Predicting Mineral Structure

Project Description

Project Description - The Problem

What is the problem?

There are many minerals found in the world which cannot be identified on the spot. This is because many minerals are similar in looks and also have similar properties. It is only through further analysis is can be identified correctly. This further analysis utilises lots of time and resources which can be used elsewhere, such as mapping and understanding how the local geographical landscape is composed.

Project Description - The Solution

How can this problem be solved?

There is no definite way to solve the problem, but rather ways to make it simpler and faster to handle. By using historical values and expert knowledge, the identification and classification of minerals becomes much easier.

Project Description - AI utilization

3) How does AI help to make this possible:

Artificial Intelligence can be used to simplify and reduce the amount of time spent on this problem. Using a pre-existing database, the algorithm will be able to predict the correct mineral, using less data than what was previously required. Also, minerals can be classified into similar groups, almost instantaneously, through the use of artificial intelligence.

Data Description

Data Description

The dataset being utilised consists of 140 columns of different attributes, alongside 3112 rows of unique minerals.

The first column used, titled ‘Name’, acts as the primary key / unique identifier of the dataset. It allows the data to be differentiated for each mineral.

The prediction column will be the 'Crystal Structure' column.

Methodology

Methodology

To build the mineral structure prediction code, we require an algorithm upon which the foundation will be based on.

There are 5 algorithms to choose from:

1. TensorFlow
2. PyTorch
3. Linear Regression
4. SVM Regressor
5. Random Forest Regressor

Methodology - TensorFlow

TensorFlow is an open-source library which can be utilised by Python code. It is described as one the most specialised algorithms
to train and test various models.
TensorFlow uses symbolic math which allows for deep neural network training.
The TensorFlow algorithm enables an ability where
the developers can create graphs / structures which can describe the data and how it flows, hence allowing it to be modeled.

Methodology - PyTorch

Pytorch is exceptionally similar to TensorFlow, where it is an algorithm based off the Torch Library which is utilised to train and
test models.
However, the main difference between the two algorithms is how they function. PyTorch uses dynamic computational graphs, which are much
more practical than static graphs, allowing users to make accurate evaluations from the graphs.

Methodology - Linear Regression

Linear Regression is a simple and mathematical way to obtain a result based off of two variables, a dependant and independent.

By plotting the variables in a graph, a correlation or link will be identified through various stages of training and testing, and a

straight solid line (best fit) will be produced.

Predictions can be solidified based on the line of best fit and errors can easily be identified.

Methodology - SVM Regressor

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Ironically, an SVM Regressor is most commonly used for classification, however that is not to say it can be used for regression as well.

The SVM Regressor works almost identically as a regular linear regression algorithm does, however creates a line of best fit for data which is discrete, where normal patterns cannot be recognised.

Methodology - Random Forest Regressor

Random Forest Regressor utilises many decision trees to predict and output from a variable. The algorithm calculates the most populous
and most commonly occuring outcome and outputs it as a prediction.

Methodology - Final

It would be in the best interest to utilize the SVM Regressor and Random Forest Regressor, in order to create an accurate prediction.

The Program

```
[2] # Data handling and Visualisation Tools
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn import datasets, linear_model
    import seaborn as sns
    import hvplot.pandas
```

```
[3] # Loading Dataset
    dataset = pd.read_csv('/content/Minerals_Database.csv')
```

```python
# Visualising Dataset
dataset.head(20)
```

	Unnamed: 0	Name	Crystal Structure	Mohs Hardness	Diaphaneity	Specific Gravity	Optical	Refractive Index	Dispersion	Hydrogen	...	Acetate	Phosphate	Sulphate	Carbonate	Ammonium	Hydrated Water	count	Molar Mass	Mola Volum
0	0	Abenakiite-(Ce)	5.0	4.50	0.0	3.240	3.0	1.580	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	23.0	817.339002	0.12339
1	1	Abernathyite	4.0	2.75	3.0	3.446	3.0	1.592	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	1.0	9.0	435.069330	0.05608
2	2	Abhurite	5.0	2.00	3.0	4.420	3.0	2.085	0.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	17.0	921.092220	0.12263
3	3	Abswurmbachite	0.0	0.00	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	12.0	550.019900	0.03365
4	4	Actinolite	2.0	5.50	2.0	1.050	4.0	1.634	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	28.0	861.185368	0.11207
5	5	Acuminite	2.0	3.50	3.0	3.295	4.0	1.457	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	8.0	225.618151	0.04488
6	6	Adamite	3.0	3.50	0.0	4.400	4.0	1.742	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	8.0	270.707130	0.05602
7	7	Adelite	0.0	0.00	0.0	0.000	0.0	0.000	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	10.0	251.283292	0.06726
8	8	Admontite	2.0	2.50	0.0	0.000	4.0	1.473	0.0	14.0	...	0.0	0.0	0.0	0.0	0.0	4.0	39.0	407.639360	0.30261
9	9	Aegirine	2.0	6.00	1.0	3.550	4.0	1.776	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	6.0	154.920468	0.03363
10	10	Aenigmatite	1.0	5.50	1.0	3.810	4.0	1.829	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	41.0	1110.587536	0.22418
11	11	Aerinite	2.0	3.00	2.0	2.480	4.0	3.383	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	18.0	706.631179	0.12324
12	12	Aerugite	5.0	4.00	1.0	5.900	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	8.0	331.227580	0.04482
13	13	Afghanite	5.0	3.50	3.0	2.600	3.0	1.526	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	64.0	1839.871467	0.22444
14	14	Afwillite	2.0	3.50	2.0	2.630	4.0	1.624	0.0	24.0	...	0.0	0.0	0.0	0.0	0.0	0.0	84.0	2712.511360	0.62779
15	15	Agrellite	1.0	5.50	2.0	2.880	4.0	1.576	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	39.0	922.559878	0.24665
16	16	Agrinierite	0.0	0.00	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	5.0	27.0	1497.192900	0.12352
17	17	Ahlfeldite	2.0	0.00	0.0	0.000	0.0	0.000	0.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	11.0	221.682160	0.10084
18	18	Ajoite	1.0	2.00	2.0	2.960	4.0	1.591	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	79.0	2011.923656	0.42597
19	19	Akaganeite	2.0	0.00	2.0	3.520	0.0	0.000	0.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	10.0	214.168120	0.08945

20 rows × 140 columns

```
[5] # Understanding dataset
    dataset.shape
```

```
(3112, 140)
```

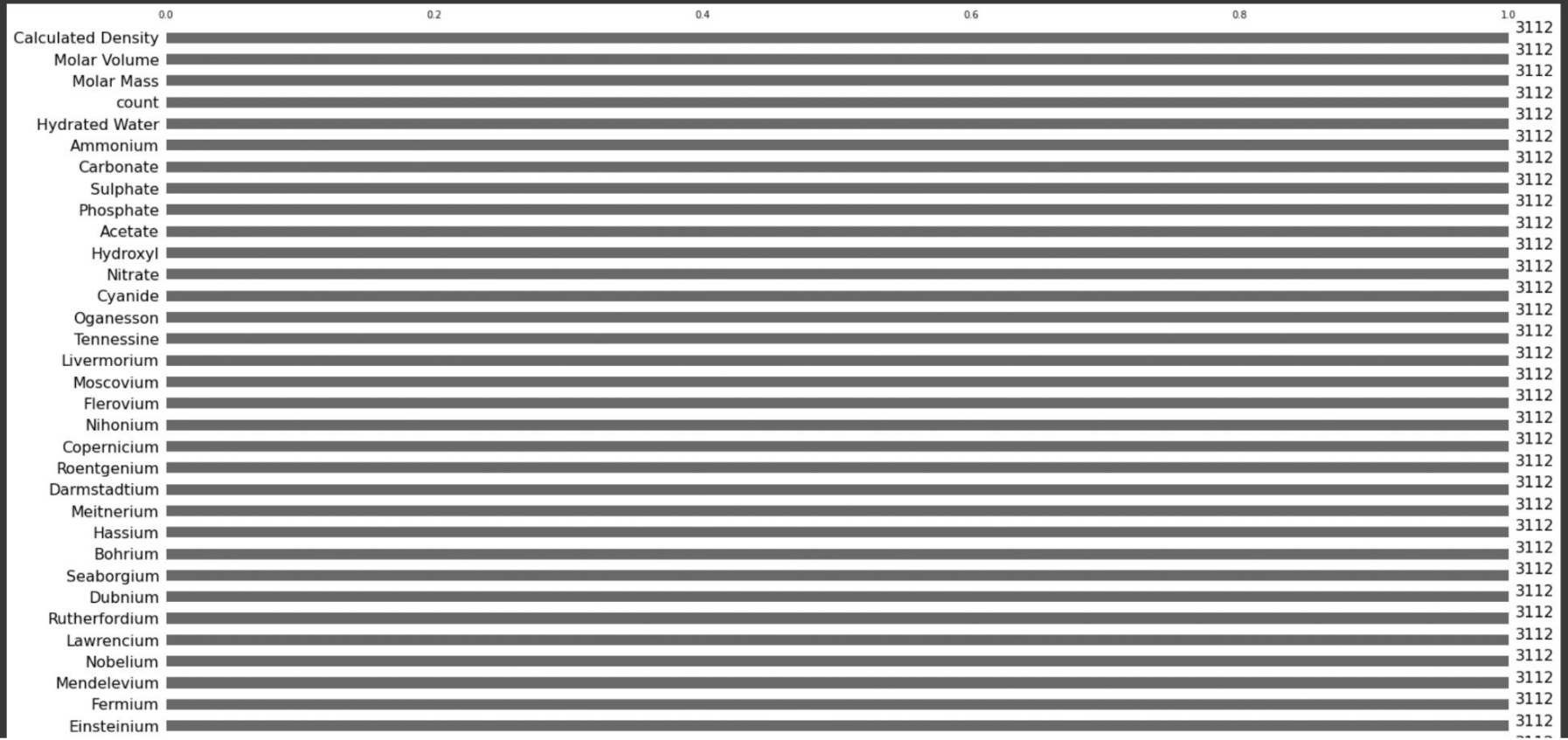
```
[6] # Identifying any duplicate values
    print(f"Number of duplicates: {dataset.duplicated().sum()}")
    print(f"Percentage of duplicates: {dataset.duplicated().sum()/len(dataset)*100}%")
```

```
Number of duplicates: 0
Percentage of duplicates: 0.0%
```

```python
# Identifying the cardinality of the dataset
dataset.nunique()
```

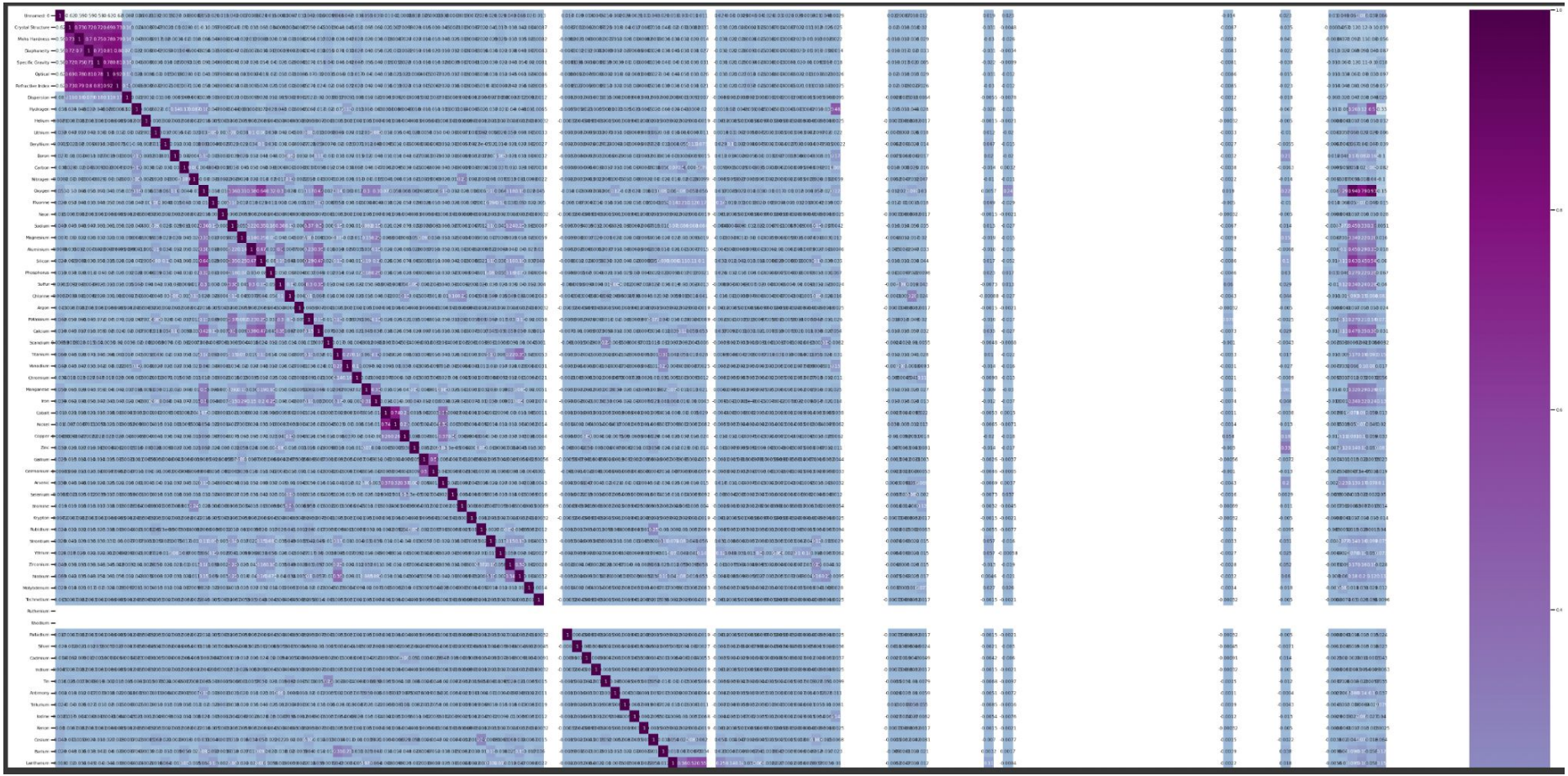
```
Unnamed: 0                 3112
Name                       3112
Crystal Structure             7
Mohs Hardness                46
Diaphaneity                   4
                            ...
Hydrated Water               27
count                       116
Molar Mass                 2937
Molar Volume               2901
Calculated Density         2509
Length: 140, dtype: int64
```

```
[8] #Identifying missing values
    import missingno as msno
    msno.bar(dataset)
    plt.show()
```



Exploratory Data Analysis (EDA)

```
[9] # Creating a heatmap graph to find the best correlation
    sns.set_context('poster', font_scale=0.5)
    plt.figure(figsize=(80,80))
    sns.heatmap(dataset.corr(), annot=True, cmap=plt.cm.BuPu)
    plt.show()
```

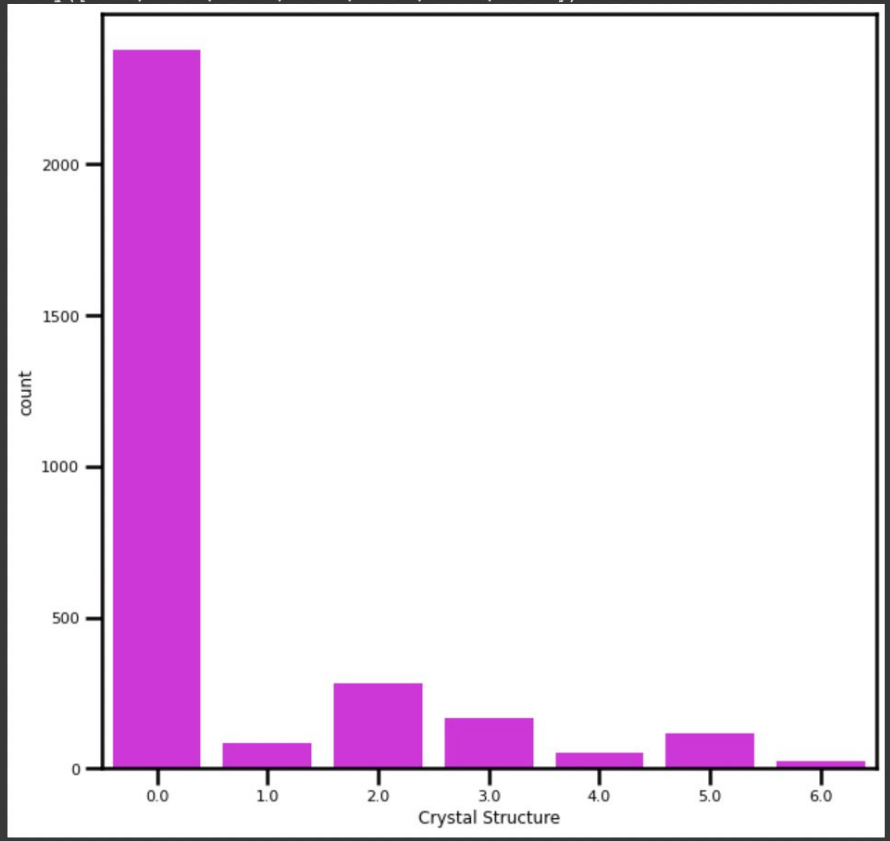

Brinell Hardness
Crystal Structure
Mohs Hardness
Diaphaneity
Specific Gravity
Optical
Refractive Index
Dispersion
Hydrogen
Helium
Lithium
Beryllium
Boron
Carbon
Nitrogen
Oxygen
Fluorine
Neon
Sodium
Magnesium
Aluminum
Silicon
Phosphorus
Sulfur
Chlorine
Argon
Potassium
Calcium
Scandium
Titanium
Vanadium
Chromium
Manganese
Iron
Cobalt
Nickel
Copper
Zinc
Gallium
Germanium
Arsenic
Selenium
Bromine
Krypton
Rubidium
Strontium
Yttrium
Zirconium
Niobium
Molybdenum
Technetium
Ruthenium
Rhodium
Palladium
Silver
Cadmium
Indium
Tin
Antimony
Tellurium
Iodine
Xenon
Cesium
Barium
Lanthanum
1.0
0.8
0.6
0.4

```
[ ] # Further identification of correlated values to Crystal Structure
    target = abs(dataset.corr()["Crystal Structure"])
    relevant_attributes = target [target>0.5]
    relevant_attributes
```

```
Unnamed: 0            0.624126
Crystal Structure     1.000000
Mohs Hardness         0.729784
Diaphaneity           0.724877
Specific Gravity      0.722504
Optical               0.694525
Refractive Index      0.729662
Name: Crystal Structure, dtype: float64
```

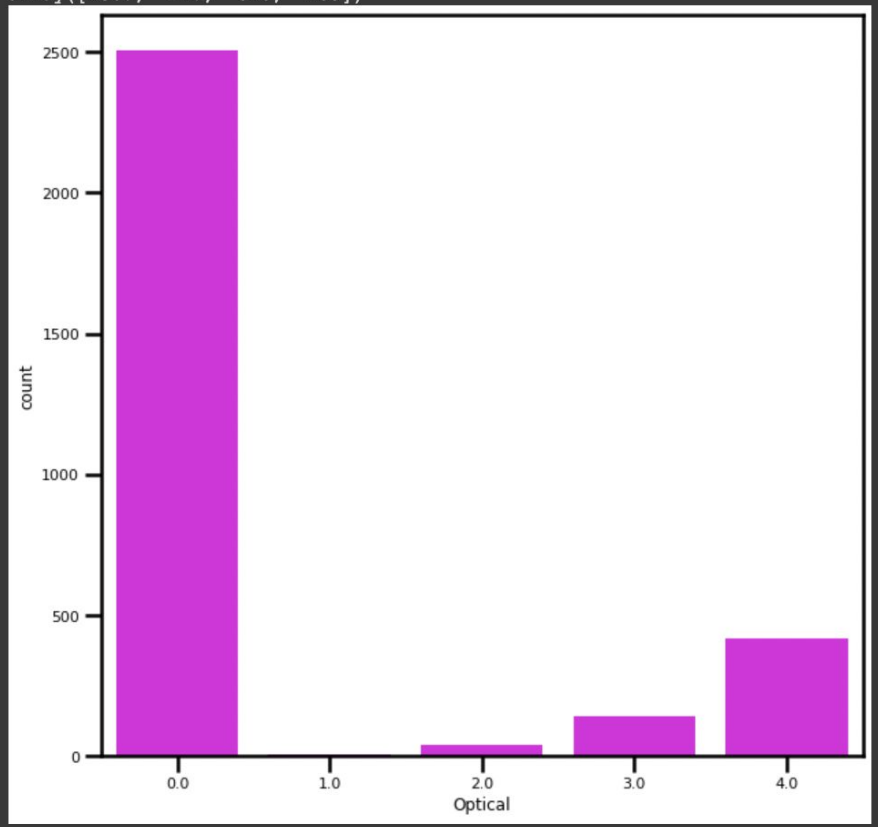
```python
# Looking at the distribution of Crystal Structure
plt.figure(figsize=(10,10))
sns.countplot(data=dataset,x = 'Crystal Structure',color='magenta')
np.bincount(dataset['Crystal Structure'])
```

```
array([2380,   84,  284,  169,   53,  116,   26])
```



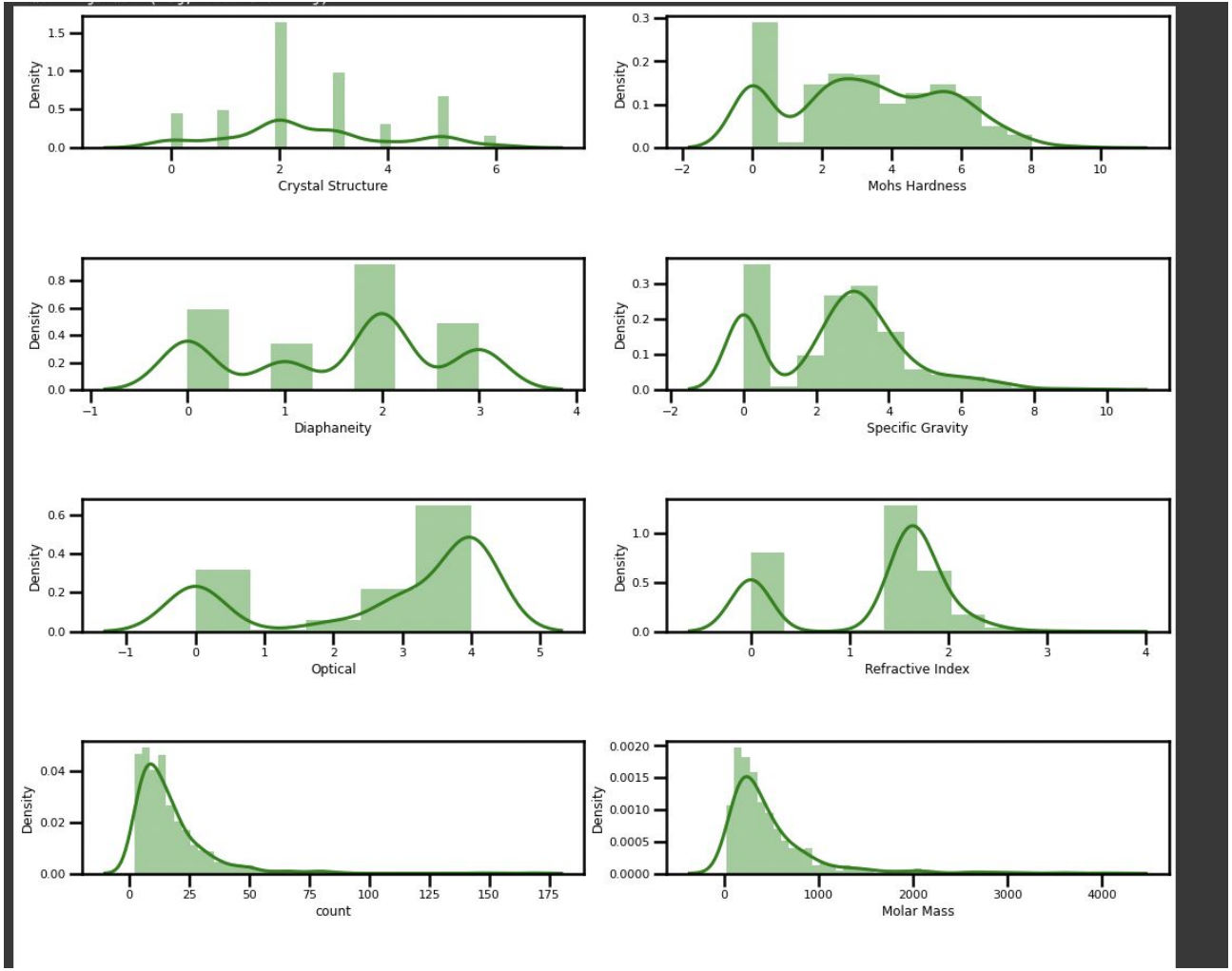
```python
#Looking at the Distribution of Diaphaneity
plt.figure(figsize=(10,10))
sns.countplot(data=dataset,x = 'Optical',color='magenta')
np.bincount(dataset['Diaphaneity'])
```

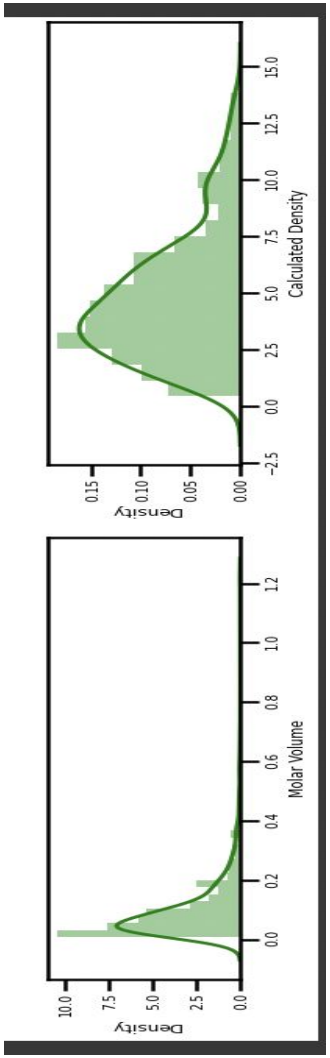
```
array([2507,  117,  320,  168])
```



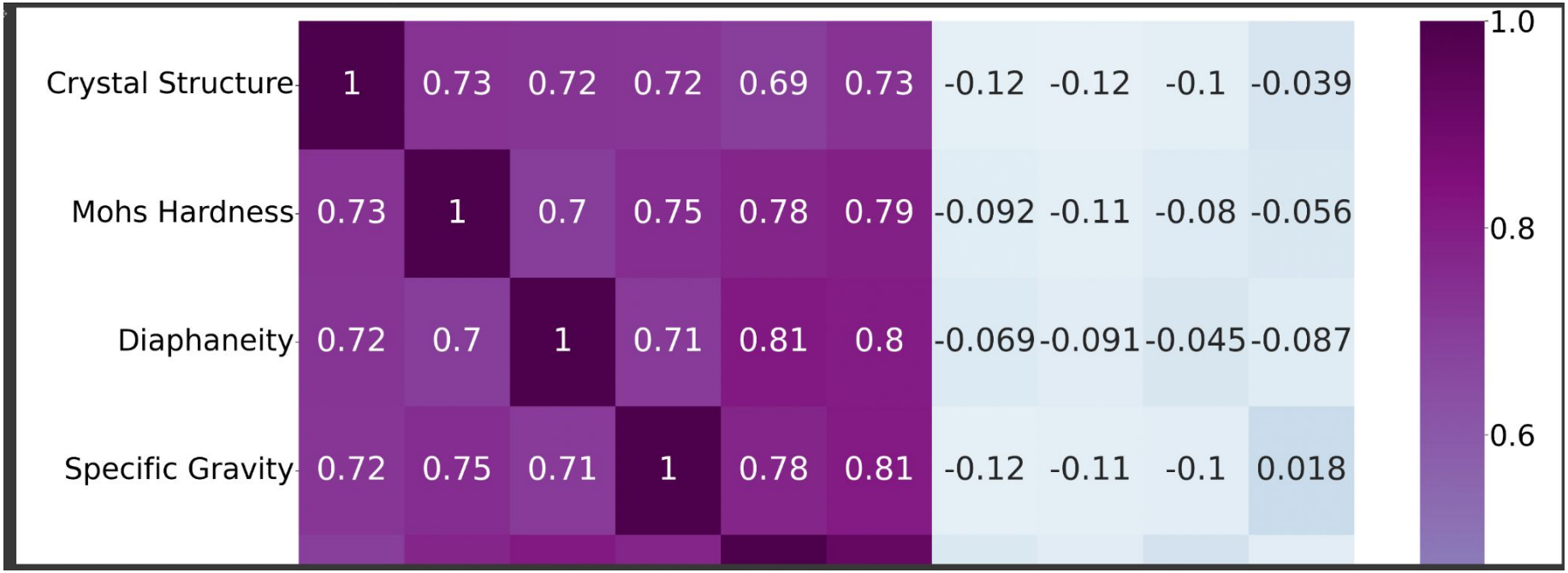
```
[ ] # Creating an array of most likely attributes
    characteristic_list = ['Crystal Structure', 'Mohs Hardness', 'Diaphaneity', 'Specific Gravity', 'Optical',
                           'Refractive Index', 'count', 'Molar Mass', 'Molar Volume', 'Calculated Density']
    attribute_list = dataset[characteristic_list].iloc[0:809]
```

```python
# Identifying the distribution of material characteristics
fig, axs = plt.subplots(ncols=2, nrows=5, figsize=(15, 15))
index = 0
axs = axs.flatten()
for k, v in attribute_list.items():
    g = sns.distplot(v, ax=axs[index], color='green')
    index += 1
plt.tight_layout(pad = 0.4, w_pad = 0.5, h_pad = 5.0)
```


Density
Crystal Structure
Mohs Hardness
Diaphaneity
Specific Gravity
Optical
Refractive Index
count
Molar Mass


Density
Calculated Density
Molar Volume

```python
# Creating a heatmap graph to find correlations between columns
sns.set_context('poster', font_scale=4.5)
plt.figure(figsize=(80,80))
sns.heatmap(dataset[characteristic_list].corr(), annot=True, cmap=plt.cm.BuPu)
plt.show()
```


Crystal Structure
Mohs Hardness
Diaphaneity
Specific Gravity
1
0.73
0.72
0.72
0.69
0.73
-0.12
-0.12
-0.1
-0.039
0.73
1
0.7
0.75
0.78
0.79
-0.092
-0.11
-0.08
-0.056
0.72
0.7
1
0.71
0.81
0.8
-0.069
-0.091
-0.045
-0.087
0.72
0.75
0.71
1
0.78
0.81
-0.12
-0.11
-0.1
0.018
1.0
0.8
0.6

```
[ ] # Selecting the features with the greatest correlation
    cor_target = abs(dataset.corr()["Crystal Structure"])
    relevant_features = cor_target [cor_target>0.5]
    relevant_features
```

```
Unnamed: 0           0.624126
Crystal Structure    1.000000
Mohs Hardness        0.729784
Diaphaneity          0.724877
Specific Gravity     0.722504
Optical              0.694525
Refractive Index     0.729662
Name: Crystal Structure, dtype: float64
```

```
[ ] # Identifying the number of mineral with values greater than 0
    attribute_list = attribute_list.loc[dataset['Mohs Hardness'] > 0]
    attribute_list = attribute_list.loc[dataset['Diaphaneity'] > 0]
    attribute_list = attribute_list.loc[dataset['Specific Gravity'] > 0]
    attribute_list = attribute_list.loc[dataset['Refractive Index'] > 0]
    len(attribute_list)
```

```
451
```

```
[ ] # Importing Machine Learning Tools
    import tensorflow as tf
    from sklearn import linear_model
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import confusion_matrix, plot_confusion_matrix, accuracy_score
    from sklearn import metrics
    from sklearn import svm
    from sklearn import preprocessing
    from sklearn import utils
    import torch
    from sklearn.preprocessing import StandardScaler
    from sklearn.svm import SVR
```

```
dataset.drop(dataset.index[(dataset['Crystal Structure'] == 0)],axis=0,inplace=True)
X = dataset[['Specific Gravity','Optical','Mohs Hardness','Diaphaneity','Molar Mass','Refractive Index','Calculated Density','Molar Volume','count']]
y = dataset['Crystal Structure']
lab = preprocessing.LabelEncoder()
y_transformed = lab.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y_transformed, test_size = 0.30, random_state = 42)
y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

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

```
(512, 9)
(220, 9)
(512, 1)
(220, 1)
```

```
[ ] # SVM Regressor
    z = svm.SVC(kernel='linear')
    z = z.fit(X_train, y_train)
    z_pred = z.predict(X_test)
    confusion_matrix(y_test, z_pred)
    accuracy_svm = accuracy_score(y_test, z_pred)
    print(accuracy_svm)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/u
  y = column_or_1d(y, warn=True)
0.4
```

```
[ ] # Random Forest Regressor
    forest = RandomForestClassifier(n_estimators = 150, criterion = "entropy")
    forest = forest.fit(X_train, y_train)
    forest_pred = forest.predict(X_test)
    confusion_matrix(y_test, forest_pred)
    rf_accuracy = accuracy_score(y_test, forest_pred)
    print(rf_accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWa
  This is separate from the ipykernel package so we can avoid doing imports unti
0.509090909090909
```

Conclusion

Conclusion

In conclusion, the Random Forest Regressor has the best accuracy, meaning that it should be utilised as the primary program for mineral prediction.

However, the accuracy was only 51%, meaning that the program is not completely accurate - but able to classify minerals to an appropriate degree.