EESTech Challenge 2017
Patras Local Round
Exercise 2

March 31, 2017

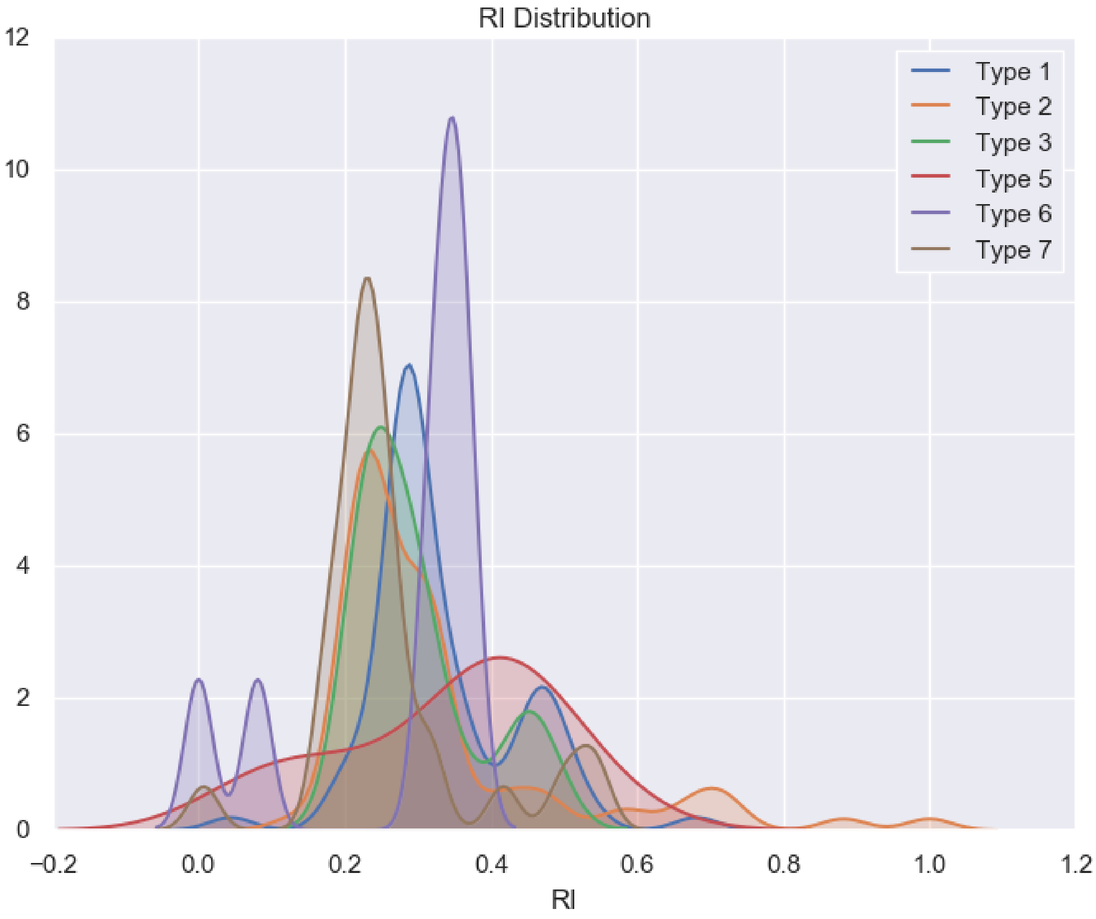
```
In [1]: #%%
        import pandas as pd
        import os
        import matplotlib.pyplot as plt
        plt.style.use('classic')
        pd.options.display.max_columns = 100
        pd.options.display.max_rows = 100

        import numpy as np
        import seaborn as sns; sns.set()
```

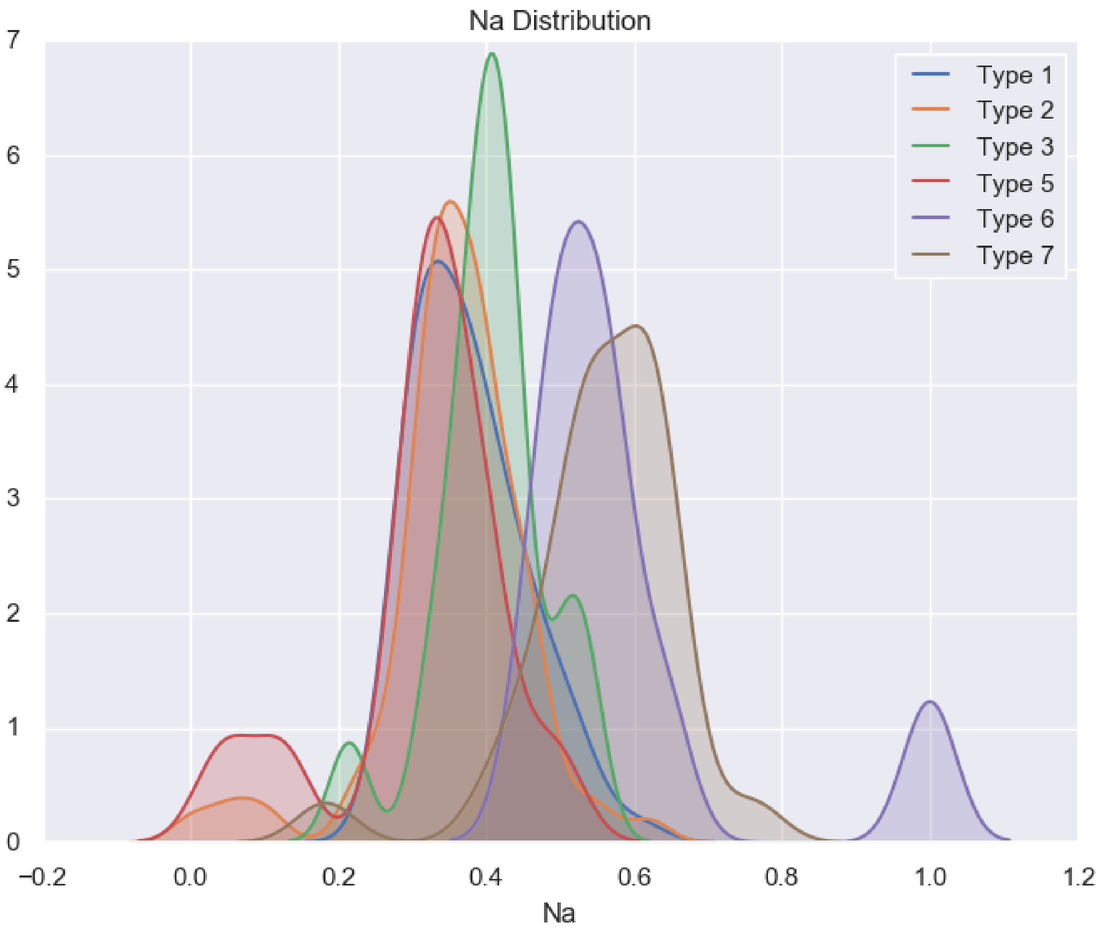
```
In [2]: #%% Set script directory as current working directory
        # abspath = os.path.abspath(__file__)
        abspath = 'C:/Users/Tilemahos/Documents/EESTech Challenge 2017/Askisi 2/'
        dname = os.path.dirname(abspath)
        os.chdir(dname)
        del abspath,dname
```

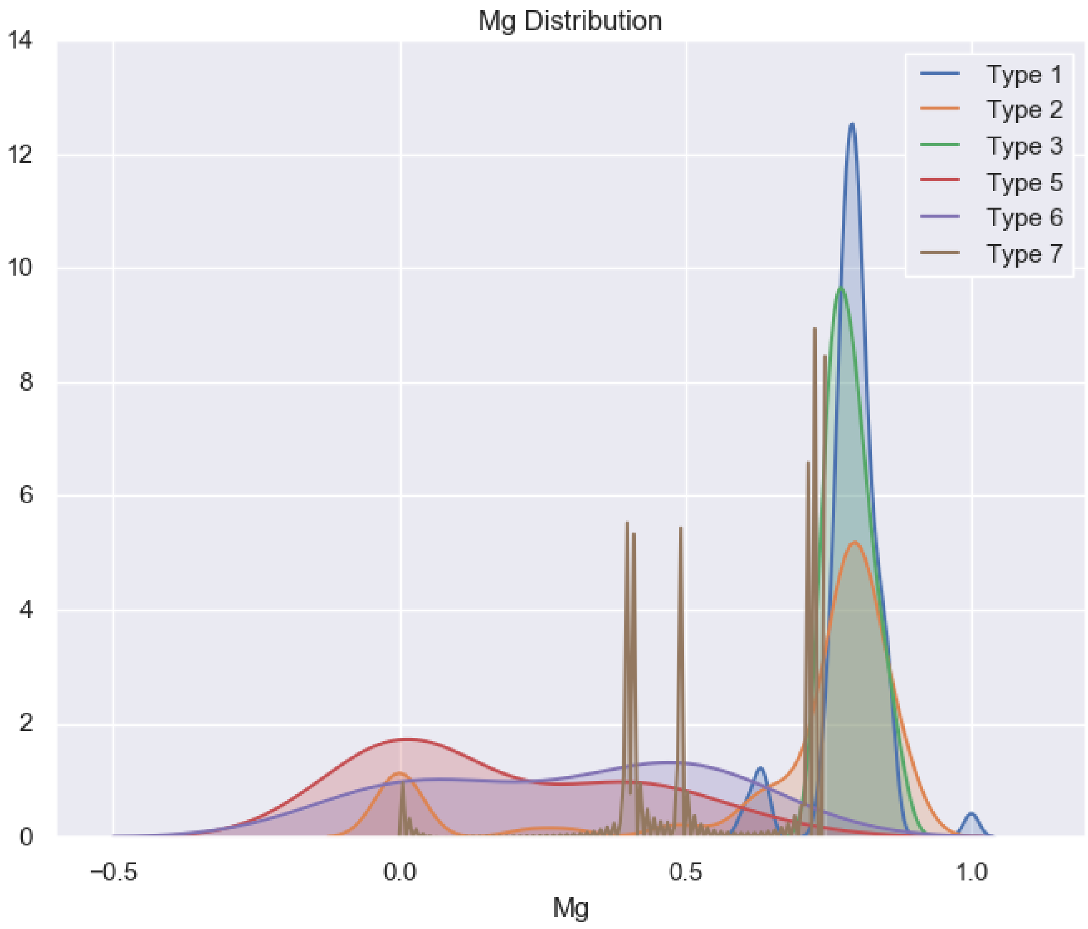
```
In [3]: #%%
        # Ignore division errors
        np.seterr(divide='ignore', invalid='ignore')

        # Load data into pandas DataFrame (skipping 1st ID column)
        df = pd.read_csv('glass.data', header=None,usecols=range(1,11))

        # Name columns according to glass.tag
        df.columns = [ 'RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type' ]
        df.index.names = ['ID']
```

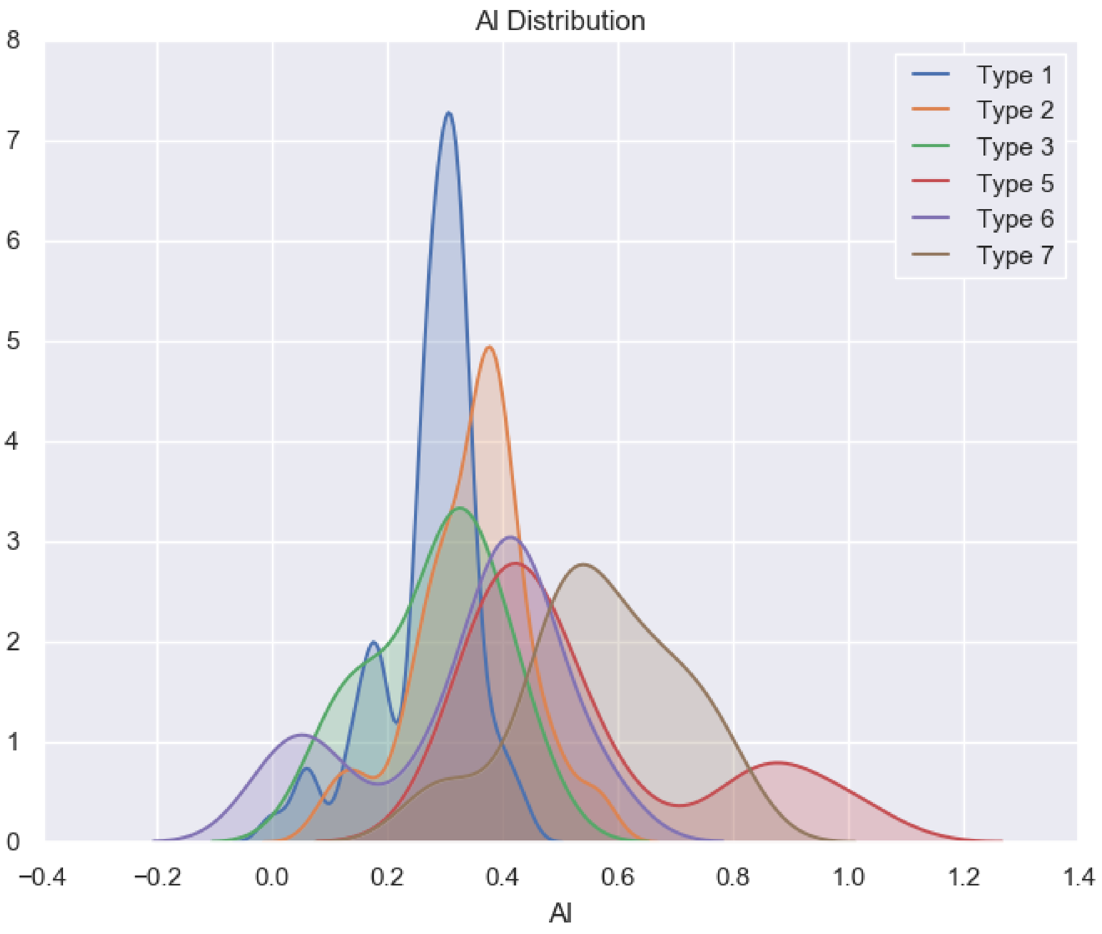
```
In [4]: #%% Scale Data
        from sklearn.preprocessing import MinMaxScaler
        mmsc = MinMaxScaler()
        df.iloc[:,:-1] = mmsc.fit_transform(df.iloc[:,:-1])
```

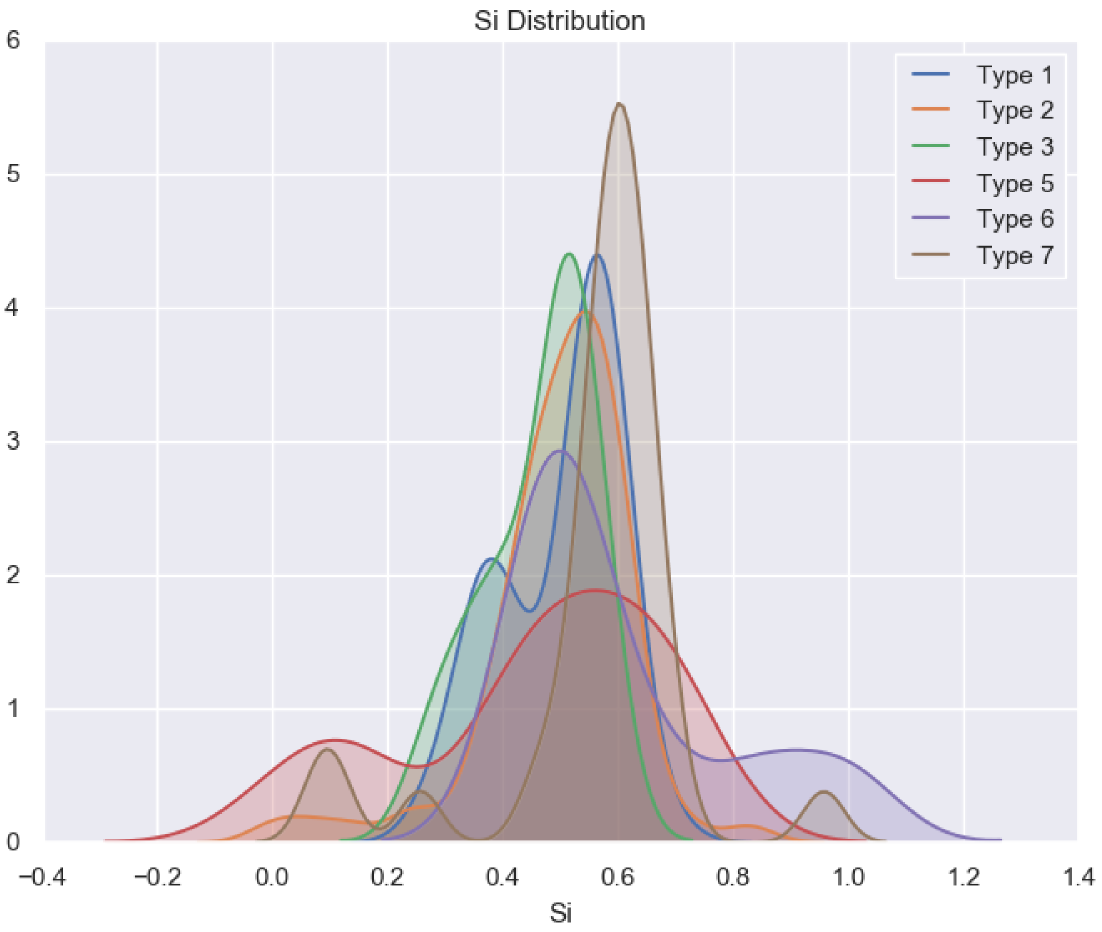
```
In [5]: #%% Plot type distributions for each element
        trange = [1,2,3,5,6,7]
        for i in range(0,9):
            plot_data = df.iloc[:,int(i)] # i-th column
            for tp in trange:
                if np.cov(plot_data[df.Type==tp]) > 0.0:
              sns.kdeplot(plot_data[df.Type==tp], label='Type %d'%tp, shade=True)
                else:
                    print('\nType %d has a flat %s distribution at %.1f\n'%(\
                            tp,plot_data.name,np.median(plot_data[df.Type==tp])))
            plt.xlabel(plot_data.name);
            plt.title(plot_data.name + ' Distribution');
            plt.show()
```



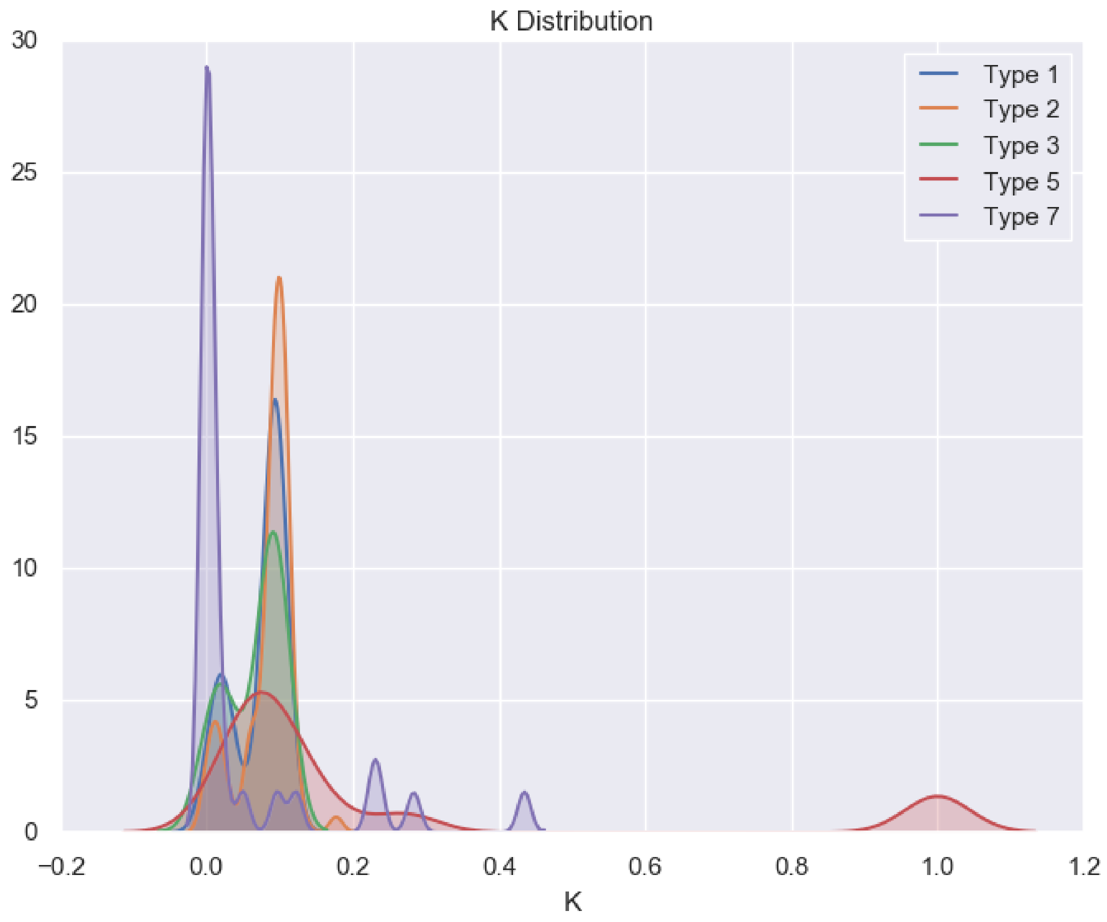

Na Distribution
7
6
5
4
3
2
1
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Na
Type 1
Type 2
Type 3
Type 5
Type 6
Type 7

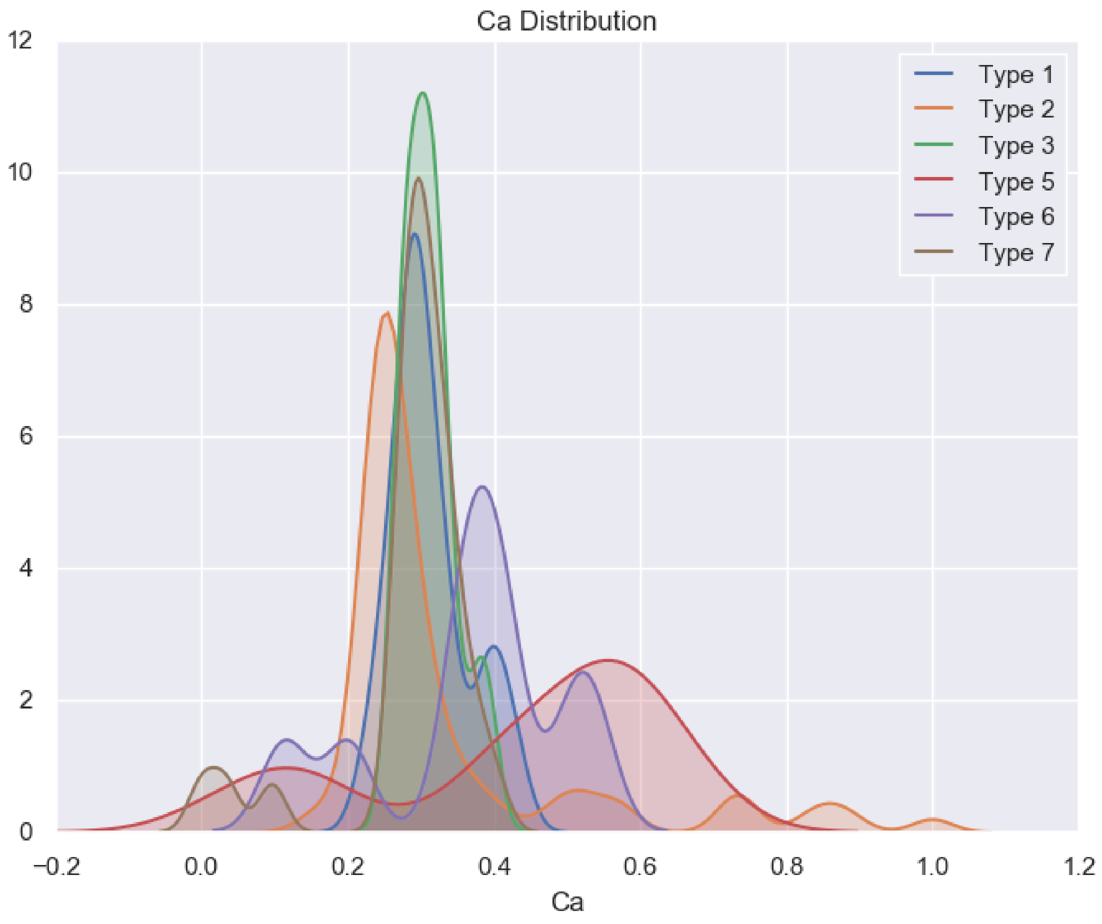

Mg Distribution
14
12
10
8
6
4
2
0
−0.5
0.0
0.5
1.0
Mg
Type 1
Type 2
Type 3
Type 5
Type 6
Type 7


AI Distribution
8
7
6
5
4
3
2
1
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
AI
Type 1
Type 2
Type 3
Type 5
Type 6
Type 7

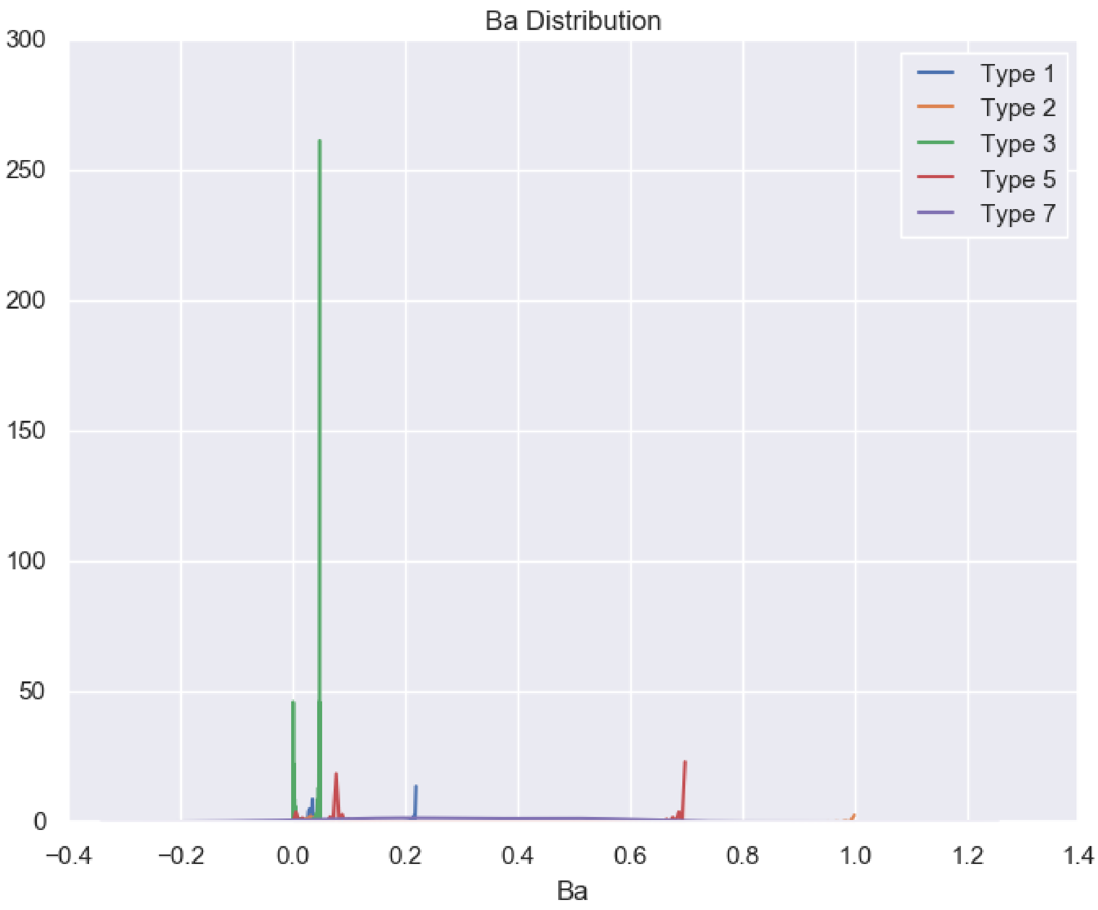

Si Distribution
6
5
4
3
2
1
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Si
Type 1
Type 2
Type 3
Type 5
Type 6
Type 7

```
Type 6 has a flat K distribution at 0.0
```

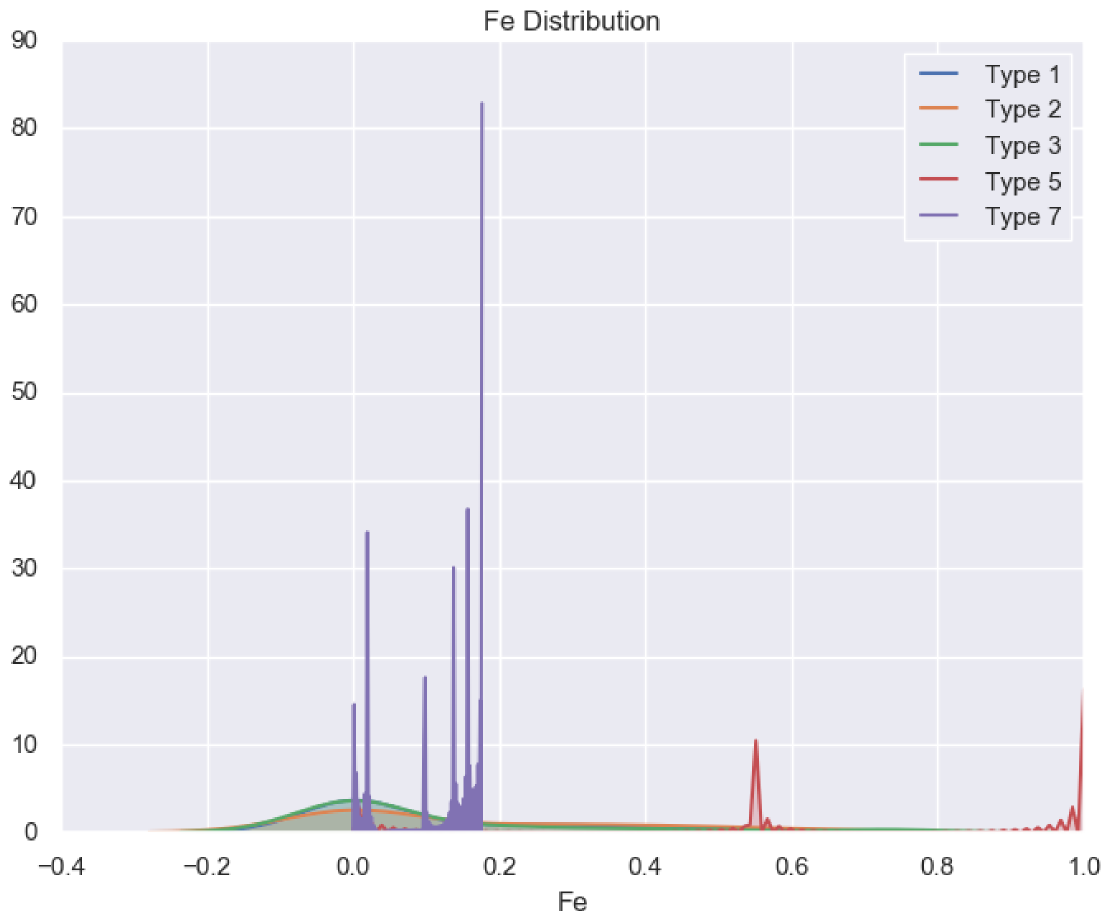



Ca Distribution
12
10
8
6
4
2
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Ca
Type 1
Type 2
Type 3
Type 5
Type 6
Type 7

```
Type 6 has a flat Ba distribution at 0.0
```



```
Type 6 has a flat Fe distribution at 0.0
```



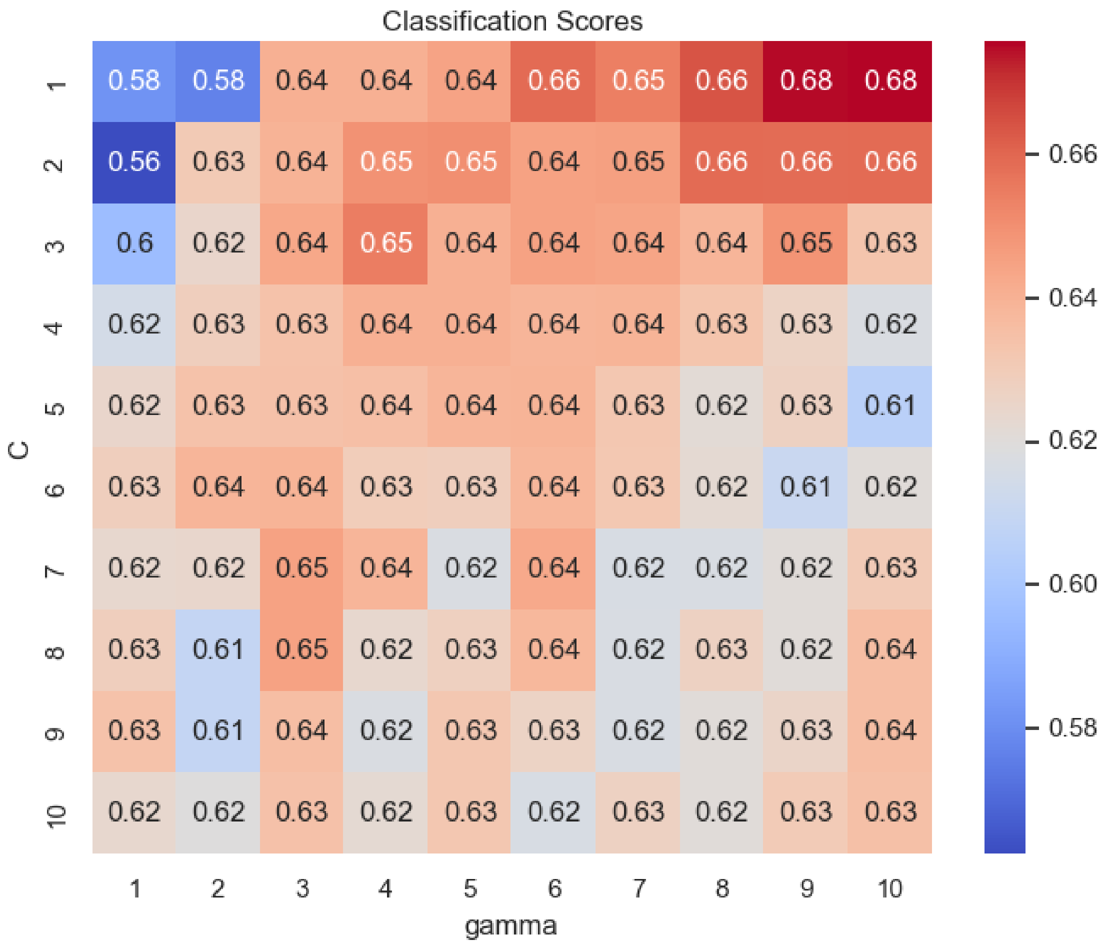
```
In [6]: #%% Type 6 elements
        # From the previous loop it appears that Type 6 has 0.0 quantity of three elements
        type_6 = set(df[df.Type==6].index)

        # Samples where Fe, Ba and K are 0
        check = (df.Fe<=0.0) & (df.Ba<=0.0) & (df.K<=0.0)
        check_list = set(df.Type[check].index)

        # Observe that the above list contains, among a few others, all of Type 6 samples
        print('Type 6 Classification:\n')
        if type_6.issubset(check_list):
            print('Recall: 100%\n')
        print("Precision: {0:.2f}%\n".format((1-(len(check_list)-len(type_6))/len(df))*100))

        # Remove samples classified as Type 6
        df = df[~check.values]
```

```
Type 6 Classification:


Recall: 100%

Precision: 97.66%
```

```
In [7]: #%% Classify remaining Types
        #y = labels
        y = df.iloc[:,-1].values

        #X = data
        X = df.iloc[:,:-1].values

        from sklearn.decomposition import PCA
        pca = PCA()
        X = PCA(n_components=3).fit_transform(X)

        from sklearn.svm import SVC
        from sklearn.model_selection import cross_val_score
```

```
In [8]: #%%Check classification score for C, gamma value pairs between 1 and 10
        C_range = np.arange(1,11)
        gamma_range = np.arange(1,11)
        k_scores = []
        for C in C_range:
            for gamma in gamma_range:
                clf = SVC(C=C, gamma=gamma)
                scores = cross_val_score(clf, X, y, cv=10)
                k_scores.append(scores.mean())
        #%% Reshape to 10x10 matrix
        k_scores = np.array(k_scores).reshape(10,10)
```

```
In [9]: #%%
        # Show scoring matrix as image
        dfk = pd.DataFrame(k_scores)
        dfk.index = C_range
        dfk.columns = gamma_range
        ax = sns.heatmap(dfk,cmap='coolwarm',annot=True)
        ax.set(title='Classification Scores',xlabel='gamma',ylabel='C')
        plt.show()
```



```
In [10]: #%%
         # Choose best parameters
         best_params = np.unravel_index(k_scores.argmax(), k_scores.shape)
         C_best = C_range[best_params[0]]
         gamma_best = gamma_range[best_params[1]]
```

```
In [11]: #%% New model with best parameters
         clf = SVC(C=10-C_best+1, gamma=gamma_best)
```

```
In [12]: #%% Evaluate
         scores = cross_val_score(clf, df.iloc[:,:-1].values , df.iloc[:,-1].values , cv=10)
         print ('Cross Validation', np.mean(scores))
```

```
Cross Validation 0.690495356037
```