1. import neccessery libraries

In [19]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

2. Import data

In [2]:
```
glass_data = pd.read_csv('glass.csv')
glass_data
```

Out[2]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 10 columns

In [3]:
```
glass_data.loc[glass_data['Type'] == 1, 'Type'] = 'building_windows_float_p
glass_data.loc[glass_data['Type'] == 2, 'Type'] = 'building_windows_non_flo
glass_data.loc[glass_data['Type'] == 3, 'Type'] = 'vehicle_windows_float_pr
glass_data.loc[glass_data['Type'] == 4, 'Type'] = 'vehicle_windows_non_floa
glass_data.loc[glass_data['Type'] == 5, 'Type'] = 'containers'
glass_data.loc[glass_data['Type'] == 6, 'Type'] = 'tableware'
glass_data.loc[glass_data['Type'] == 7, 'Type'] = 'headlamps'
```

In [4]:
```
glass_data.head()
```

Out[4]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	building_windows_float_processed
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	building_windows_float_processed
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	building_windows_float_processed
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	building_windows_float_processed
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	building_windows_float_processed

3.Data understanding

In [6]:

```
glass_data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
RI	214.0	1.518365	0.003037	1.51115	1.516522	1.51768	1.519157	1.53393
Na	214.0	13.407850	0.816604	10.73000	12.907500	13.30000	13.825000	17.38000
Mg	214.0	2.684533	1.442408	0.00000	2.115000	3.48000	3.600000	4.49000
Al	214.0	1.444907	0.499270	0.29000	1.190000	1.36000	1.630000	3.50000
Si	214.0	72.650935	0.774546	69.81000	72.280000	72.79000	73.087500	75.41000
K	214.0	0.497056	0.652192	0.00000	0.122500	0.55500	0.610000	6.21000
Ca	214.0	8.956963	1.423153	5.43000	8.240000	8.60000	9.172500	16.19000
Ba	214.0	0.175047	0.497219	0.00000	0.000000	0.00000	0.000000	3.15000
Fe	214.0	0.057009	0.097439	0.00000	0.000000	0.00000	0.100000	0.51000

In [7]:

```
glass_data.shape
```

Out[7]:

```
(214, 10)
```

In [8]:

```
glass_data.dtypes
```

Out[8]:

```
RI       float64
Na       float64
Mg       float64
Al       float64
Si       float64
K        float64
Ca       float64
Ba       float64
Fe       float64
Type      object
dtype: object
```

In [9]:

```
glass_data.isna().sum()
```

Out[9]:

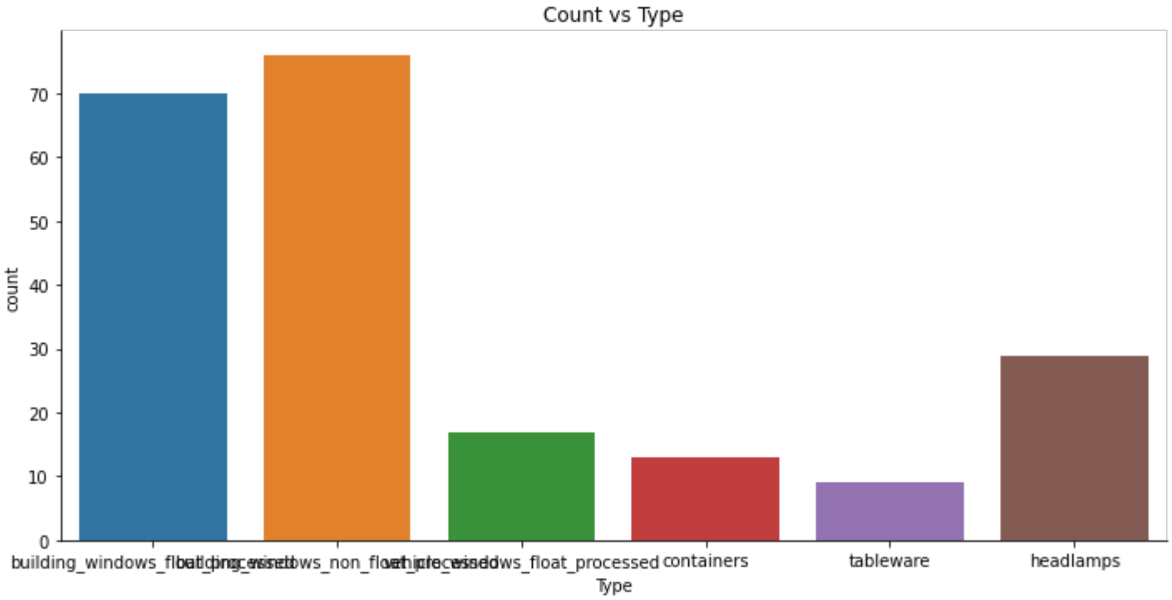
```
RI       0
Na       0
Mg       0
Al       0
```

```
Si       0
K        0
Ca       0
Ba       0
Fe       0
Type     0
dtype: int64
```

In [20]:

```
plt.figure(figsize=(12,8))
sns.factorplot('Type',data=glass_data,kind='count',size=5,aspect=2)
plt.title('Count vs Type' )
plt.show()
```

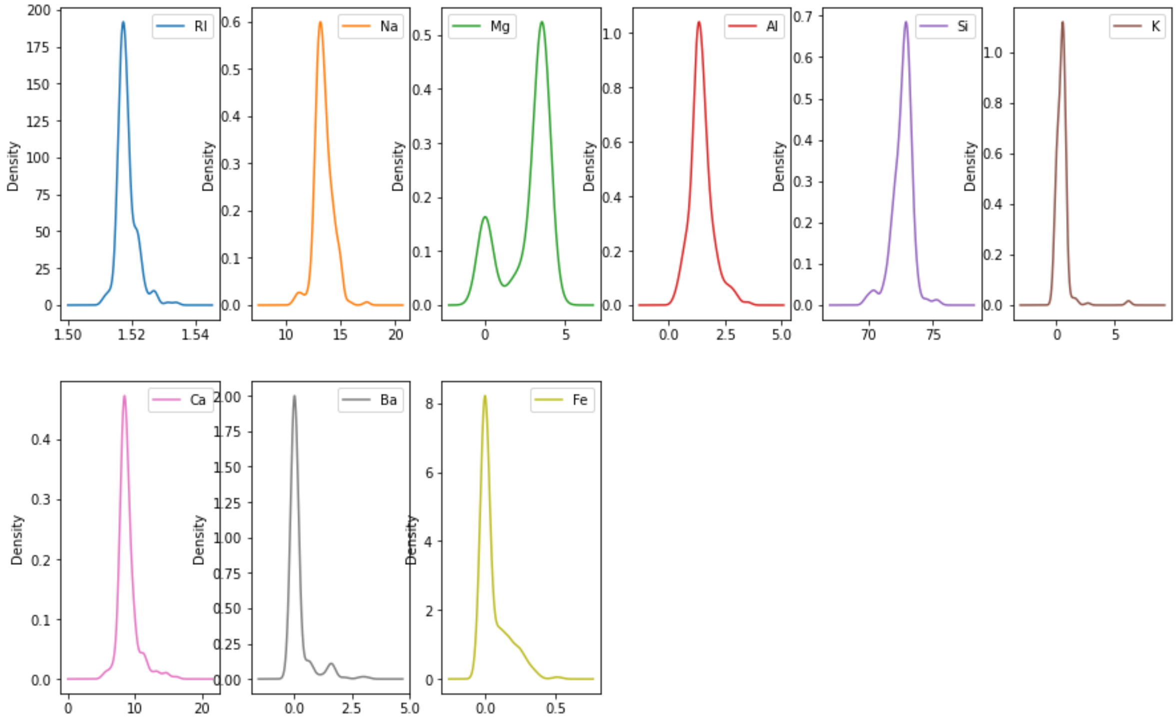
```
<Figure size 864x576 with 0 Axes>
```



As shown in the graphs above, majority of the glass types are building_windows_float_processed and building_windows_non_float_processed, followed by headlamps

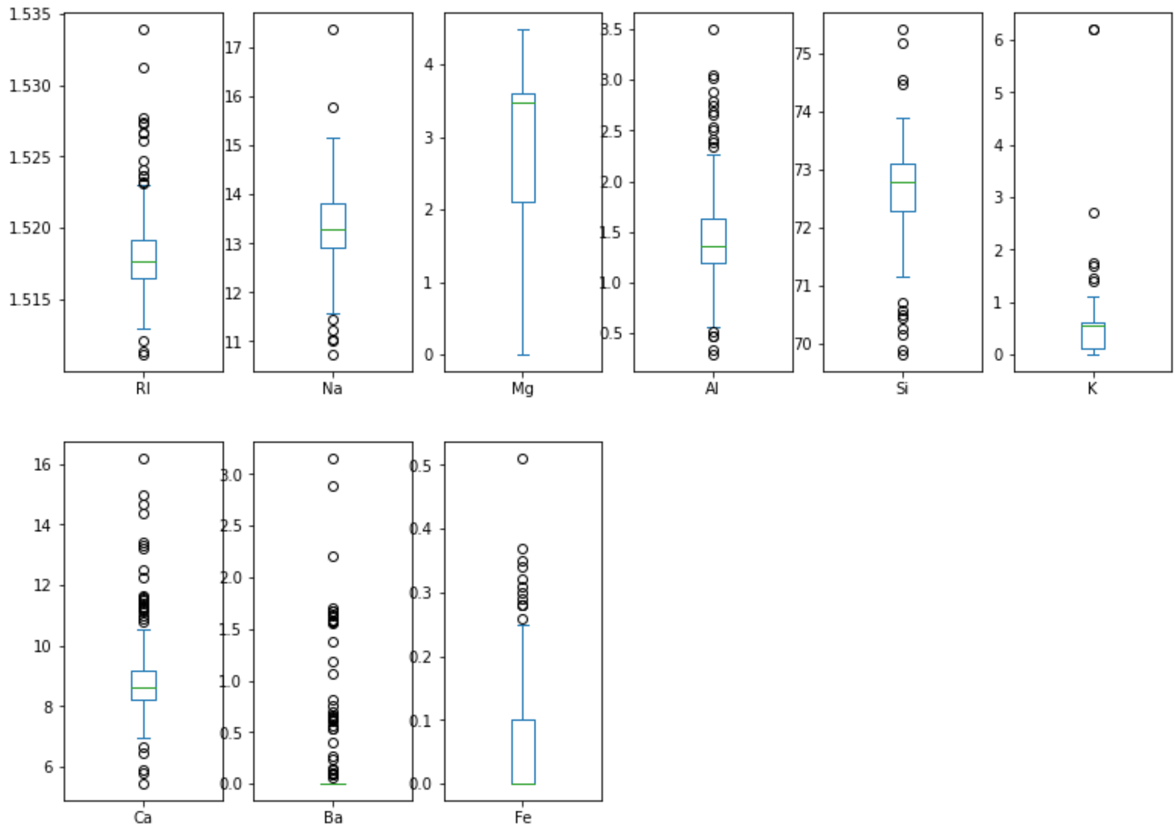
In [28]:

```
glass_data.plot(kind='density', subplots=True, layout=(5,6),figsize=(15,25)
plt.show()
```



In [27]:

```
glass_data.plot(kind='box', subplots=True, layout=(5,6),figsize=(13,25),sha
plt.show()
```



4. Finding correlation

In [29]:

```
cor = glass_data.corr(method='pearson')
```

In [31]:

```
cor.style.background_gradient(cmap='rainbow')
```

Out[31]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	
RI	1.000000	-0.191885	-0.122274	-0.407326	-0.542052	-0.289833	0.810403	-0.000386	0.1430
Na	-0.191885	1.000000	-0.273732	0.156794	-0.069809	-0.266087	-0.275442	0.326603	-0.2413
Mg	-0.122274	-0.273732	1.000000	-0.481799	-0.165927	0.005396	-0.443750	-0.492262	0.0830
Al	-0.407326	0.156794	-0.481799	1.000000	-0.005524	0.325958	-0.259592	0.479404	-0.0744
Si	-0.542052	-0.069809	-0.165927	-0.005524	1.000000	-0.193331	-0.208732	-0.102151	-0.0942
K	-0.289833	-0.266087	0.005396	0.325958	-0.193331	1.000000	-0.317836	-0.042618	-0.0077
Ca	0.810403	-0.275442	-0.443750	-0.259592	-0.208732	-0.317836	1.000000	-0.112841	0.1249
Ba	-0.000386	0.326603	-0.492262	0.479404	-0.102151	-0.042618	-0.112841	1.000000	-0.0586
Fe	0.143010	-0.241346	0.083060	-0.074402	-0.094201	-0.007719	0.124968	-0.058692	1.0000

As seen in the above graph, there is a high correlation exists between some of the variables. We can use KNN to reduce the hight correlated variables

5. KNN

5.1 Finding optimal number of K

In [32]:

```
X = np.array(glass_data.iloc[:,3:5])
y = np.array(glass_data['Type'])
```

In [33]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [34]:

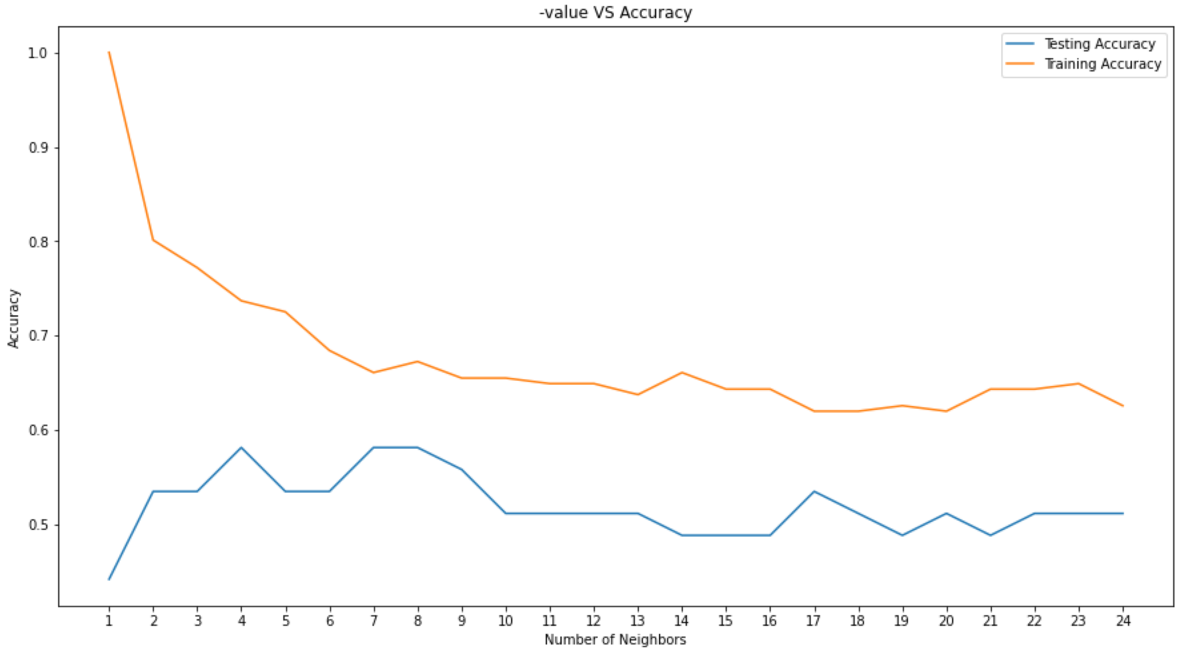
```
k_values = np.arange(1,25)
train_accuracy = []
test_accuracy = []
```

In [35]:

```
for i, k in enumerate(k_values):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train,y_train)
    train_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
```

In [36]:

```
plt.figure(figsize=[15,8])
plt.plot(k_values, test_accuracy, label = 'Testing Accuracy')
plt.plot(k_values, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.show()
```



k=4 produces the most accurate results

5.2 Applying the algorithm

In [40]:
```
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train,y_train)
y_pred_knn = knn.predict(X_test)
```

In [41]:
```
scores = []
cv_scores = []
```

In [43]:
```
score = accuracy_score(y_pred_knn,y_test)
scores.append(score)
```

In [44]:
```
score_knn=cross_val_score(knn, X,y, cv=10)
```

In [45]:
```
score_knn.mean()
```

Out[45]:
```
0.6127705627705629
```

In [46]:
```
score_knn.std()*2
```

Out[46]:
```
0.23547117559816877
```

In [47]:
```
cv_score = score_knn.mean()
```

In [48]:
```
cv_scores.append(cv_score)
```

In [49]:
```
cv_scores
```

Out[49]:

```
[0.6127705627705629]
```

Support Vector Machine Accuracy: 0.61 (+/- 0.22)

In [ ]: