In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv('glass.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type of glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [4]:

```
df.tail()
```

Out[4]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type of glass
209	210	1.51623	14.14	0.0	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.0	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.0	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.0	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.0	2.08	73.36	0.00	8.62	1.67	0.0	7

In [5]:

```
df.shape
```

Out[5]:

```
(214, 11)
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['Id', 'RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe',
       'Type of glass'],
      dtype='object')
```

In [7]:

```
df.duplicated().sum()
```

Out[7]:

```
0
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Id               0
RI               0
Na               0
Mg               0
Al               0
Si               0
K                0
Ca               0
Ba               0
Fe               0
Type of glass    0
dtype: int64
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             214 non-null    int64  
 1   RI             214 non-null    float64
 2   Na             214 non-null    float64
 3   Mg             214 non-null    float64
 4   Al             214 non-null    float64
 5   Si             214 non-null    float64
 6   K              214 non-null    float64
 7   Ca             214 non-null    float64
 8   Ba             214 non-null    float64
 9   Fe             214 non-null    float64
 10  Type of glass  214 non-null    int64  
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

In [10]:

```
df.describe()
```

Out[10]:

	Id	RI	Na	Mg	Al	Si	K	
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.0
mean	107.500000	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.9
std	61.920648	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.4
min	1.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.4
25%	54.250000	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.2
50%	107.500000	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.6
75%	160.750000	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.1
max	214.000000	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.1

In [11]:

```
df.nunique()
```

Out[11]:

```
Id                214
RI                178
Na                142
Mg                 94
Al                118
Si                133
K                  65
Ca                143
Ba                 34
Fe                 32
Type of glass       6
dtype: int64
```

In [12]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:

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

In [14]:

```
df['Type of glass'].unique()
```

Out[14]:

```
array([1, 2, 3, 5, 6, 7], dtype=int64)
```

In [15]:

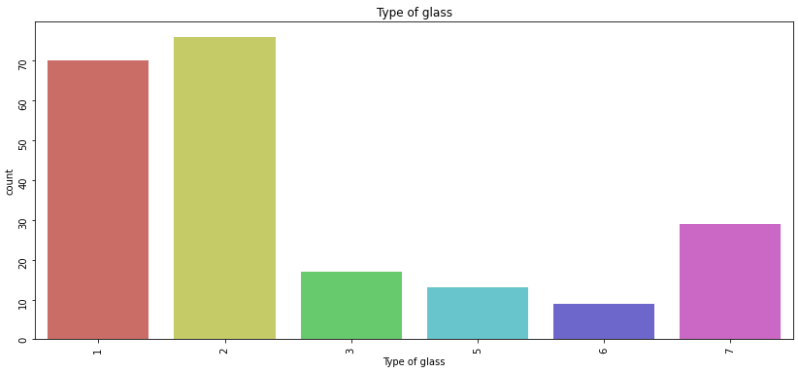
```
df['Type of glass'].value_counts()
```

Out[15]:

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: Type of glass, dtype: int64
```

In [16]:

```
plt.figure(figsize = (14,6))
sns.countplot('Type of glass',data=df, palette = 'hls')
plt.title('Type of glass')
plt.xticks(rotation = 90)
plt.yticks(rotation = 90)
plt.show()
```



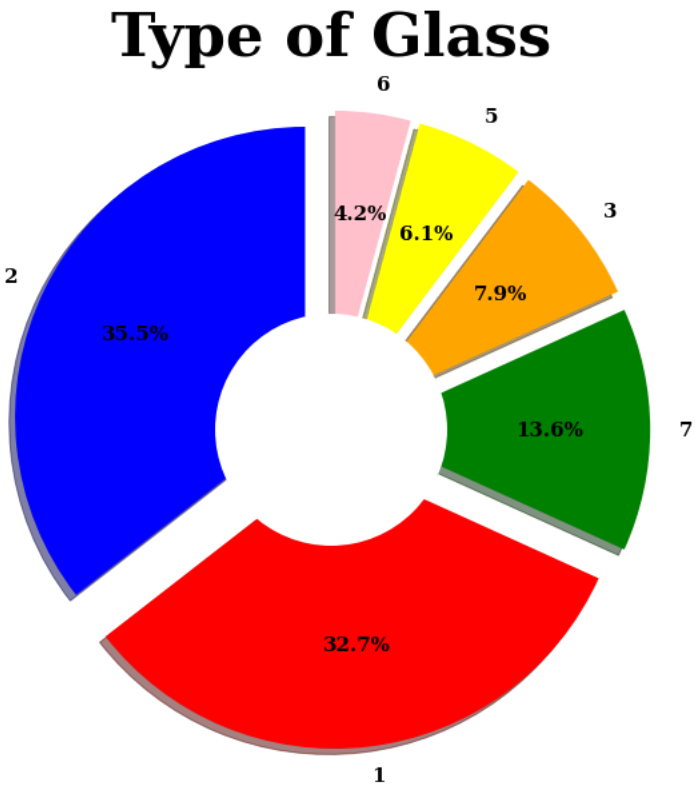
In [17]:

```
glass_data = df['Type of glass'].value_counts()

explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
plt.figure(figsize=(14, 10))
patches, texts, pcts = plt.pie(glass_data,
                               labels = glass_data.index,
                               colors = ['blue', 'red', 'green', 'orange',
                                         'yellow', 'pink'],
                               pctdistance = 0.65,
                               shadow = True,
                               startangle = 90,
                               explode = explode,
                               autopct = '%1.1f%%',
                               textprops={ 'fontsize': 15,
                                           'color': 'black',
                                           'weight': 'bold',
                                           'family': 'serif' })
plt.setp(pcts, color='black')

hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('Type of Glass', size=45, **hfont)

centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```

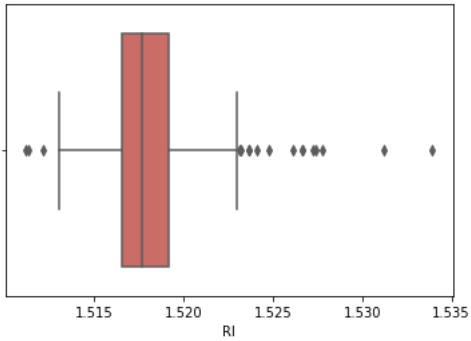


In [18]:

```
df = df.drop('Id', axis = 1)
```

In [19]:

```
for i in df.columns:
    sns.boxplot(df[i], palette = 'hls')
    plt.show()
```



In [20]:

```
df.corr()
```

Out[20]:

	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	0.143
Na	-0.191885	1.000000	-0.273732	0.156794	-0.069809	-0.266087	-0.275442	0.326603	-0.241
Mg	-0.122274	-0.273732	1.000000	-0.481799	-0.165927	0.005396	-0.443750	-0.492262	0.083
Al	-0.407326	0.156794	-0.481799	1.000000	-0.005524	0.325958	-0.259592	0.479404	-0.074
Si	-0.542052	-0.069809	-0.165927	-0.005524	1.000000	-0.193331	-0.208732	-0.102151	-0.094
K	-0.289833	-0.266087	0.005396	0.325958	-0.193331	1.000000	-0.317836	-0.042618	-0.007
Ca	0.810403	-0.275442	-0.443750	-0.259592	-0.208732	-0.317836	1.000000	-0.112841	0.124
Ba	-0.000386	0.326603	-0.492262	0.479404	-0.102151	-0.042618	-0.112841	1.000000	-0.058
Fe	0.143010	-0.241346	0.083060	-0.074402	-0.094201	-0.007719	0.124968	-0.058692	1.000
Type of glass	-0.164237	0.502898	-0.744993	0.598829	0.151565	-0.010054	0.000952	0.575161	-0.188

In [21]:

```
import numpy as np
```

In [22]:

```
corr_matrix = df.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]

# Drop features
df.drop(to_drop, axis=1, inplace=True)
```

In [23]:

```
plt.figure(figsize = (14,6))
sns.heatmap(corr_matrix, annot = True)
plt.show()
```

In [24]:

```
X = df.drop('Type of glass', axis = 1)
y = df['Type of glass']
```

In [25]:

```
from sklearn.model_selection import train_test_split
```

In [26]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [27]:

```
from sklearn import preprocessing
```

In [28]:

```
scaler = preprocessing.RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

In [29]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='entropy', random_state=0)
dt.fit(X_train, y_train)
```

Out[29]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [30]:

```
y_pred_dt = dt.predict(X_test)
```

In [31]:

```
from sklearn.metrics import confusion_matrix
```

In [32]:

```
cm_dt = confusion_matrix(y_test, y_pred_dt)
```

In [33]:

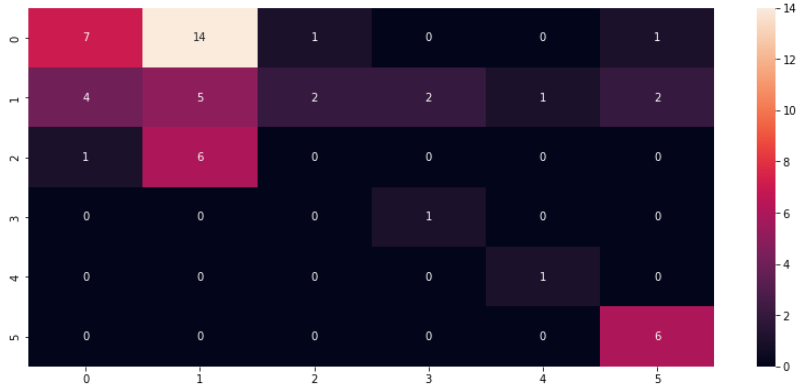
```
cm_dt
```

Out[33]:

```
array([[ 7, 14,  1,  0,  0,  1],
       [ 4,  5,  2,  2,  1,  2],
       [ 1,  6,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  6]], dtype=int64)
```

In [34]:

```
plt.figure(figsize = (14,6))
sns.heatmap(cm_dt, annot = True)
plt.show()
```



In [35]:

```
from sklearn.metrics import classification_report
```

In [36]:

```
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           1       0.58      0.30      0.40        23
           2       0.20      0.31      0.24        16
           3       0.00      0.00      0.00         7
           5       0.33      1.00      0.50         1
           6       0.50      1.00      0.67         1
           7       0.67      1.00      0.80         6

    accuracy                           0.37        54
   macro avg       0.38      0.60      0.44        54
weighted avg       0.40      0.37      0.35        54
```

In [37]:

```
# rejecting decision tree because of low accuracy
```

In [38]:

```
from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier(n_estimators= 10, criterion="entropy")
rf.fit(X_train, y_train)
```

Out[38]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [39]:

```
y_pred_rf = rf.predict(X_test)
```

In [40]:

```
cm_rf = confusion_matrix(y_test, y_pred_rf)
```

In [41]:

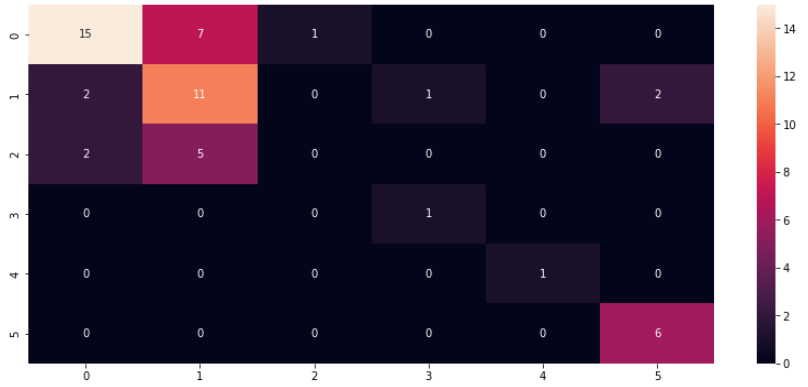
```
cm_rf
```

Out[41]:

```
array([[15,  7,  1,  0,  0,  0],
       [ 2, 11,  0,  1,  0,  2],
       [ 2,  5,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  6]], dtype=int64)
```

In [42]:

```
plt.figure(figsize = (14,6))
sns.heatmap(cm_rf, annot = True)
plt.show()
```



In [43]:

```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           1       0.79      0.65      0.71        23
           2       0.48      0.69      0.56        16
           3       0.00      0.00      0.00         7
           5       0.50      1.00      0.67         1
           6       1.00      1.00      1.00         1
           7       0.75      1.00      0.86         6

    accuracy                           0.63        54
   macro avg       0.59      0.72      0.63        54
weighted avg       0.59      0.63      0.60        54
```

In [44]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [45]:

```
gradient_booster = GradientBoostingClassifier(learning_rate=0.1)
gradient_booster.fit(X_train,y_train)
```

Out[45]:

GradientBoostingClassifier

GradientBoostingClassifier()

In [46]:

```
y_pred_gb = gradient_booster.predict(X_test)
```

In [47]:

```
cm_gb = confusion_matrix(y_test, y_pred_gb)
```

In [48]:

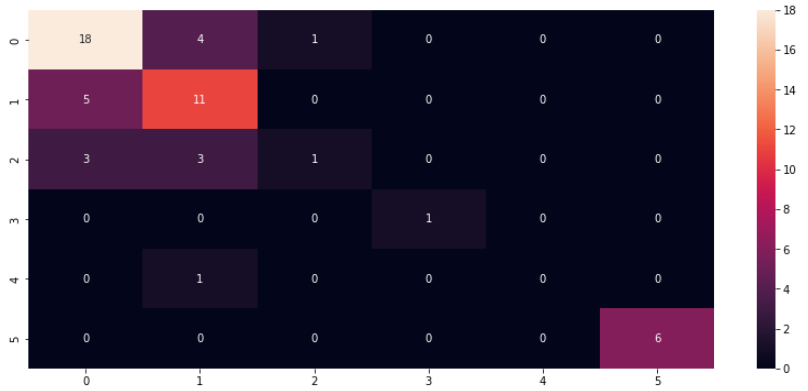
```
cm_gb
```

Out[48]:

```
array([[18,  4,  1,  0,  0,  0],
       [ 5, 11,  0,  0,  0,  0],
       [ 3,  3,  1,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0],
       [ 0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  6]], dtype=int64)
```

In [49]:

```
plt.figure(figsize = (14,6))
sns.heatmap(cm_gb, annot = True)
plt.show()
```



In [50]:

```
print(classification_report(y_test, y_pred_gb))
```

```
              precision    recall  f1-score   support

           1       0.69      0.78      0.73        23
           2       0.58      0.69      0.63        16
           3       0.50      0.14      0.22         7
           5       1.00      1.00      1.00         1
           6       0.00      0.00      0.00         1
           7       1.00      1.00      1.00         6

    accuracy                           0.69        54
   macro avg       0.63      0.60      0.60        54
weighted avg       0.66      0.69      0.66        54
```