 'Hook Load' (0.107). The least important feature is 'Gamma at Bit' (0.05).

In [ ]: