KNN Assignment

Data Set - Glass

1. Import Necessary libraries

In [1]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns

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

3. Data Understanding

3.1 Initial Analysis :

In [3]:
```
glass_data.head()
```

Out[3]:

	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	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

In [4]:
```
glass_data.shape
```

Out[4]:
```
(214, 10)
```

In [5]:
```
glass_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   RI      214 non-null    float64
 1   Na      214 non-null    float64
 2   Mg      214 non-null    float64
 3   Al      214 non-null    float64
 4   Si      214 non-null    float64
 5   K       214 non-null    float64
 6   Ca      214 non-null    float64
 7   Ba      214 non-null    float64
 8   Fe      214 non-null    float64
 9   Type    214 non-null    int64
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
```

In [6]:
```
glass_data.isna().sum()
```

Out[6]:
```
RI      0
Na      0
Mg      0
Al      0
Si      0
K       0
Ca      0
Ba      0
Fe      0
Type    0
dtype: int64
```

In [7]:
```
glass_data.describe()
```

Out[7]:

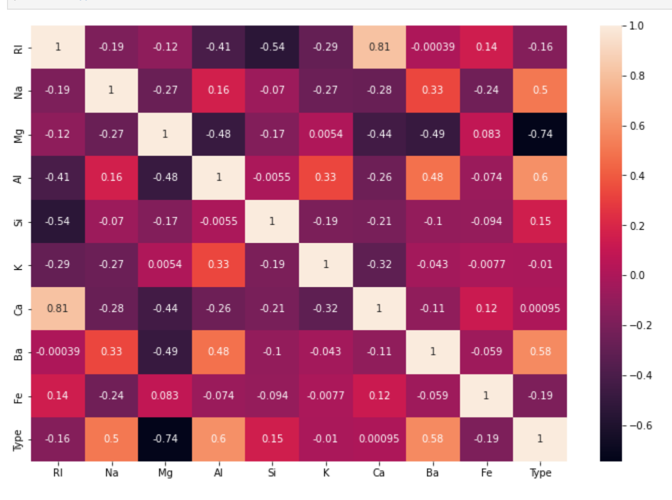
	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009	2.780374
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439	2.103739
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000	1.000000
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000	1.000000
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000	2.000000
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000	3.000000
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000	7.000000

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      int64
dtype: object
```

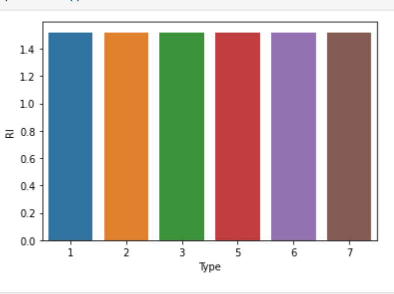
3.2 Correlation Matrix :

In [9]:
```
plt.figure(figsize = (12,8))
sns.heatmap(glass_data.corr(),annot = True)
plt.show()
```

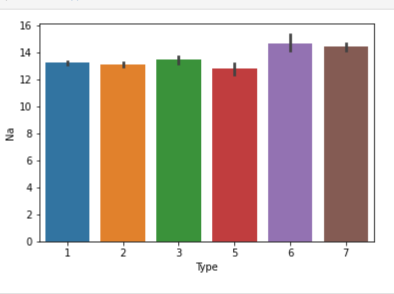


4. Perform Assumption Check

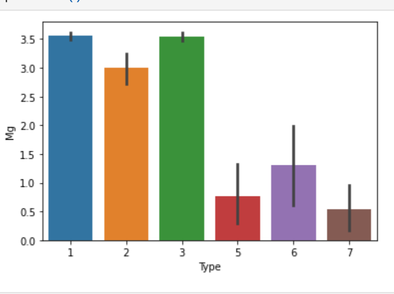
In [10]:
```
sns.barplot(x = 'Type' , y = 'RI',data = glass_data)
plt.show()
```



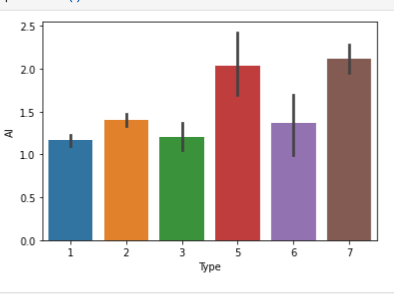
In [11]:
```
sns.barplot(x = 'Type' , y = 'Na',data = glass_data)
plt.show()
```



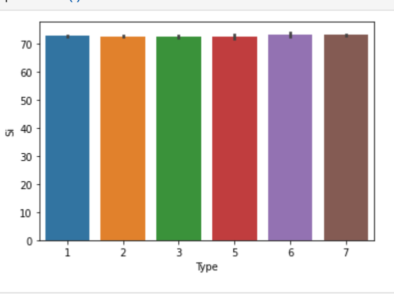
In [12]:
```
sns.barplot(x = 'Type' , y = 'Mg',data = glass_data)
plt.show()
```



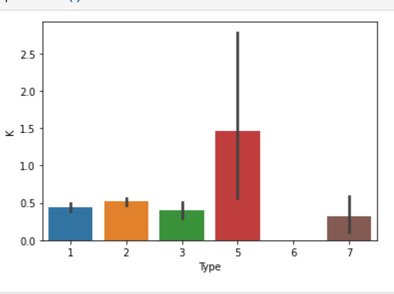
In [13]:
```
sns.barplot(x = 'Type' , y = 'Al',data = glass_data)
plt.show()
```



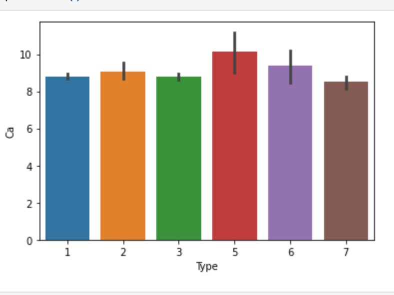
In [14]:
```
sns.barplot(x = 'Type' , y = 'Si',data = glass_data)
plt.show()
```



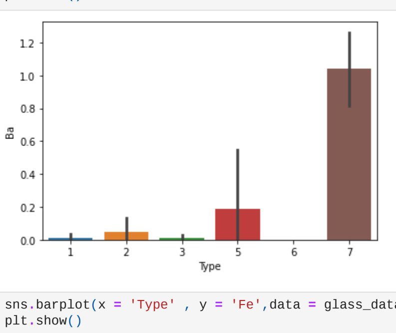
In [15]:
```
sns.barplot(x = 'Type' , y = 'K',data = glass_data)
plt.show()
```



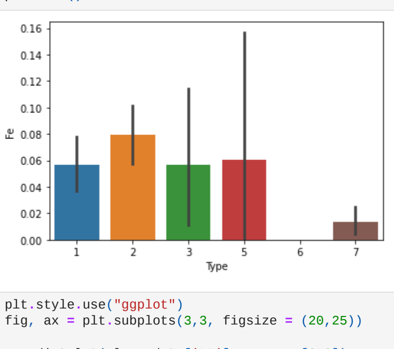
In [16]:
```
sns.barplot(x = 'Type' , y = 'Ca',data = glass_data)
plt.show()
```



In [17]:
```
sns.barplot(x = 'Type' , y = 'Ba',data = glass_data)
plt.show()
```



In [18]:
```
sns.barplot(x = 'Type' , y = 'Fe',data = glass_data)
plt.show()
```

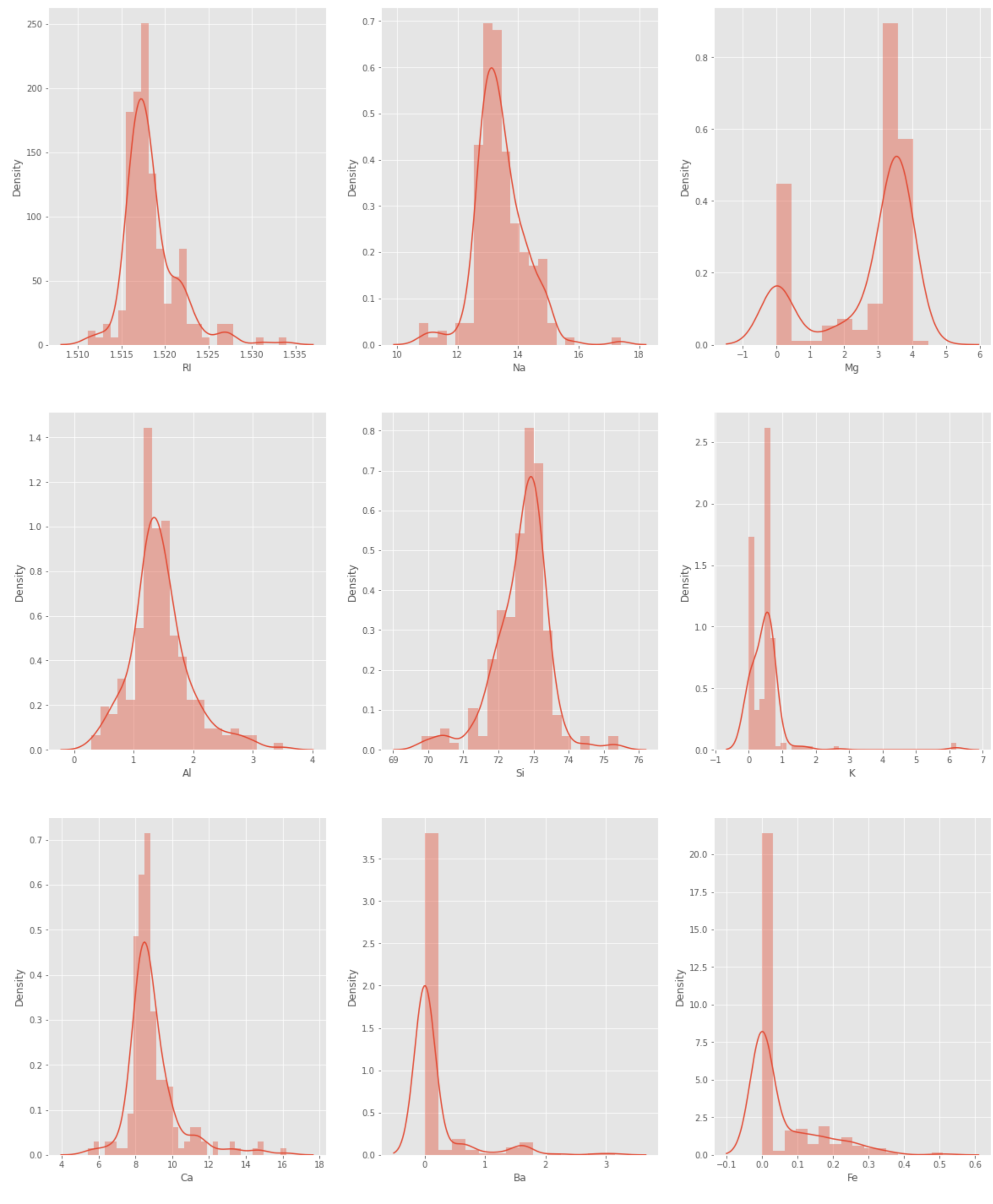


In [19]:
```
plt.style.use("ggplot")
fig, ax = plt.subplots(3,3, figsize = (20,25))

sns.distplot(glass_data['RI'], ax = ax[0,0])
sns.distplot(glass_data['Na'], ax = ax[0,1])
sns.distplot(glass_data['Mg'], ax = ax[0,2])

sns.distplot(glass_data['Al'], ax = ax[1,0])
sns.distplot(glass_data['Si'], ax = ax[1,1])
sns.distplot(glass_data['K'],  ax = ax[1,2])

sns.distplot(glass_data['Ca'], ax = ax[2,0])
sns.distplot(glass_data['Ba'], ax = ax[2,1])
sns.distplot(glass_data['Fe'], ax = ax[2,2])
plt.show()
```



5. Train Test Split

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
X = glass_data.drop('Type', axis = 1)
y = glass_data['Type']
```

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.30,random_state = 42)
```

In [23]:
```
X_train.shape,X_test.shape
```

Out[23]:
```
((149, 9), (65, 9))
```

In [24]:
```
y_test.shape,y_train.shape
```

Out[24]:
```
((65,), (149,))
```

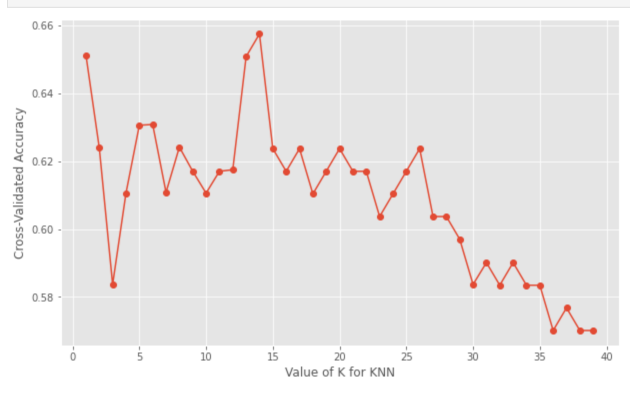
6. KNN (K Neighrest Neighbour Classifier)

In [25]:
```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

In [26]:
```
k_range = range(1, 40)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k)
    train_scores = cross_val_score(knn, X_train, y_train, cv = 5)
    k_scores.append(train_scores.mean())

plt.figure(figsize = (10,6))
plt.plot(k_range, k_scores,marker = "o")
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```



Model :

In [27]:
```
model = KNeighborsClassifier(n_neighbors = 2)
model
```

Out[27]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [28]:
```
model_pred = model.fit(X_train,y_train).predict(X_train)
model_pred
```

Out[28]:
```
array([2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 2, 5, 2, 1, 2, 1, 1, 1, 2,
       2, 1, 1, 2, 1, 3, 2, 2, 2, 2, 1, 7, 1, 1, 2, 2, 1, 1, 7, 6, 1, 1,
       2, 7, 3, 1, 1, 2, 1, 2, 3, 2, 1, 1, 1, 1, 6, 1, 7, 1, 1, 5, 1, 1,
       2, 1, 2, 1, 5, 1, 2, 1, 7, 1, 1, 5, 1, 1, 1, 2, 1, 2, 7, 5, 7, 1,
       2, 1, 1, 2, 1, 2, 7, 7, 1, 1, 1, 1, 2, 2, 1, 5, 2, 7, 1, 6, 7, 1,
       1, 2, 1, 2, 7, 2, 1, 1, 2, 1, 1, 3, 7, 3, 2, 1, 1, 5, 1, 1, 1, 2,
       1, 1, 1, 2, 2, 7, 2, 2, 7, 1, 2, 2, 2, 1, 2, 6, 2], dtype=int64)
```

In [29]:
```
model_accuracy = model.score(X_test, y_test)
model_accuracy
print('Model accuracy is:',model_accuracy)
```

```
Model accuracy is: 0.6615384615384615
```

7. Plot Confusion Matrix

In [30]:
```
from sklearn.metrics import confusion_matrix
```

In [31]:
```
cm_pred = model.predict(X_test)
cm = confusion_matrix(y_test, cm_pred)
cm
```

Out[31]:
```
array([[16,  2,  1,  0,  0,  0],
       [ 7, 12,  1,  1,  2,  0],
       [ 2,  1,  1,  0,  0,  0],
       [ 0,  2,  0,  4,  0,  0],
       [ 0,  1,  0,  0,  2,  0],
       [ 0,  1,  0,  0,  1,  8]], dtype=int64)
```

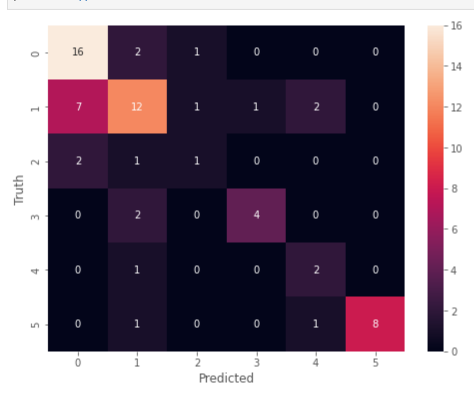
In [32]:
```
pred_df = pd.DataFrame({'Actual' : y_test, 'Predicted' : cm_pred})
pred_df.head()
```

Out[32]:

	Actual	Predicted
9	1	1
197	7	7
66	1	1
191	7	7
117	2	2

In [33]:
```
plt.figure(figsize = (8,6))

sns.heatmap(cm, annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.show()
```



8. Grid Search for Algorithm Tuning

In [34]:
```
from sklearn.model_selection import GridSearchCV
```

In [35]:
```
n_neighbors = np.array(range(1,40))
param_grid = dict(n_neighbors = n_neighbors)
```

In [36]:
```
model = KNeighborsClassifier()
model
```

Out[36]:
```
KNeighborsClassifier()
```

In [37]:
```
grid = GridSearchCV(estimator = model, param_grid = param_grid)
grid.fit(X_train, y_train)
```

Out[37]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39])})
```

In [38]:
```
print('The Best Score Accuracy :',grid.best_score_)
print('The Best Parameter  :',grid.best_params_)
```

```
The Best Score Accuracy : 0.657471264367816
The Best Parameter  : {'n_neighbors': 14}
```

Conclusion :

```
a) For KNN Model Accuracy of Animals  : 0.6615

b) The Grid Best Score Accuracy       : 0.6574
```