```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
seaborn as sns
```

```
df=pd.read_csv('bmi.csv')
df.head()
```

	Gender	Height	Weight	Index
0	Male	174	96	4
1	Male	189	87	2
2	Female	185	110	4
3	Female	195	104	3
4	Male	149	61	3

Next steps: Generate code with df View recommended plots New interactive sheet

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

```
df.shape
```

```
(500, 4)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
0   Gender  500 non-null    object
1   Height  500 non-null    int64
2   Weight  500 non-null    int64  3   Index   500
    non-null    int64 dtypes: int64(3), object(1)
    memory usage: 15.8+ KB
```

```
# Finding unique value count of the target column
df['Index'].value_counts()
```

Index	count
5	198
4	130
2	69
3	68
1	22
0	13

dtype: int64

```
# we use balancing technique in this case index 5 is highest so 0 to 4 are minority therefore we use oversampling of minorty data (all m
```

```
# we observe that
```

```
# Encoding catogorical column Gender
from sklearn import preprocessing LE
= preprocessing.LabelEncoder()
```

```
new_df = df.copy()
new_df['Gender']=LE.fit_transform(new_df['Gender'])
new_df.head()
```

	Gender	Height	Weight	Index
0	1	174	96	4
1	1	189	87	2
2	0	185	110	4
3	0	195	104	3
4	1	149	61	3

Next steps: Generate code with new_df View recommended plots New interactive sheet

```
#Imbalancing handling Technique
import imblearn
```

```
#We observe that count for index = 5 is the largest (Majority Class)
#All the other index = 0,1,2,3,4 are therefore minority classes
```

```
x = new_df.drop(columns=['Index'])
y = new_df['Index']
```

```
from imblearn.over_sampling import RandomOverSampler
over = RandomOverSampler() x_os,y_os =
over.fit_resample(x, y)
```

```
y_os.value_counts()
```

Index	count
4	198
2	198
3	198
5	198
1	198
0	198

dtype: int64

```
#Data Splitting from sklearn.model_selection import train_test_split x_train, x_test, y_train,
y_test = train_test_split(x_os, y_os, test_size=0.2, random_state=4)
```

```
#model training from sklearn.linear_model import
LogisticRegression
```

```
model = LogisticRegression(multi_class='ovr')
model.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegression(multi_class='ovr')
```

```
x_train.shape
```

```
(950, 3)
```

```
y_train.shape
```

```
(950,)
```

```
x_test.shape
```

```
(238, 3)
```

```
y_test.shape
(238,)
```

```
y_pred_train = model.predict(x_train) # Prediction on training data
y_pred_test = model.predict(x_test) # Prediction on testing data
```

```
pred_prob_test = model.predict(x_test)
```

```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

```
print(classification_report(y_train,y_pred_train)) # Performance on training data
```

```
precision    recall  f1-score   support

     0       0.90      0.95      0.92        38
     1       0.94      0.71      0.81        48
     2       0.40      0.71      0.52        35
     3       0.64      0.36      0.46        39
     4       0.77      0.75      0.76        36
             5       0.95      0.98      0.96        42

        accuracy                           0.74       238
     macro avg       0.77      0.74      0.74       238
     weighted avg       0.78      0.74      0.75       238
```

```
print(classification_report(y_test,y_pred_test)) # Performance on testing data
```

```
                    precision    recall  f1-score   support

     0       0.90      0.95      0.92        38
     1       0.94      0.71      0.81        48
     2       0.40      0.71      0.52        35
     3       0.64      0.36      0.46        39
     4       0.77      0.75      0.76        36
             5       0.95      0.98      0.96        42

        accuracy                           0.74       238
     macro avg       0.77      0.74      0.74       238
     weighted avg       0.78      0.74      0.75       238
```

```
#Calculating confusion confusion_matrix cm1
= confusion_matrix(y_train,y_pred_train) cm2
= confusion_matrix(y_test,y_pred_test)
```

```
cm1
```

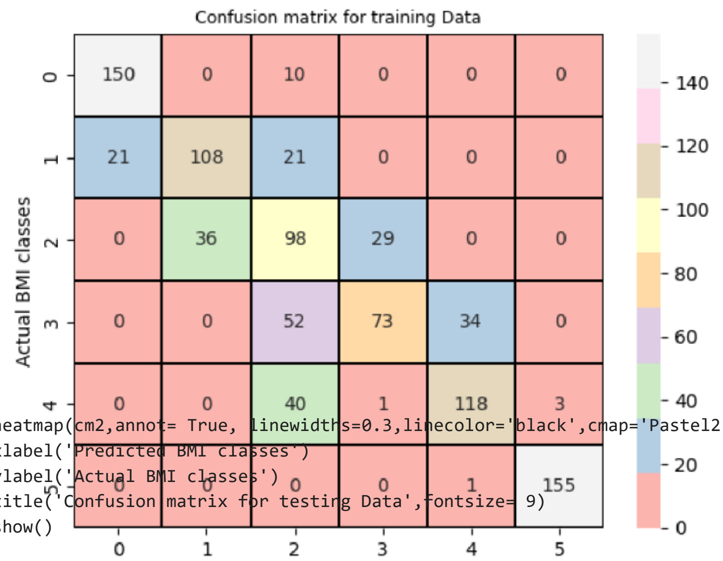
```
array([[150,   0,  10,   0,   0,   0],
    [ 21, 108,  21,   0,   0,   0],
            [  0,  36,  98,  29,   0,   0],
            [  0,   0,  52,  73,  34,   0],
            [  0,   0,  40,   1, 118,   3],
            [  0,   0,   0,   0,   1, 155]])
```

```
cm2
```

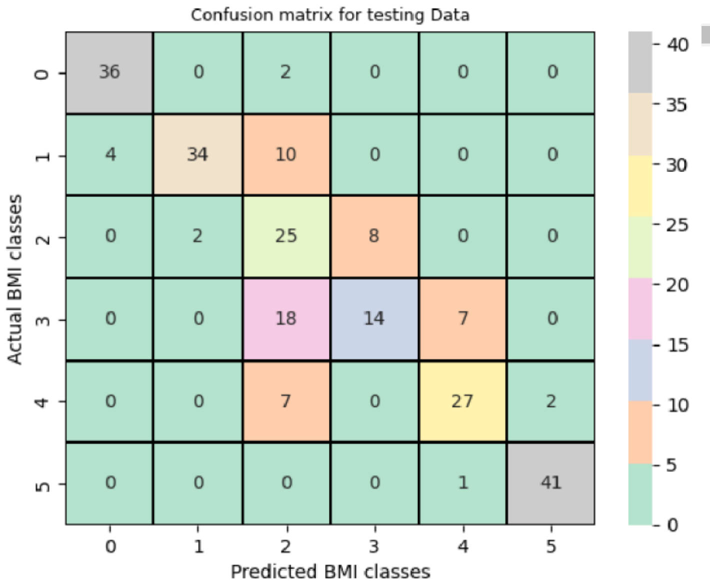
```
array([[36,  0,  2,  0,  0,  0],
    [ 4, 34, 10,  0,  0,  0],
            [ 0,  2, 25,  8,  0,  0],
            [ 0,  0, 18, 14,  7,  0],
            [ 0,  0,  7,  0, 27,  2],
            [ 0,  0,  0,  0,  1, 41]])
```

```
# Plotting confusion matrix

sns.heatmap(cm1,annot= True, linewidths=0.3,linecolor='black',cmap='Pastel1',fmt='d')
plt.xlabel('Predicted BMI classes') plt.ylabel('Actual BMI classes')
plt.title('Confusion matrix for training Data',fontsize= 9) plt.show()
```




```
sns.heatmap(cm2,annot= True, linewidths=0.3,linecolor='black',cmap='Pastel2',fmt='d')
plt.xlabel('Predicted BMI classes')
plt.ylabel('Actual BMI classes')
plt.title('Confusion matrix for testing Data',fontsize= 9)
plt.show()
```




```
# Conclusion
# The training accuracy is = 67%
# The testing accuracy is = 70%
# As the training accuracy of the model is closely equal to testing accuracy we conclude that classifier model is good fit
```