Soil Fertility Classification by sing Machine Learning

November 8, 2023

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

0.0.1 1. EDA

[2]:
```
df = pd.read_csv('dataset.csv')
```

[3]:
```
df.head()
```

[3]:

	N	P	K	pH	EC	OC	S	Zn	Fe	Cu	Mn	B	Output
0	138	8.6	560	7.46	0.62	0.70	5.9	0.24	0.31	0.77	8.71	0.11	0
1	213	7.5	338	7.62	0.75	1.06	25.4	0.30	0.86	1.54	2.89	2.29	0
2	163	9.6	718	7.59	0.51	1.11	14.3	0.30	0.86	1.57	2.70	2.03	0
3	157	6.8	475	7.64	0.58	0.94	26.0	0.34	0.54	1.53	2.65	1.82	0
4	270	9.9	444	7.63	0.40	0.86	11.8	0.25	0.76	1.69	2.43	2.26	1

0.0.2 Input

- N - ratio of Nitrogen (NH4+) content in soil
- P - ratio of Phosphorous (P) content in soil
- K - ratio of Potassium (K) content in soil
- ph - soil acidity (pH)
- ec - electrical conductivity
- oc - organic carbon
- S - sulfur (S)
- zn - Zinc (Zn)
- fe - Iron (Fe)
- cu - Copper (Cu)
- Mn - Manganese (Mn)
- B - Boron (B) ### Output

- Class fertility (0 “Less Fertile”, 1 “Fertile”, 2 “Highly Fertile”)

[4]:
```
df.isnull().sum()
```

[4]:
```
N         0
P         0
K         0
pH        0
EC        0
OC        0
S         0
Zn        0
Fe        0
Cu        0
Mn        0
B         0
Output    0
dtype: int64
```

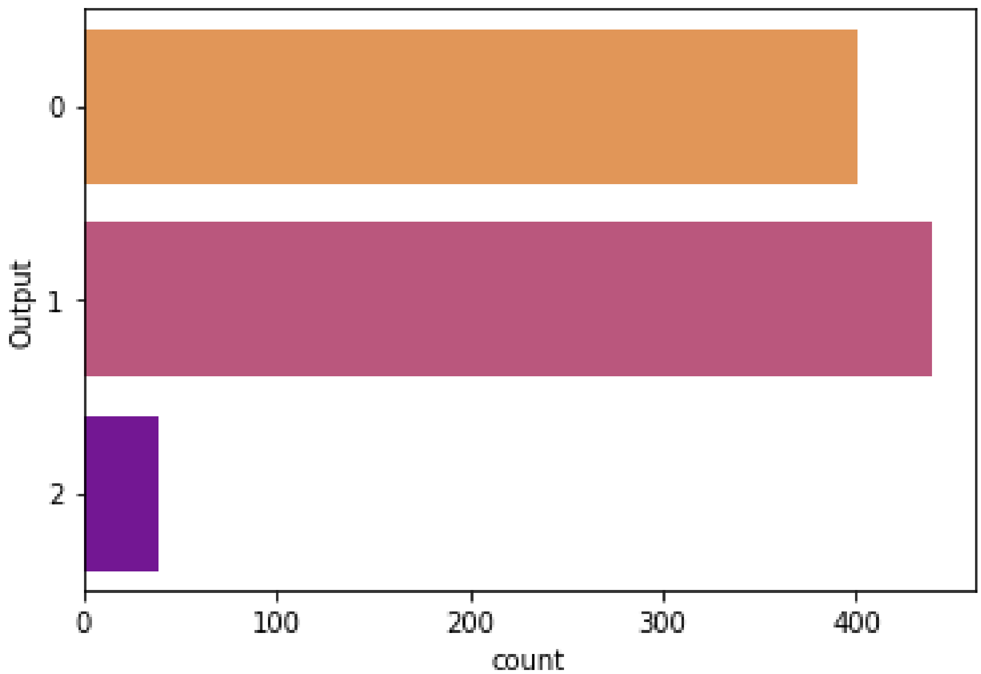
[5]:
```
df.describe()
```

[5]:
```
               N           P           K          pH          EC          OC  \
count  880.00000  880.000000  880.000000  880.000000  880.000000  880.000000
mean   246.73750   14.562159  499.978409    7.510500    0.543659    0.617989
std     77.38886   21.967755  124.222838    0.464912    0.141597    0.842986
min      6.00000    2.900000   11.000000    0.900000    0.100000    0.100000
25%    201.00000    6.800000  412.000000    7.350000    0.430000    0.380000
50%    257.00000    8.100000  475.000000    7.500000    0.545000    0.590000
75%    307.00000   10.550000  581.000000    7.630000    0.640000    0.780000
max    383.00000  125.000000  887.000000   11.150000    0.950000   24.000000

                S          Zn          Fe          Cu          Mn           B  \
count  880.000000  880.000000  880.000000  880.000000  880.000000  880.000000
mean     7.545080    0.469273    4.140523    0.952443    8.666500    0.590159
std      4.424184    1.894234    3.110011    0.465900    4.298828    0.570800
min      0.640000    0.070000    0.210000    0.090000    0.110000    0.060000
25%      4.700000    0.280000    2.050000    0.630000    6.225000    0.270000
50%      6.640000    0.360000    3.565000    0.930000    8.345000    0.405000
75%      8.750000    0.470000    6.320000    1.250000   11.472500    0.610000
max     31.000000   42.000000   44.000000    3.020000   31.000000    2.820000

           Output
count  880.000000
mean     0.588636
std      0.575462
min      0.000000
25%      0.000000
50%      1.000000
```

```
75%        1.000000
max        2.000000
```

[6]:
```
#Visualization of the class in Fertilizer category with countplot
sns.countplot(y='Output',data=df,palette="plasma_r")
```

[6]: <Axes: xlabel='count', ylabel='Output'>

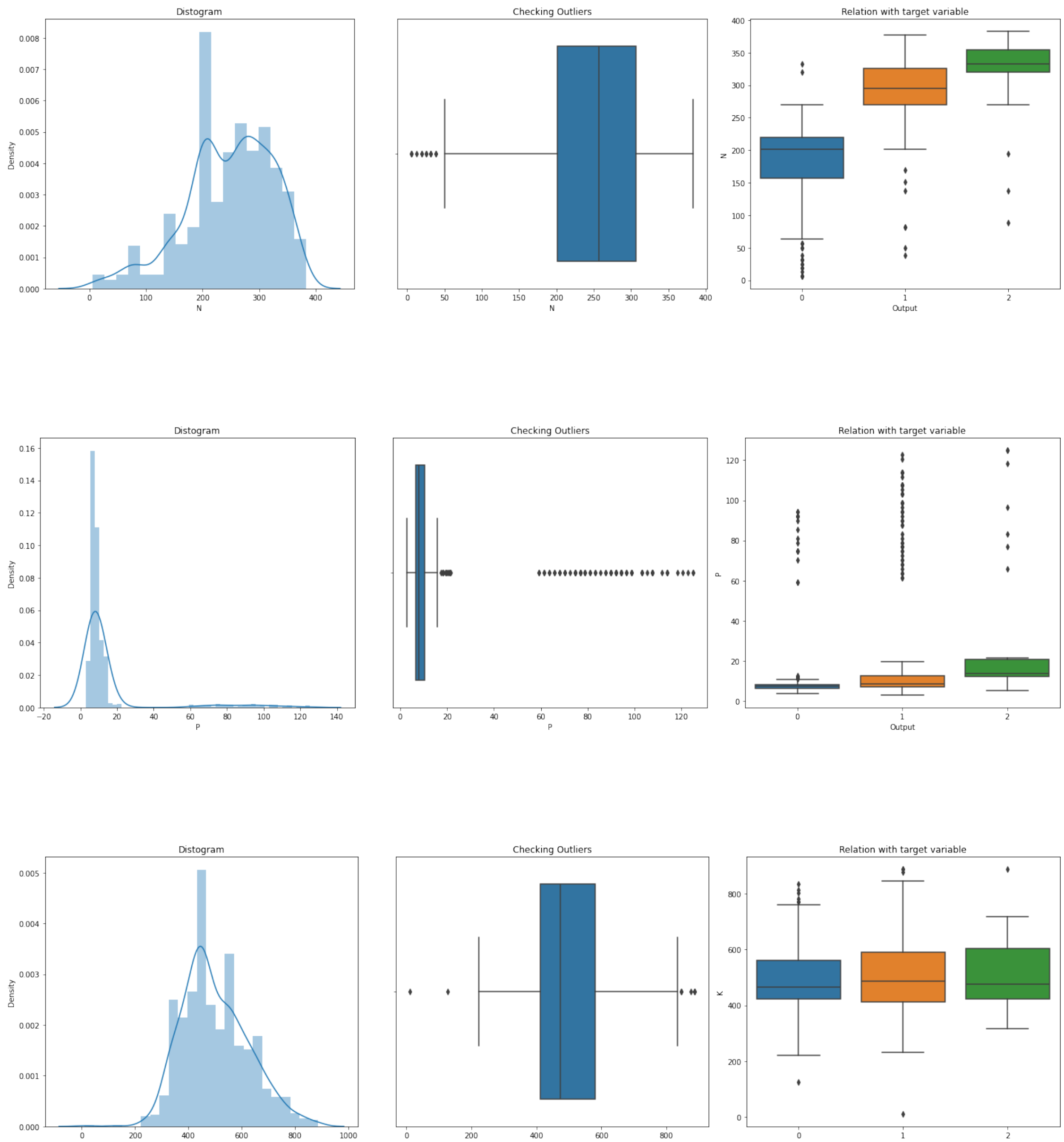


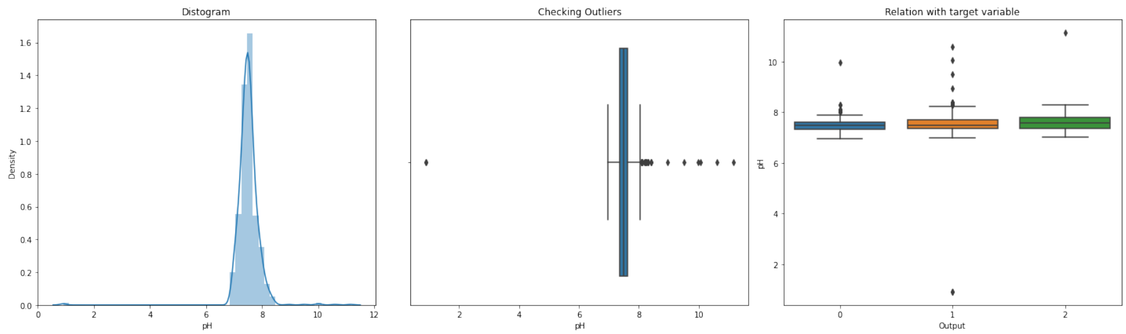
[7]:
```
df['Output'].value_counts()
```

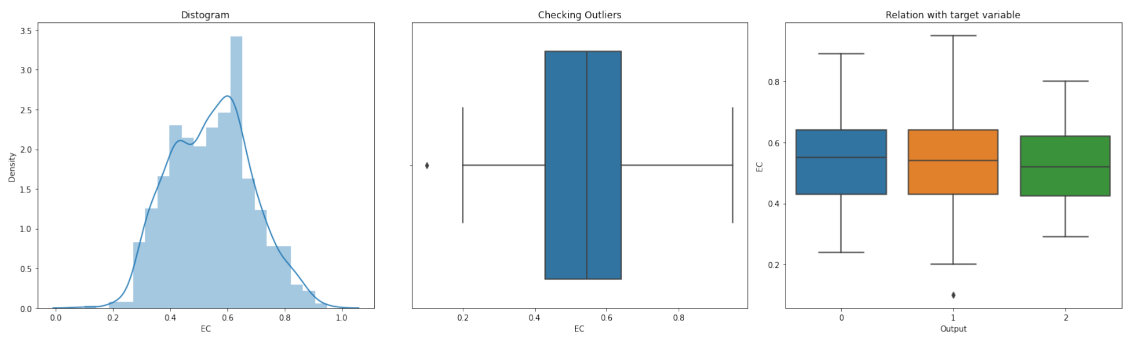
[7]:
```
1    440
0    401
2     39
Name: Output, dtype: int64
```

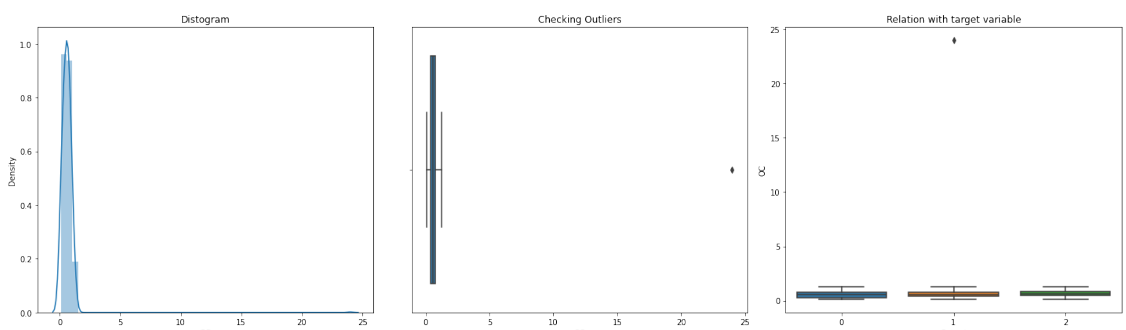
[8]:
```
def plot_conti(x):
    fig, axes = plt.subplots(nrows=1,ncols=3,figsize=(20,6),tight_layout=True)
    axes[0].set_title('Distogram')
    sns.distplot(x,ax=axes[0])
    axes[1].set_title('Checking Outliers')
    sns.boxplot(x,ax=axes[1])
    axes[2].set_title('Relation with target variable')
    sns.boxplot(y = x,x = df['Output'])
```

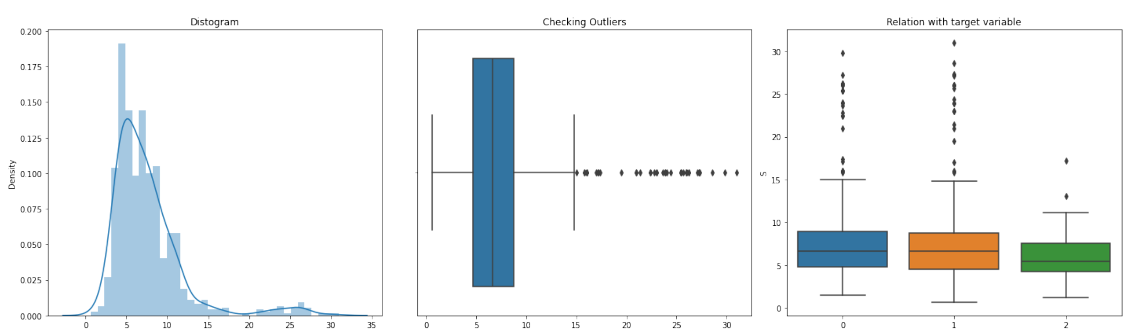
```
[9]: # Create box plots for each numerical variable
for i, column in enumerate(df.columns[:-1]):
    plot_conti(df[column])
plt.tight_layout()
plt.show();
```

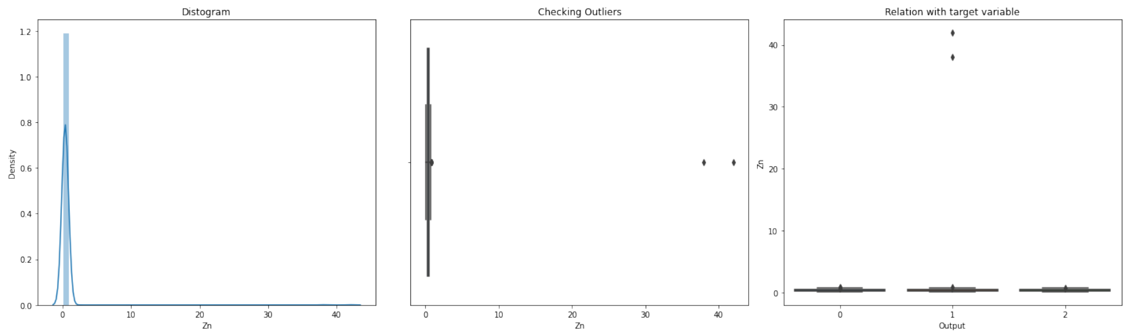


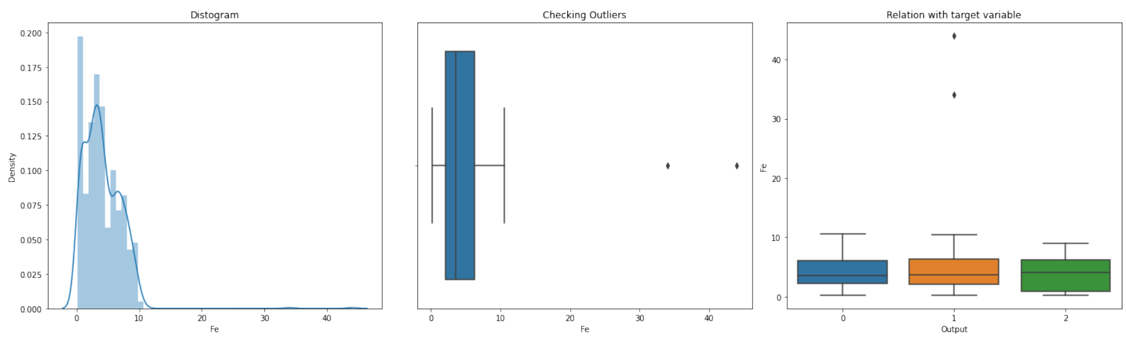

Distogram
Density
pH
Checking Outliers
pH
Relation with target variable
pH
Output

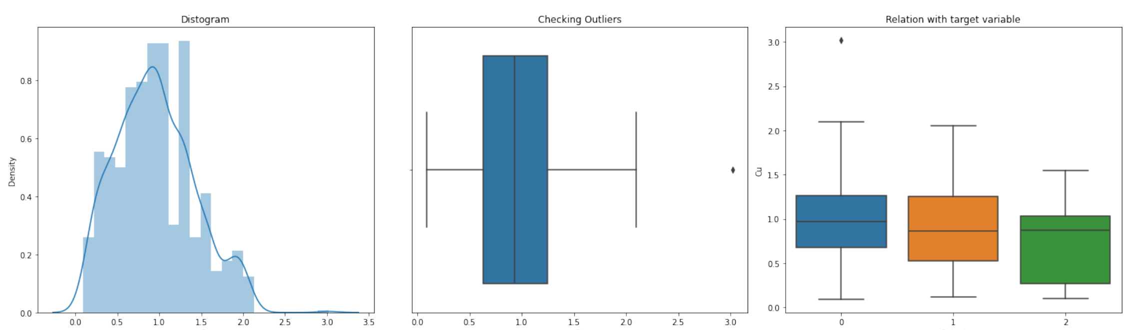

Distogram
Density
EC
Checking Outliers
EC
Relation with target variable
EC
Output

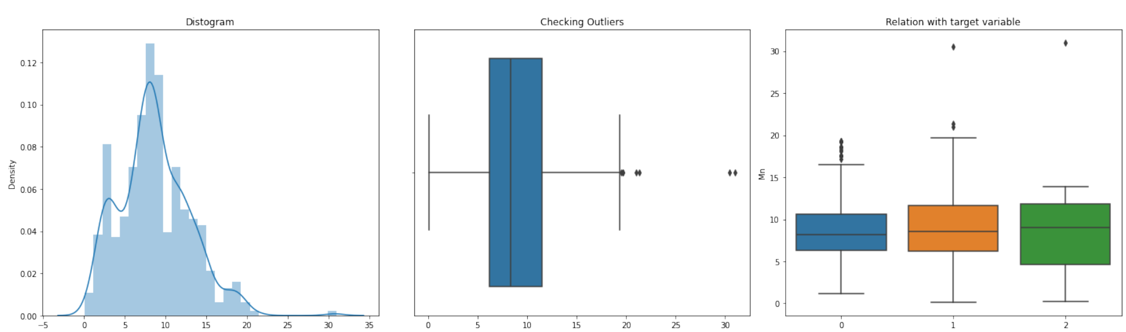

Distogram
Density
OC
Checking Outliers
OC
Relation with target variable
OC
Output

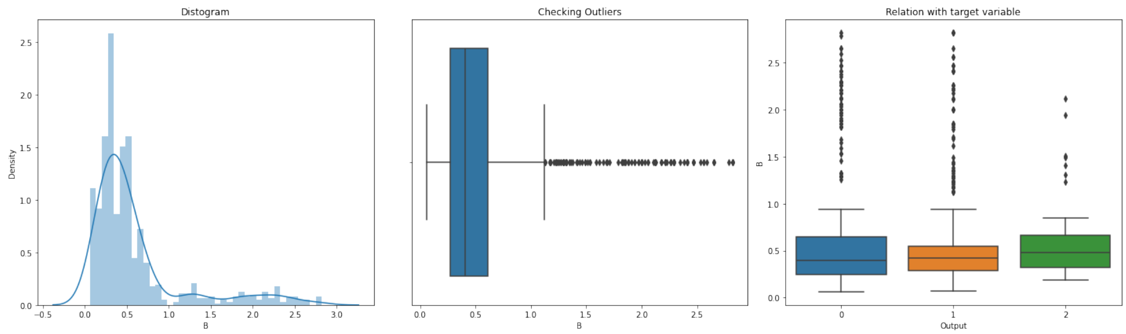

Distogram
Density
S
Checking Outliers
S
Relation with target variable
S
Output


Distogram
Density
Zn
Checking Outliers
Zn
Relation with target variable
Zn
Output

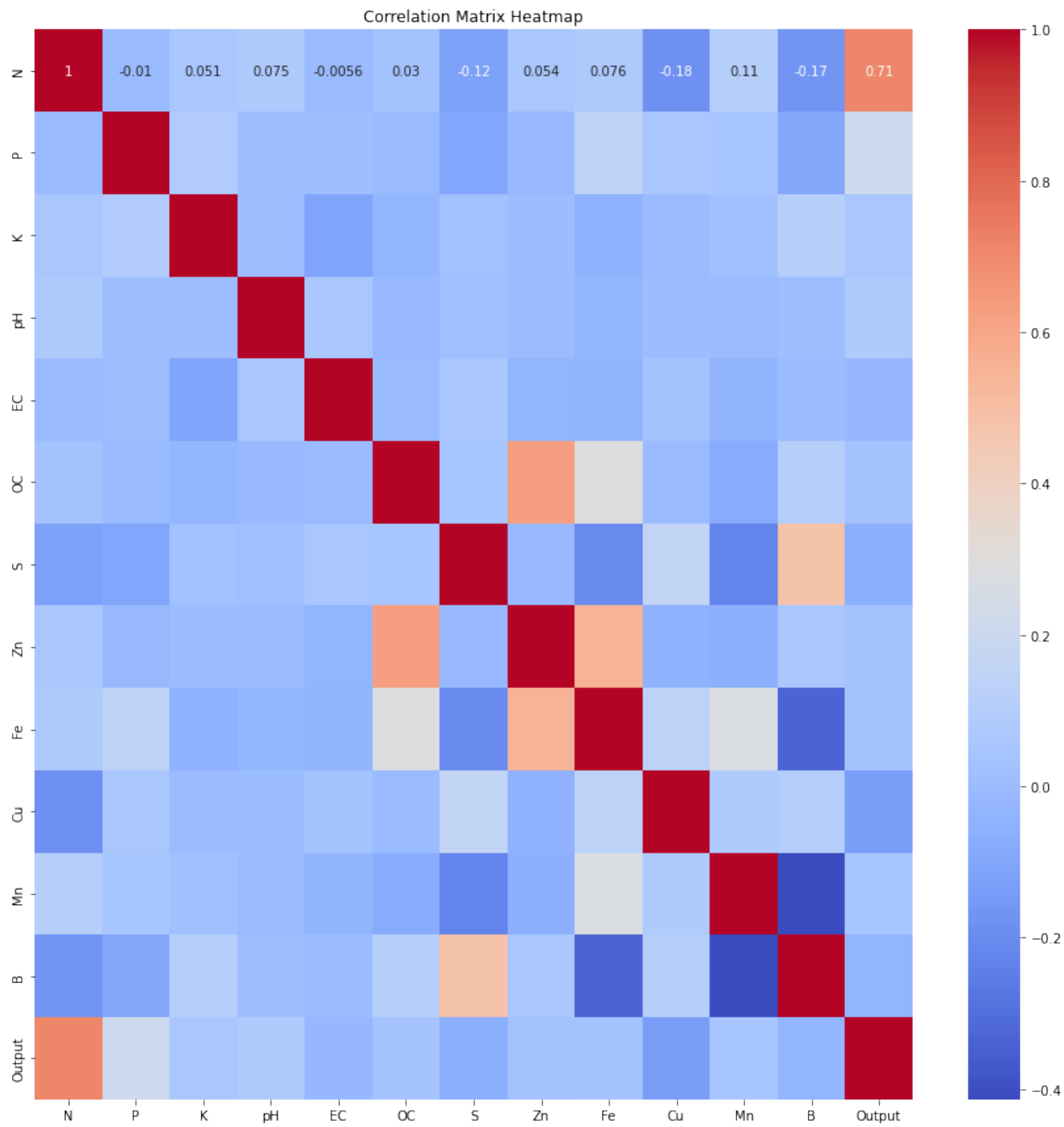

Distogram
Density
Fe
Checking Outliers
Fe
Relation with target variable
Fe
Output


Distogram
Density
Cu
Checking Outliers
Cu
Relation with target variable
Cu
Output


Distogram
Density
Mn
Checking Outliers
Mn
Relation with target variable
Mn
Output



[31]:
```
# Create a correlation matrix heatmap
plt.figure(figsize=(15,15))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



As we can see 'N' has a highest corr with 'Output'. At this point I don't that we should use only 'N' feature or use all the rest to fit the model. Therefore, u guys can try to them.

0.0.3 2. Data Preprocessing

[11]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
```

[12]:
```python
# Split the data into features (X) and target (y)
X = df.drop('Output', axis=1)
y = df['Output']
```

[13]:
```python
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.0.4 3. Model Fitting

a. Random Forest Classifier

[14]:
```python
# Scale the feature data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[15]:
```python
# Train a Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

[15]:
```
RandomForestClassifier(random_state=42)
```

[16]:
```python
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

[17]:
```python
# Print a classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94        78
           1       0.87      0.91      0.89        88
           2       0.00      0.00      0.00        10

    accuracy                           0.89       176
   macro avg       0.59      0.63      0.61       176
weighted avg       0.84      0.89      0.86       176
```

b. SVM

[18]:
```python
from sklearn import svm
```

[19]:
```python
# Train a SVM classifier
clf = svm.SVC(kernel='linear') # Linear Kernel
clf.fit(X_train, y_train)
```

[19]:
```
SVC(kernel='linear')
```

[20]:
```python
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

[21]:
```python
# Print a classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89        78
           1       0.82      0.88      0.85        88
           2       0.00      0.00      0.00        10

    accuracy                           0.84       176
   macro avg       0.56      0.60      0.58       176
weighted avg       0.79      0.84      0.82       176
```

c. Gradient Boosting Classifier

[22]:
```python
from sklearn.ensemble import GradientBoostingClassifier
```

[23]:
```python
# Train a Gradient Boosting classifier
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,␣
 ↪max_depth=1, random_state=42)
clf.fit(X_train, y_train)
```

[23]:
```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
```

[24]:
```python
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

[25]:
```python
# Print a classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.90        78
           1       0.85      0.85      0.85        88
           2       0.50      0.10      0.17        10

    accuracy                           0.85       176
   macro avg       0.74      0.63      0.64       176
weighted avg       0.84      0.85      0.84       176
```

[27]:
```python
pd.DataFrame({
    'Model': ['RandomForestClassifier', 'SVM', 'GradientBoostingClassifier'],
    'Accuracy': [89,84,85]
})
```

[27]:

	Model	Accuracy
0	RandomForestClassifier	89
1	SVM	84
2	GradientBoostingClassifier	85

[ ]:

```

```