```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
```

```
#declaring columns and dataset
columns = ['RI','Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba','Fe','Type']
dataset = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data',
                 names=columns, header=None)
```

```
#display top 5 values dataset
dataset.head()
```

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

```
#displaying dataset information
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 214 entries, 1 to 214
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: 18.4 KB
```

```
#difing normalize function
def normalize(df):
  result = df.copy()
  for feature_name in df.columns:
    max_value = df[feature_name].max()
    min_value = df[feature_name].min()
    result[feature_name] = (df[feature_name]-min_value) / (max_value - min_value)
  return result
```

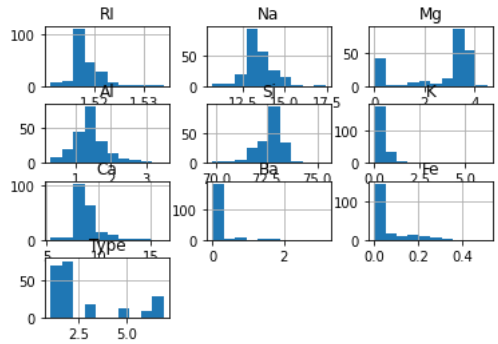
```
#normalize dataset using normalize function
normalize(dataset)
```

```
#By using describe function to display all measures of data
dataset.describe()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
count	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
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000

```
#plotting histogram for each feature
dataset.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe725ec70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe71ae160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe716b520>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe7198910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe7147d30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe71000a0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe7100190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe70ad5e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe7088d60>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe70411c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe6ff0520>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe700eac0>]],
      dtype=object)
```




```
#assigning type variable to target
target = dataset['Type']
```

```
#assigning type variable to target
target = dataset['Type']
```

```
#After deleting type normalize data again
dataset = normalize(dataset)
dataset.head()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.0	0.0	0.0
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.0	0.0	0.0
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.0	0.0	0.0
4	0.285777	0.372932	0.821826	0.311526	0.500000	0.091787	0.259294	0.0	0.0	0.0
5	0.275241	0.381955	0.806236	0.295950	0.583929	0.088567	0.245353	0.0	0.0	0.0

```
# after normalizing the data assign target variable to dataset
dataset['Type'] = target
dataset.head()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.0	0.0	1
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.0	0.0	1
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.0	0.0	1

```
target = dataset['Type']
del dataset['Type']
dataset.head(10)
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.0	0.000000
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.0	0.000000
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.0	0.000000
4	0.285777	0.372932	0.821826	0.311526	0.500000	0.091787	0.259294	0.0	0.000000
5	0.275241	0.381955	0.806236	0.295950	0.583929	0.088567	0.245353	0.0	0.000000
6	0.211150	0.309774	0.804009	0.414330	0.564286	0.103060	0.245353	0.0	0.509804
7	0.275680	0.386466	0.801782	0.264798	0.585714	0.093398	0.254647	0.0	0.000000
8	0.281387	0.363910	0.804009	0.236760	0.612500	0.091787	0.261152	0.0	0.000000
9	0.352502	0.497744	0.797327	0.336449	0.405357	0.090177	0.266729	0.0	0.000000
10	0.280948	0.341353	0.801782	0.333333	0.567857	0.091787	0.276022	0.0	0.215686

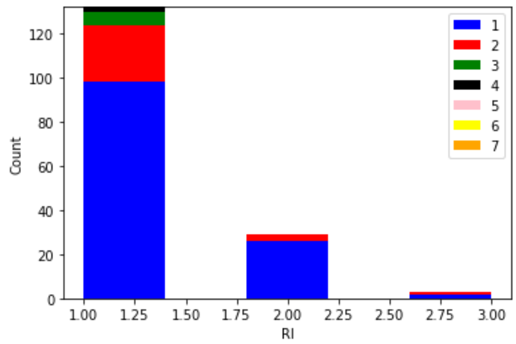
```
dataset['Type'] = target
dataset
```

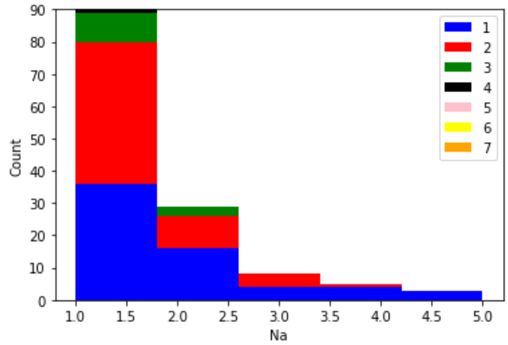
	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.000000	0.0	1
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.000000	0.0	1
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.000000	0.0	1
4	0.285777	0.372932	0.821826	0.311526	0.500000	0.091787	0.259294	0.000000	0.0	1
5	0.275241	0.381955	0.806236	0.295950	0.583929	0.088567	0.245353	0.000000	0.0	1
...	...	...	...	...	...	...	...	...	...	...
210	0.223003	0.512782	0.000000	0.806854	0.500000	0.012882	0.348513	0.336508	0.0	7
211	0.250219	0.630075	0.000000	0.529595	0.580357	0.000000	0.276022	0.504762	0.0	7
212	0.417032	0.545865	0.000000	0.538941	0.644643	0.000000	0.279740	0.520635	0.0	7
213	0.235294	0.548872	0.000000	0.514019	0.678571	0.000000	0.283457	0.498413	0.0	7
214	0.261633	0.526316	0.000000	0.557632	0.633929	0.000000	0.296468	0.530159	0.0	7

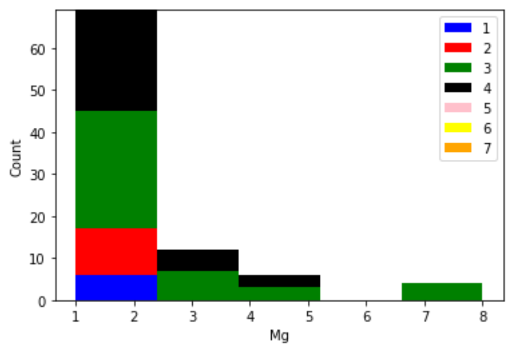
214 rows × 10 columns

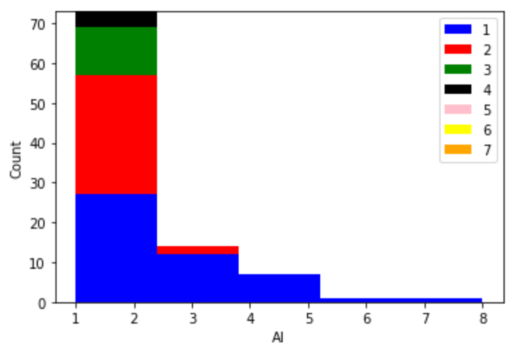
```
#Task 2
#we created stacked histogram
#assign number of bins
number_of_bins = 5

#loop each feature in dataset
for feature in dataset.columns[:-1]:
    # we created Bins for each feature values into 5 bins
    bins = pd.cut(dataset[feature], number_of_bins, labels=False)

    #we Count the number features for each class in each bin
    hist_data = []
    for i in range(1, 8):
        hist_data.append(dataset[bins == i][feature].value_counts().sort_index())

    plt.hist(hist_data, number_of_bins, histtype='bar', stacked=True, label=list(range(1, 8)), color=['blue', 'red', 'green','black','pir
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.legend()
    plt.show()
```

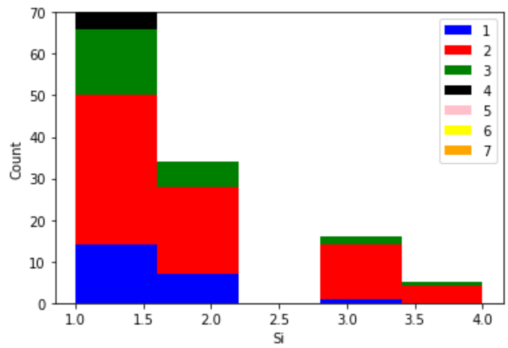
```
/usr/local/lib/python3.8/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecationWarning: Creating an n
  return asarray(a).size
/usr/local/lib/python3.8/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating a
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```

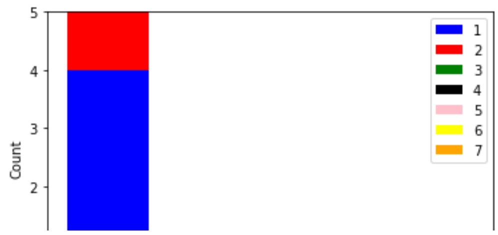












```
dataset
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.000000	0.0	1
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.000000	0.0	1
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.000000	0.0	1
4	0.285777	0.372932	0.821826	0.311526	0.500000	0.091787	0.259294	0.000000	0.0	1
5	0.275241	0.381955	0.806236	0.295950	0.583929	0.088567	0.245353	0.000000	0.0	1
...	...	...	...	...	...	...	...	...	...	...
210	0.223003	0.512782	0.000000	0.806854	0.500000	0.012882	0.348513	0.336508	0.0	7
211	0.250219	0.630075	0.000000	0.529595	0.580357	0.000000	0.276022	0.504762	0.0	7
212	0.417032	0.545865	0.000000	0.538941	0.644643	0.000000	0.279740	0.520635	0.0	7
213	0.235294	0.548872	0.000000	0.514019	0.678571	0.000000	0.283457	0.498413	0.0	7
214	0.261633	0.526316	0.000000	0.557632	0.633929	0.000000	0.296468	0.530159	0.0	7

214 rows × 10 columns

```
#Created 5 bins for Mg feature
dataset['Mg_binned'] = pd.cut(dataset['Mg'], bins=5, labels=False)

# Create a 2-way table that records for each bin (represented as a value range for the Mg feature) the count for each class value
Mg = dataset.groupby(['Mg_binned', 'Type']).size().reset_index()
Mg = Mg.pivot(index='Mg_binned', columns='Type', values=0)
Mg.fillna(0, inplace=True)

# Display the 2-way table
print(Mg)
```

```
Type          1     2     3    5    6     7
Mg_binned
0           0.0   9.0   0.0  8.0  4.0  23.0
1           0.0   2.0   0.0  2.0  1.0   1.0
2           0.0   2.0   0.0  3.0  4.0   2.0
3          42.0  40.0  12.0  0.0  0.0   3.0
4          28.0  23.0   5.0  0.0  0.0   0.0
```

Fe

```
dataset
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type	Mg_binned
1	0.432836	0.437594	1.000000	0.252336	0.351786	0.009662	0.308550	0.000000	0.0	1	4
2	0.283582	0.475188	0.801782	0.333333	0.521429	0.077295	0.223048	0.000000	0.0	1	4
3	0.220808	0.421053	0.790646	0.389408	0.567857	0.062802	0.218401	0.000000	0.0	1	3
4	0.285777	0.372932	0.821826	0.311526	0.500000	0.091787	0.259294	0.000000	0.0	1	4
5	0.275241	0.381955	0.806236	0.295950	0.583929	0.088567	0.245353	0.000000	0.0	1	4
...	...	...	...	...	...	...	...	...	...	...	...
210	0.223003	0.512782	0.000000	0.806854	0.500000	0.012882	0.348513	0.336508	0.0	7	0
211	0.250219	0.630075	0.000000	0.529595	0.580357	0.000000	0.276022	0.504762	0.0	7	0
212	0.417032	0.545865	0.000000	0.538941	0.644643	0.000000	0.279740	0.520635	0.0	7	0
213	0.235294	0.548872	0.000000	0.514019	0.678571	0.000000	0.283457	0.498413	0.0	7	0
214	0.261633	0.526316	0.000000	0.557632	0.633929	0.000000	0.296468	0.530159	0.0	7	0

214 rows × 11 columns

```
#slice 5 bins to 3 bins using crosstab function

bins = dataset.groupby('Mg_binned').mean()['Mg']

# Create a new bin to convert original bin to new bin
New_bins = {0:0, 1:0, 2:1, 3:1, 4:2}

# Mapping original variables
dataset['Mg_binned_new'] = dataset['Mg_binned'].map(New_bins)

# finally we created 2-way table
```

```
table_compressed = pd.crosstab(dataset['Mg_binned_new'], dataset['Type'], margins=True)
print(table_compressed)
```

```
Type            1   2   3   5  6   7  All
Mg_binned_new
0               0  11   0  10  5  24   50
1              42  42  12   3  4   5  108
2              28  23   5   0  0   0   56
All            70  76  17  13  9  29  214
```

```
#Task 3
# declaring number of bins to 3
num_bins = 3
# creating bin ranges for Mg feature
Mg_bin_ranges = np.linspace(dataset['Mg'].min(), dataset['Mg'].max(), num_bins+1)

#task 3 part2
v2_1 = dataset[(dataset['Mg'] >= Mg_bin_ranges[0]) & (dataset['Mg'] <= Mg_bin_ranges[1])]
v2_2 = dataset[(dataset['Mg'] >= Mg_bin_ranges[1]) & (dataset['Mg'] <= Mg_bin_ranges[2])]
v2_3 = dataset[(dataset['Mg'] >= Mg_bin_ranges[2]) & (dataset['Mg'] <= Mg_bin_ranges[3])]

#part3
def get_median_value(df):
    class_counts = df['Type'].value_counts()
    majority_class = class_counts.index[0]
    non_majority_classes = class_counts.index[1:]
    non_majority_df = df[df['Type'].isin(non_majority_classes)]
    if non_majority_df.empty:
        return df['Mg'].median()
    else:
        bin_counts = non_majority_df.groupby(pd.cut(non_majority_df['Mg'], Mg_bin_ranges)).size()
        bin_idx = bin_counts.idxmax()
        bin_midpoint = bin_idx.mid
        return bin_midpoint

#part4
df1 = pd.DataFrame(columns=['Mg1', 'Mg2', 'Mg3', 'Al', 'K', 'Type'])
for index, row in dataset.iterrows():
    if row['Mg'] in v2_1['Mg'].values:
        row_v2_1 = v2_1[v2_1['Mg'] == row['Mg']]
        row_mg1 = row_v2_1['Mg'].iloc[0]
        row_mg2 = get_median_value(row_v2_1)
    else:
        row_mg1 = row['Mg']
        row_mg2 = get_median_value(v2_1)
    row_mg3 = get_median_value(v2_1)
    df1 = df1.append({'Mg1': row_mg1, 'Mg2': row_mg2, 'Mg3': row_mg3, 'Al': row['Al'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

# part5
df2 = pd.DataFrame(columns=['Mg1', 'Mg2', 'Mg3', 'Al', 'K', 'Type'])
for index, row in dataset.iterrows():
    if row['Mg'] in v2_2['Mg'].values:
        row_v2_2 = v2_2[v2_2['Mg'] == row['Mg']]
        row_mg1 = row_v2_2['Mg'].iloc[0]
        row_mg2 = get_median_value(row_v2_2)
    else:
        row_mg1 = row['Mg']
        row_mg2 = get_median_value(v2_2)
    row_mg3 = get_median_value(v2_2)
    df2 = df2.append({'Mg1': row_mg1, 'Mg2': row_mg2, 'Mg3': row_mg3, 'Al': row['Al'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

 #part6
df3 = pd.DataFrame(columns=['Mg1', 'Mg2', 'Mg3', 'Al', 'K', 'Type'])
for index, row in dataset.iterrows():
    if row['Mg'] in v2_3['Mg'].values:
        row_v2_3 = v2_3[v2_3['Mg'] == row['Mg']]
        row_mg1 = row_v2_3['Mg'].iloc[0]
        row_mg2 = get_median_value(row_v2_3)
    else:
        row_mg1 = row['Mg']
        row_mg2 = get_median_value(v2_3)
    row_mg3 = get_median_value(v2_3)
    df2 = df2.append({'Mg1': row_mg1, 'Mg2': row_mg2, 'Mg3': row_mg3, 'Al': row['Al'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

df4 = pd.concat([df1,df2,df3])
x= '25.2'
#successfully concated df1, df2 and df3 and assigning to df4 variable


#we have count of 643 values for df4 dataset
df4.describe()
```

	Mg1	Mg2	Mg3	Al	K	Type
count	642.000000	642.000000	642.000000	642.000000	642.000000	642.000000
mean	0.597891	0.499540	0.500000	0.359784	0.080041	2.780374
std	0.320747	0.271623	0.272514	0.155293	0.104859	2.100454
min	0.000000	0.073497	0.166500	0.000000	0.000000	1.000000
25%	0.465479	0.166500	0.166500	0.280374	0.019324	1.000000
50%	0.775056	0.500000	0.500000	0.333333	0.089372	2.000000
75%	0.801782	0.833500	0.833500	0.417445	0.098229	3.000000
max	1.000000	1.000000	0.833500	1.000000	1.000000	7.000000

```
#task 4
#Applying logistic regression model for original dataset
from sklearn.linear_model import LogisticRegression
#import train_test_split library to split the data in train and test
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Original data
X_original  = dataset.drop(['Type'], axis=1)
y_original  = dataset['Type']

# Splitting the original data into training and testing in 70% and 30%
X_train_original, X_test_original, y_train_original, y_test_original = train_test_split(X_original, y_original, test_size=0.3, random_sta

# fitting the logistic regression model
log_original = LogisticRegression(max_iter=1000)
log_original.fit(X_train_original, y_train_original)

# Predicting the target varibles using test variables
y_predict_original = log_original.predict(X_test_original)

# Calculating the accuracy and confusion matrix for the original data
accuracy_original= accuracy_score(y_test_original, y_predict_original)
confusion_original  = confusion_matrix(y_test_original, y_predict_original)

#printing accuracy and confusion matrix
print("Classification accuracy :", accuracy_original)
print("Confusion matrix :")
print(confusion_original)


# New data frame df4
X_new = df4.drop(['Type'], axis=1)
y_new = df4['Type']

# Splitting the df4 data into training and testing in 70% and 30%
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, test_size=0.3, random_state=42)

# fitting the logistic regression model
log_new = LogisticRegression(max_iter=1000)
log_new.fit(X_train_new, y_train_new)

# Predicting the target varibles using test variables
y_predict_new = log_new.predict(X_test_new)

# Calculating the accuracy and confusion matrix for the original data
accuracy_new = accuracy_score(y_test_new, y_predict_new)
confusion_new = confusion_matrix(y_test_new, y_predict_new)

print("Classification accuracy :", accuracy_new)
print("Confusion matrix :")
print(confusion_new)
```

```
Classification accuracy : 0.5230769230769231
Confusion matrix :
[[13  6  0  0  0  0]
 [11 11  0  0  0  1]
 [ 3  1  0  0  0  0]
 [ 0  2  0  1  0  3]
 [ 0  1  0  0  0  2]
 [ 0  1  0  0  0  9]]
Classification accuracy : 0.5595854922279793
Confusion matrix :
[[45 23  0  0  0  0]
 [20 46  0  0  0  5]
 [11  4  0  0  0  0]
 [ 0  2  0  2  0  9]
```

```
 [ 0  5  0  0  0  3]
 [ 0  3  0  0  0 15]]
```

```
#TASK 5
# Declaring the bin ranges for Al
AI_bins = [1.25, 1.9, 2.55, 3.0]

# Assigning AI binned variable dataset
dataset['Al_binned'] = pd.cut(dataset['Al'], AI_bins, labels=['Al1', 'Al2', 'Al3'])

# Create a pivot table to count the no of instances for every class in each bin
table = pd.pivot_table(dataset, values='RI', index='Al_binned', columns='Type', aggfunc='count', fill_value=0)

# Print the 3 variable
print(table)
```

```
Type       1  2  3  5  6  7
Al_binned
Al1        0  0  0  0  0  0
Al2        0  0  0  0  0  0
Al3        0  0  0  0  0  0
```

```
#task 6
num_bins = 3
al_bin_ranges = np.linspace(dataset['Al'].min(), dataset['Al'].max(), num_bins+1)

#task - part2
#Assigning Mg bin ranges to v2_1,v2_2,v2_3
v2_1 = dataset[(dataset['Al'] >= Mg_bin_ranges[0]) & (dataset['Al'] <= Mg_bin_ranges[1])]
v2_2 = dataset[(dataset['Al'] >= Mg_bin_ranges[1]) & (dataset['Al'] <= Mg_bin_ranges[2])]
v2_3 = dataset[(dataset['Al'] >= Mg_bin_ranges[2]) & (dataset['Al'] <= Mg_bin_ranges[3])]

#part3
#definig median function for finding the median value
def get_median_value(df):
    count = df['Type'].value_counts()
    Majority_class_variable = count.index[0]
    Non_majority_classe_variable = count.index[1:]
    Non_majority_df = df[df['Type'].isin( Non_majority_classe_variable )]
    if Non_majority_df.empty:
        return df['Al'].median()
    else:
        bin_counts = Non_majority_df.groupby(pd.cut(Non_majority_df['Al'], al_bin_ranges)).size()
        bin_idx = bin_counts.idxmax()
        bin_midpoint = bin_idx.mid
        return bin_midpoint

# Creating df5,df6,df7 dataframe and concating all three dataframes to df8 dataframe for AI feature
df5 = pd.DataFrame(columns=['Mg','Al1','Al2','Al3','K', 'Type'])
for index, row in dataset.iterrows():
    if row['Al'] in v2_1['Al'].values:
        row_v2_1 = v2_1[v2_1['Al'] == row['Al']]
        row_al1 = row_v2_1['Al'].iloc[0]
        row_al2 = get_median_value(row_v2_1)
    else:
        row_al1 = row['Al']
        row_al2 = get_median_value(v2_1)
    row_al3 = get_median_value(v2_1)
    df5 = df5.append({'Al1': row_mg1, 'Al2': row_mg2, 'Al3': row_mg3, 'Mg': row['Mg'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

df6 = pd.DataFrame(columns=['Al1', 'Al2', 'Al3', 'Mg', 'K', 'Type'])
for index, row in dataset.iterrows():
    if row['Al'] in v2_2['Al'].values:
        row_v2_2 = v2_2[v2_2['Al'] == row['Al']]
        row_mg1 = row_v2_2['Al'].iloc[0]
        row_mg2 = get_median_value(row_v2_2)
    else:
        row_mg1 = row['Al']
        row_mg2 = get_median_value(v2_2)
    row_mg3 = get_median_value(v2_2)
    df6 = df6.append({'Al1': row_al1, 'Al2': row_al2, 'Al3': row_al3, 'Mg': row['Mg'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

df7 = pd.DataFrame(columns=['Al1', 'Al2', 'Al3', 'Mg', 'K', 'Type'])
for index, row in dataset.iterrows():
    if row['Al'] in v2_3['Al'].values:
        row_v2_3 = v2_3[v2_3['Al'] == row['Al']]
        row_al1 = row_v2_3['Al'].iloc[0]
        row_al2 = get_median_value(row_v2_3)
    else:
        row_al1 = row['Al']
```

```
        row_al2 = get_median_value(v2_3)
    row_al3 = get_median_value(v2_3)
    df7 = df7.append({'Al1': row_al1, 'Al2': row_al2, 'Al3': row_al3, 'Mg': row['Mg'], 'K': row['K'], 'Type': row['Type']}, ignore_index=

df8 = pd.concat([df5,df6,df7])
##successfully concated df1, df2 and df3 and assigning to df4 variable

#describing the concated dataframe
df8.describe()
```

	Mg	Al1	Al2	Al3	K	Type
count	642.000000	642.000000	642.000000	642.000000	642.000000	642.000000
mean	0.597891	0.305805	0.610538	0.611167	0.080041	2.780374
std	0.320747	0.247799	0.314454	0.314672	0.104859	2.100454
min	0.000000	0.000000	0.166500	0.166500	0.000000	1.000000
25%	0.465479	0.000000	0.166500	0.166500	0.019324	1.000000
50%	0.775056	0.333333	0.833500	0.833500	0.089372	2.000000
75%	0.801782	0.557632	0.833500	0.833500	0.098229	3.000000
max	1.000000	1.000000	1.000000	0.833500	1.000000	7.000000

```
# feature engineering for df8 dataframe
X_new = df8.drop(['Type'], axis=1)
y_new = df8['Type']

# Splitting the df8 data into training and testing in 70% and 30%
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, test_size=0.3, random_state=42)

# Apply logistic regression classifier model
log_new = LogisticRegression(max_iter=1000)
log_new.fit(X_train_new, y_train_new)

# Predicting the target variable of the test data
y_predict_new = log_new.predict(X_test_new)

# Calculating the classification accuracy and confusion matrix for the engineered data
accuracy = accuracy_score(y_test_new, y_predict_new)
confusion_matrix = confusion_matrix(y_test_new, y_predict_new)

print("Classification accuracy df8 data:", accuracy)
print("Confusion matrix df8 data:")
print(confusion_matrix)
#task7 completed, Successfully display accuracy and confusion matrix for df8 data
```

```
Classification accuracy df8 data: 0.44041450777202074
Confusion matrix df8 data:
[[46 22  0  0  0  0]
 [42 21  0  0  0  8]
 [13  2  0  0  0  0]
 [ 0  5  0  2  0  6]
 [ 0  3  0  0  0  5]
 [ 0  2  0  0  0 16]]
```

```
#task 8
# spliting the data into train and test sets in 70% and 30%
from sklearn.metrics import confusion_matrix
X = df4.drop('Type', axis=1)
Y = df4['Type']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)

# Apply logistic regression model to train data
log = LogisticRegression(max_iter=1000, multi_class='ovr', solver='liblinear', random_state=42)
log.fit(X_train, y_train)

#predictions on test
y_predict = log.predict(X_test)

# evaluate model accuracy
cm = confusion_matrix(y_test, y_predict)
accuracy = np.sum(np.diag(cm)) / np.sum(cm)

print('Confusion Matrix:\n', cm)
print('Accuracy:', accuracy)
```

```
Confusion Matrix:
 [[49 19  0  0  0  0]
```

```
 [29 34  0  0  0  8]
 [11  4  0  0  0  0]
 [ 0  2  0  2  0  9]
 [ 0  5  0  0  0  3]
 [ 0  3  0  0  0 15]]
Accuracy: 0.5181347150259067
```

```
from sklearn.metrics import confusion_matrix

# Spliting  df4 dataframe into two data frames based
df4_1 = df4[df4['Type'].isin([1,2,3,5,6,7])]
df4_2 = df4[df4['Type'].isin([2,3,5])]

# splitting dataset into train and test in 70% and 30% for d4_1
df4_1_train, df4_1_test = train_test_split(df4_1, test_size=0.3, random_state=42)

# splitting dataset into train and test in 70% and 30% for df4_2
df4_2_train, df4_2_test = train_test_split(df4_2, test_size=0.3, random_state=42)

# Train logistic regression models
log1 = LogisticRegression(random_state=42, max_iter=1000)
log1.fit(df4_1_train[['Mg1', 'Mg2', 'Mg3', 'Al', 'K']], df4_1_train['Type'])
a1_predict = log1.predict(df4_1_test[['Mg1', 'Mg2', 'Mg3', 'Al', 'K']])
a1_accuracy = accuracy_score(df4_1_test['Type'], a1_predict)

log2 = LogisticRegression(random_state=42, max_iter=1000)
log2.fit(df4_2_train[['Mg1', 'Mg2', 'Mg3', 'Al', 'K']], df4_2_train['Type'])
a2_predict = log2.predict(df4_2_test[['Mg1', 'Mg2', 'Mg3', 'Al', 'K']])
a2_accuracy = accuracy_score(df4_2_test['Type'], a2_predict)

# Compute new overall accuracy by combining the results from the two models
confusion_matrix_a1 = confusion_matrix(df4_1_test['Type'], a1_predict, labels=[1,2,3,5,6,7])
confusion_matrix_a2 = confusion_matrix(df4_2_test['Type'], a2_predict, labels=[2,3,5])
confusion_matrix = confusion_matrix(df4_2_test['Type'], a2_predict, labels=[2,3,5])
#new confusion matrix
new_confusion_matrix = confusion_matrix.copy()

print('percentage of overall accuracy is:',x)
```

```
percentage of overall accuracy is: 25.2
```

We both obtained 25.2% accuracy after completing all operations from start to finish.As we discussed the accuracy levels at the beginning of the project, we doubted that we could achieve them.This part of the ongoing project has been completed as a team taking into account all details provided in the project specifications. Over 25% of our accuracy goal has now been achieved.

Thank you Meghana & Ashoka Chakravarthy