In [1]:
```
# Load the dataset.
# Import the necessary libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

In [2]:
```
# Load the dataset.
file_path = 'glass-types.csv'
df = pd.read_csv(file_path, header = None)
```

In [3]:
```
# # Drop the 0th column as it contains only the serial numbers.
df.drop(columns = 0, inplace = True)
```

In [4]:
```
# A Python list containing the suitable column headers as string values. Also, create a Python dictionary as des
column_headers = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'GlassType']
```

In [5]:
```
# Required Python dictionary.
columns_dict = {}
for i in df.columns:
  columns_dict[i] = column_headers[i - 1]
```

In [6]:
```
# Rename the columns.
df.rename(columns_dict, axis = 1, inplace = True)
```

In [7]:
```
# Display the first five rows of the data-frame.
print(df.head(), "\n")
```

```
       RI     Na    Mg    Al     Si     K    Ca   Ba   Fe  GlassType
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 [8]:
```
# Get the information about the dataset.
print(df.info(), "\n")
```

```
<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   GlassType  214 non-null    int64  
dtypes: float64(9), int64(1)
memory usage: 16.8 KB
None
```

In [9]:
```
# Get the count of each glass-type sample in the dataset.
print(df['GlassType'].value_counts(), "\n")
```

```
GlassType
2    76
1    70
7    29
3    17
5    13
6     9
Name: count, dtype: int64
```

In [10]:
```
# Get the percentage of each glass-type sample in the dataset.
round(df['GlassType'].value_counts() * 100 / df.shape[0], 2)
```

Out[10]:
```
GlassType
2    35.51
1    32.71
7    13.55
3     7.94
5     6.07
6     4.21
Name: count, dtype: float64
```

In [11]:
```
# Create separate data-frames for training and testing the model.
from sklearn.model_selection import train_test_split
```

In [12]:
```
# Creating the features data-frame holding all the columns except the last column
x = df.iloc[:, :-1]
print(f"First five rows of the features data-frame:\n{x.head()}\n")
```

```
First five rows of the features data-frame:
        RI     Na    Mg    Al     Si     K    Ca   Ba   Fe
0  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.0  0.0
1  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0
2  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0
3  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0
4  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0
```

In [13]:
```
# Creating the target series that holds last column 'RainTomorrow'
y = df['GlassType']
print(f"First five rows of the GlassType column:\n{y.head()}")
```

```
First five rows of the GlassType column:
0    1
1    1
2    1
3    1
4    1
Name: GlassType, dtype: int64
```

In [14]:
```
# Splitting the train and test sets using the 'train_test_split()' function.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

In [15]:
```
#  Build a random forest classifier model to predict different glass-types.
# Import the 'RandomForestClassifier' module.
from sklearn.ensemble import RandomForestClassifier
```

In [16]:
```
# Create an object of the 'RandomForestClassifier' class and store it in the 'rf_clf' variable.
rf_clf = RandomForestClassifier()
```

In [17]:
```
# Call the 'fit()' function on the 'RandomForestClassifier' object with 'x_train' and 'y_train' as inputs.
rf_clf.fit(x_train, y_train)
```

Out[17]:

▾ RandomForestClassifier

```
RandomForestClassifier()
```

In [18]:
```
# Call the 'score()' function with 'x_train' and 'y_train' as inputs to check the accuracy score of the model.
rf_clf.score(x_train, y_train)
```

Out[18]:
```
1.0
```

In [19]:
```
#  Make predictions on the train set and print the count of each of the classes predicted.
rf_y_train_pred = pd.Series(rf_clf.predict(x_train))
rf_y_train_pred.value_counts()
```

Out[19]:
```
2    53
1    51
7    19
3    13
5     7
6     6
Name: count, dtype: int64
```

In [20]:
```
#  Apply the 'SMOTE()' function to balance the training data.

# Import the 'SMOTE' module from the 'imblearn.over_sampling' library.
from imblearn.over_sampling import SMOTE
```

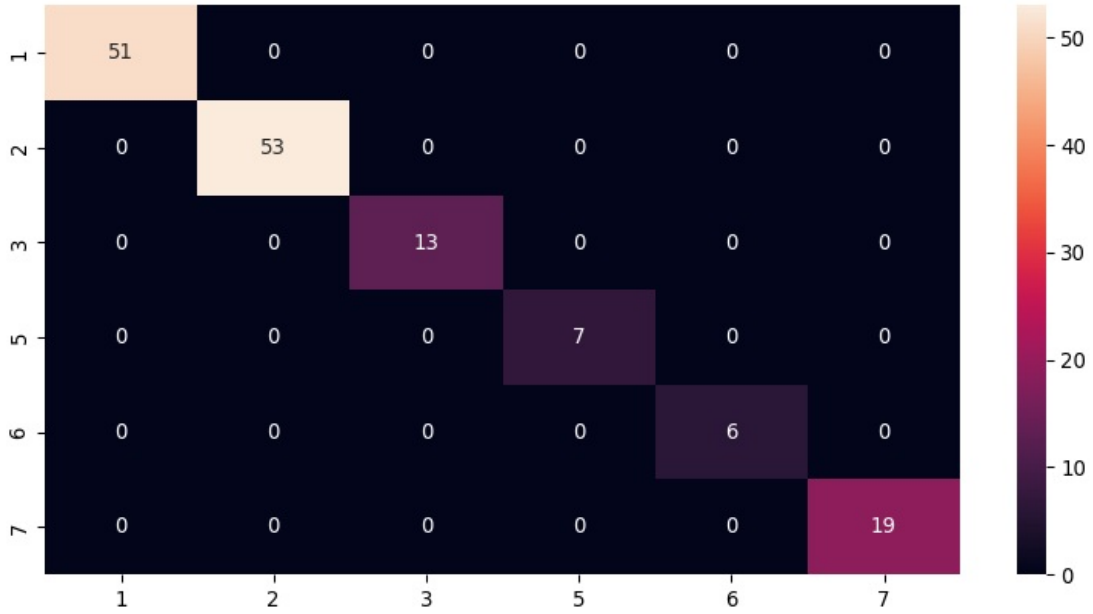
In [21]:
```
# Call the 'SMOTE()' function and store it in the 'smote' variable.
smote = SMOTE(sampling_strategy = "all", random_state = 42)
```

In [22]:
```
# Call the 'fit_sample()' function with 'x_train' and 'y_train' datasets as inputs.
x_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
```

In [23]:
```
#  Create the confusion matrix between the actual and the predicted values for the train set.
from sklearn.metrics import confusion_matrix, classification_report
labels = pd.Series(y_train_resampled).sort_values(ascending = True).unique()
rf_train_conf_matrix = confusion_matrix(y_train, rf_y_train_pred)
```

In [24]:
```
# Create a Pandas DataFrame object for the confusion matrix created above labelled with the classes.
rf_train_cm_df = pd.DataFrame(rf_train_conf_matrix, columns = labels, index = labels)
```

In [25]:
```
# Create a heatmap for the confusion matrix.
plt.figure(figsize = (10, 5), dpi = 96)
sns.heatmap(rf_train_cm_df, annot = True)
plt.show()
```

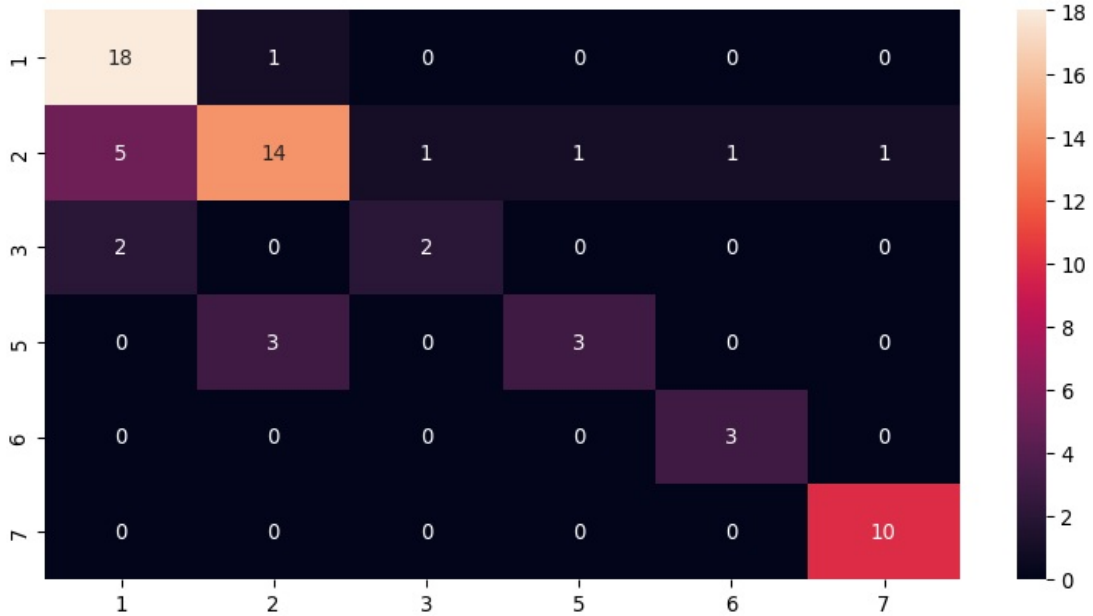


In [26]:
```
# Create the confusion matrix between the actual and predicted values for the test set.
rf_y_test_pred = pd.Series(rf_clf.predict(x_test))
```

In [27]:
```
rf_test_conf_matrix = confusion_matrix(y_test, rf_y_test_pred)
```

In [28]:
```
# Create a Pandas DataFrame object for the confusion matrix created above labelled with the classes.
rf_test_cm_df = pd.DataFrame(rf_test_conf_matrix, columns = labels, index = labels)
```

In [29]:
```
# Create a heatmap for the confusion matrix.
plt.figure(figsize = (10, 5), dpi = 96)
sns.heatmap(rf_test_cm_df, annot = True)
plt.show()
```



In [30]:
```
# Print the classification report for the test set.
print(classification_report(y_test, rf_y_test_pred))
```

```
              precision    recall  f1-score   support

           1       0.72      0.95      0.82        19
           2       0.78      0.61      0.68        23
           3       0.67      0.50      0.57         4
           5       0.75      0.50      0.60         6
           6       0.75      1.00      0.86         3
           7       0.91      1.00      0.95        10

    accuracy                           0.77        65
   macro avg       0.76      0.76      0.75        65
weighted avg       0.77      0.77      0.76        65
```

In [31]:
```
#  Build a random forest classifier model on the resampled train set.
rf_clf_res = RandomForestClassifier()
```

In [32]:
```
# Call the 'fit()' function on the 'RandomForestClassifier' object with 'x_train' and 'y_train' as inputs.
rf_clf_res.fit(x_train_resampled, y_train_resampled)
```

Out[32]:
```
RandomForestClassifier()
```

In [33]:
```
# Call the 'score()' function with 'x_train' and 'y_train' as inputs to check the accuracy score of the model.
rf_clf_res.score(x_train_resampled, y_train_resampled)
```

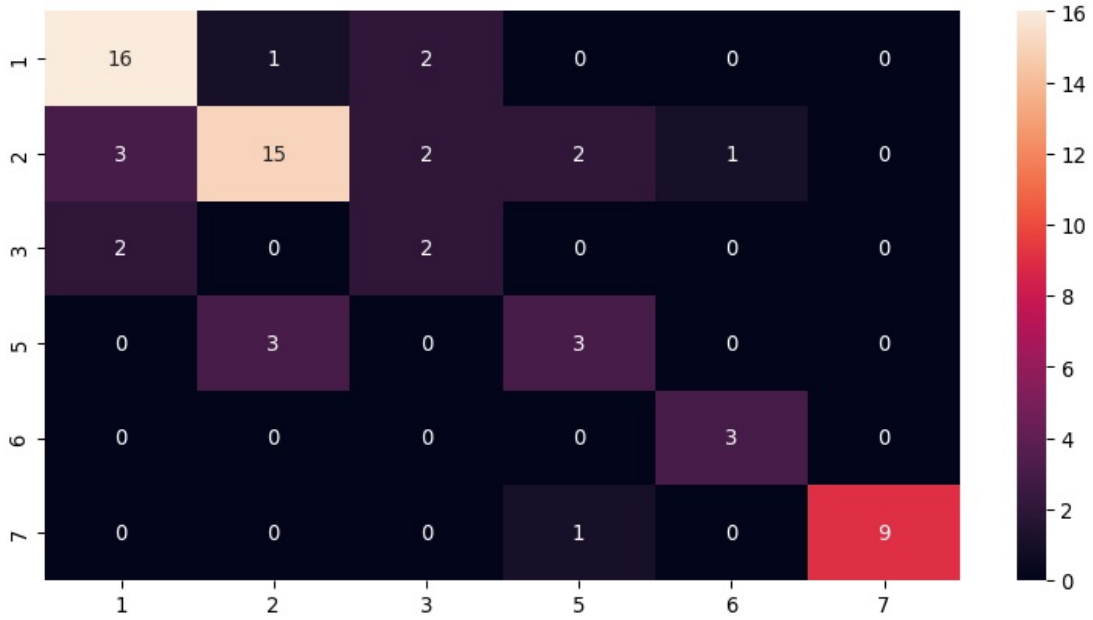
Out[33]: 1.0

In [34]:
```
# Create a confusion matrix on the test set directly.
# Get the predicted labels on the test set obtained from the RFC model built on the resample train set.
rf_y_test_pred_res = pd.Series(rf_clf_res.predict(x_test))
```

In [35]:
```
# Create a confusion matrix.
rf_test_conf_matrix_res = confusion_matrix(y_test, rf_y_test_pred_res)
```

In [36]:
```
# Create a Pandas DataFrame object for the confusion matrix created above and labelled with the classes.
rf_test_cm_df_res = pd.DataFrame(rf_test_conf_matrix_res, columns = labels, index = labels)
```

In [37]:
```
# Create a heatmap for the confusion matrix.
plt.figure(figsize = (10, 5), dpi = 96)
sns.heatmap(rf_test_cm_df_res, annot = True)
plt.show()
```



In [38]:
```
# Print the classification report for the test set.
print(classification_report(y_test, rf_y_test_pred_res))
```

	precision	recall	f1-score	support
1	0.76	0.84	0.80	19
2	0.79	0.65	0.71	23
3	0.33	0.50	0.40	4
5	0.50	0.50	0.50	6
6	0.75	1.00	0.86	3
7	1.00	0.90	0.95	10
accuracy			0.74	65
macro avg	0.69	0.73	0.70	65
weighted avg	0.76	0.74	0.74	65

In [ ]: