Make a multiclass logistic regression to predict the star type:-

In [217…
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [217…
```python
sns.set(rc={"figure.figsize":(15,8)})
```

Absolute Temperature (in K) Relative Luminosity (L/Lo) Relative Radius (R/Ro) Absolute Magnitude (Mv) Star Color (white,Red,Blue,Yellow,yellow-orange etc) Spectral Class (O,B,A,F,G,K,,M) Star Type **(Red Dwarf, Brown Dwarf, White Dwarf, Main Sequence , SuperGiants, HyperGiants)**

In [217…
```python
df=pd.read_csv('Stars.csv')
```

In [217…
```python
df
```

Out[2173]:

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	Brown Dwarf
1	3042	0.000500	0.1542	16.60	Red	M	Brown Dwarf
2	2600	0.000300	0.1020	18.70	Red	M	Brown Dwarf
3	2800	0.000200	0.1600	16.65	Red	M	Brown Dwarf
4	1939	0.000138	0.1030	20.06	Red	M	Brown Dwarf
...	...	...	...	...	...	...	...
235	38940	374830.000000	1356.0000	-9.93	Blue	O	Hypergiant
236	30839	834042.000000	1194.0000	-10.63	Blue	O	Hypergiant
237	8829	537493.000000	1423.0000	-10.73	White	A	Hypergiant
238	9235	404940.000000	1112.0000	-11.23	White	A	Hypergiant
239	37882	294903.000000	1783.0000	-7.80	Blue	O	Hypergiant

240 rows × 7 columns

In [217…
```python
df.duplicated().sum()
```

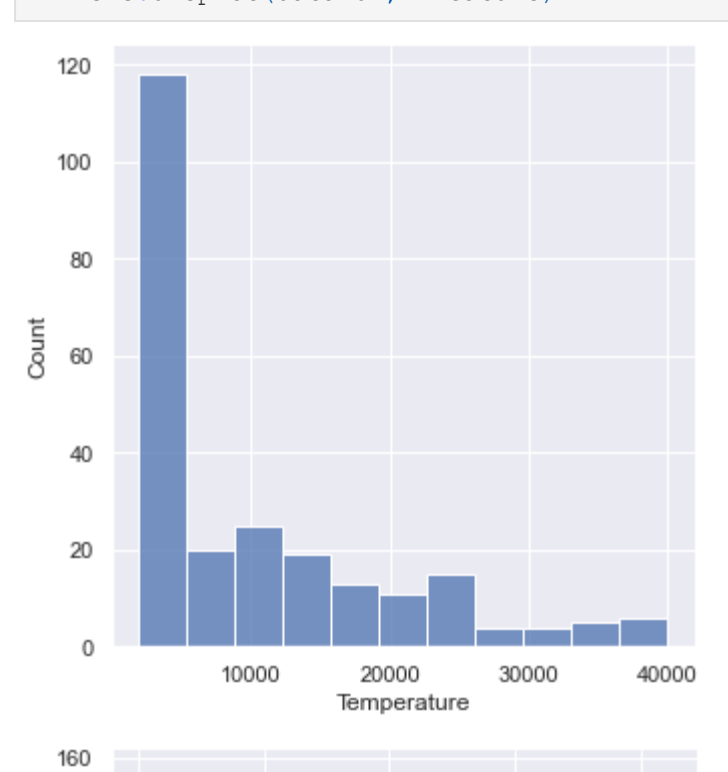
Out[2174]:
```
0
```

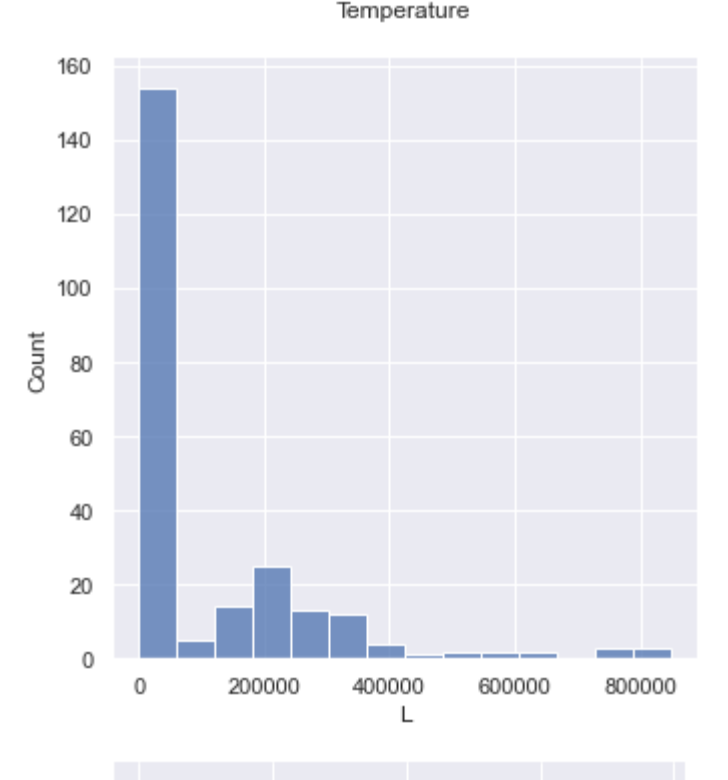
In [217…
```python
df.info()
```

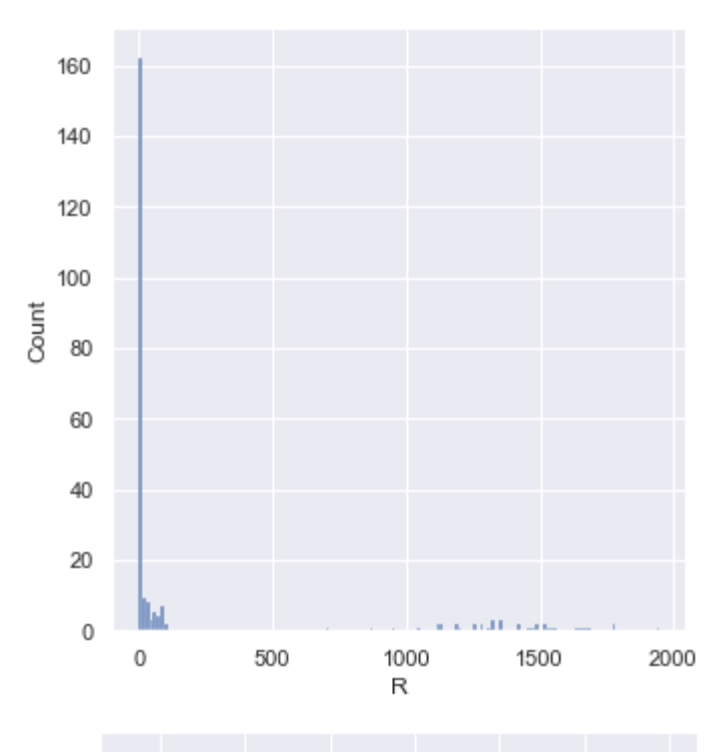
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 240 entries, 0 to 239
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Temperature     240 non-null    int64  
 1   L               240 non-null    float64
 2   R               240 non-null    float64
 3   A_M             240 non-null    float64
 4   Color           240 non-null    object 
 5   Spectral_Class  240 non-null    object 
 6   Type            240 non-null    object 
dtypes: float64(3), int64(1), object(3)
memory usage: 13.2+ KB
```

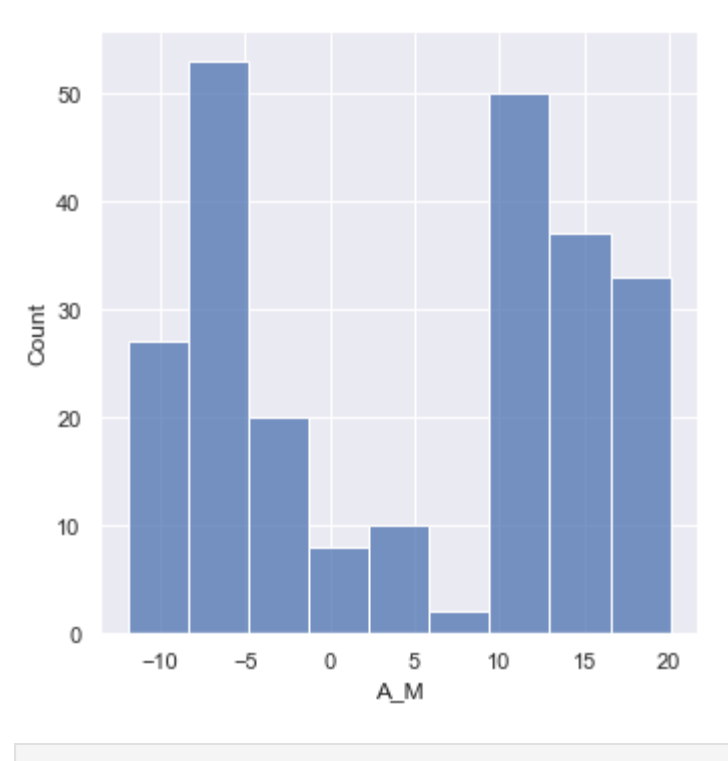
In [217…
```python
numeric_columns=df.select_dtypes('number').columns
```

In [217…
```python
for feature in numeric_columns:
    sns.displot(data=df,x=feature)
```

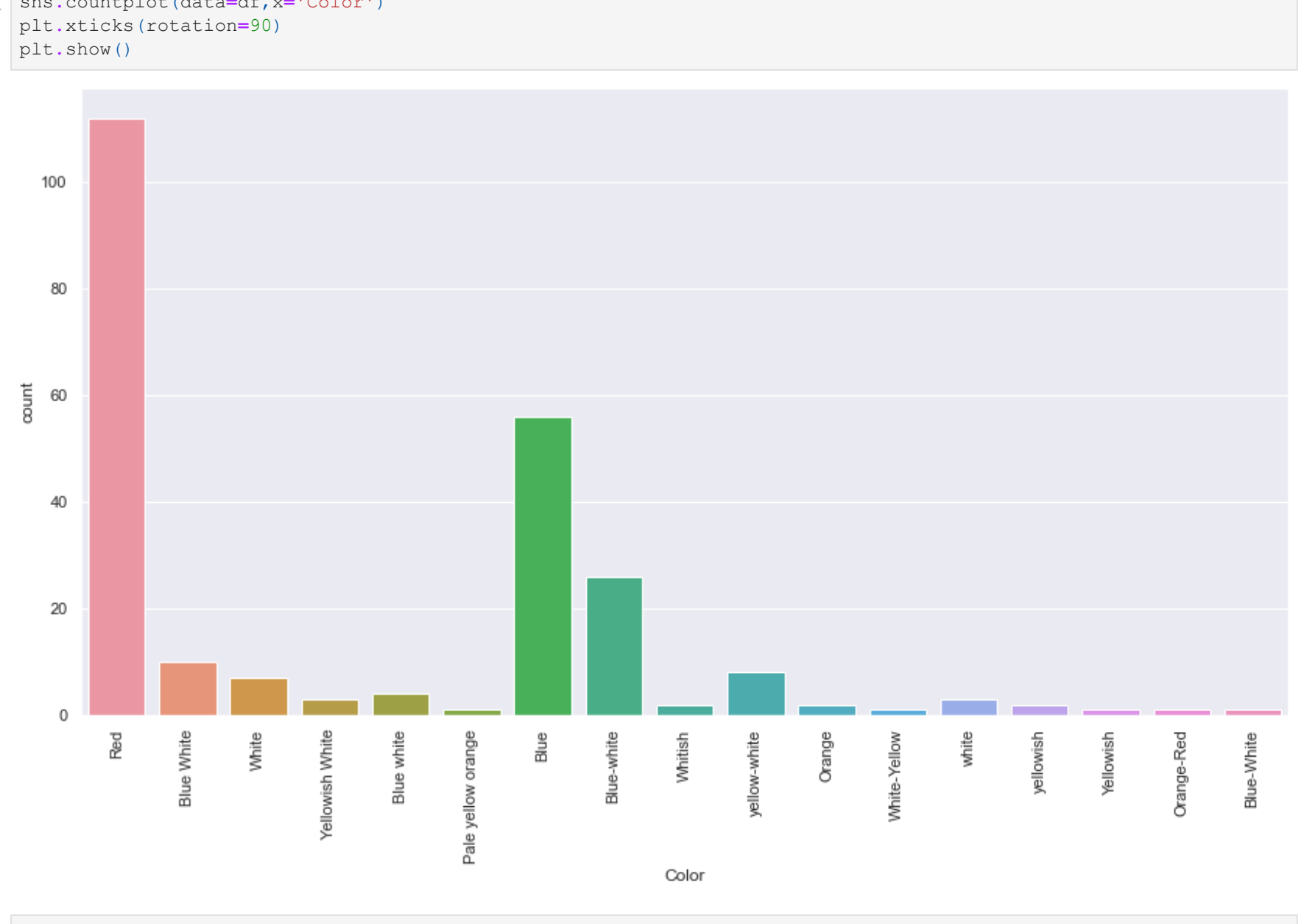






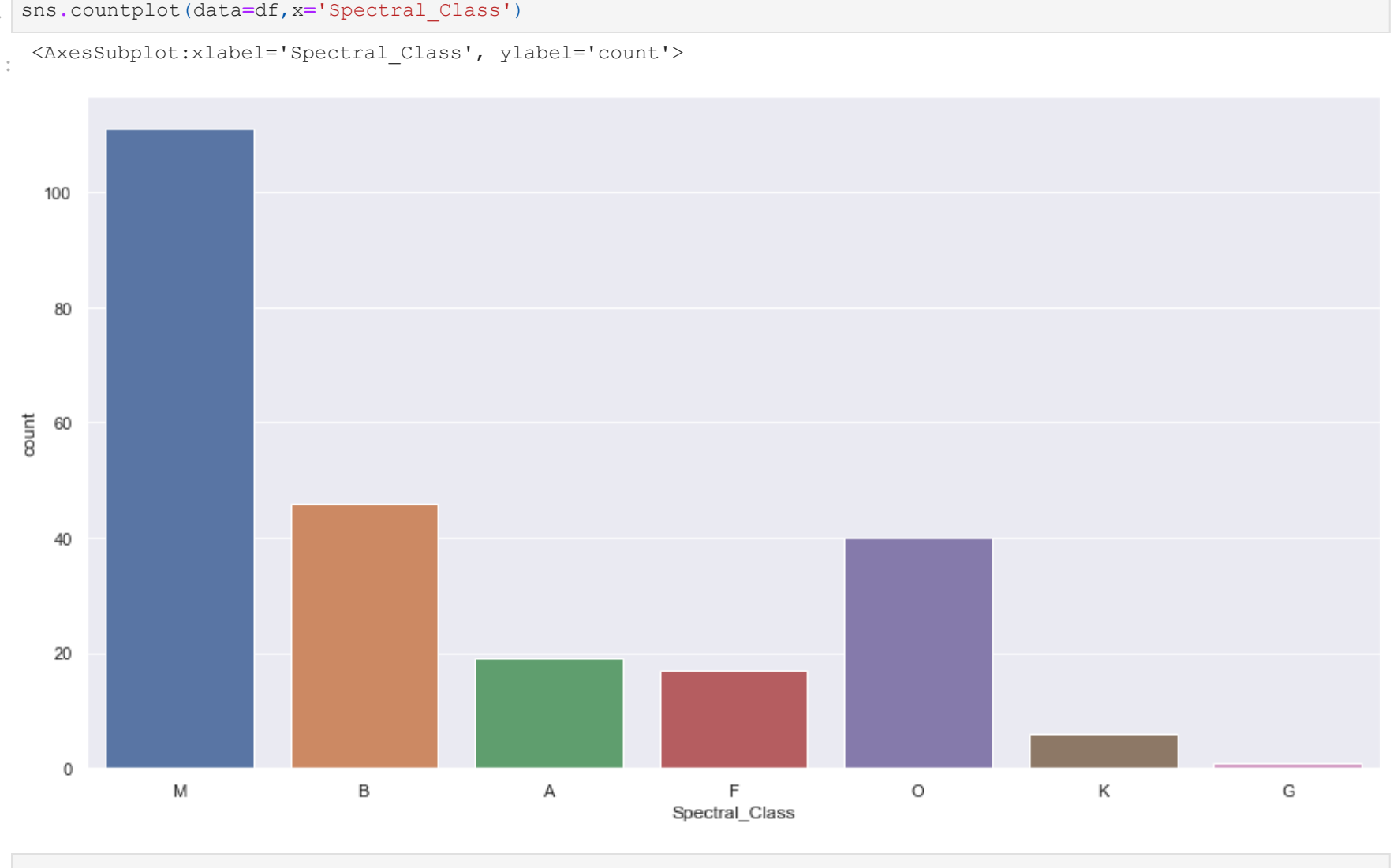


In [217…
```python
sns.countplot(data=df,x='Color')
plt.xticks(rotation=90)
plt.show()
```



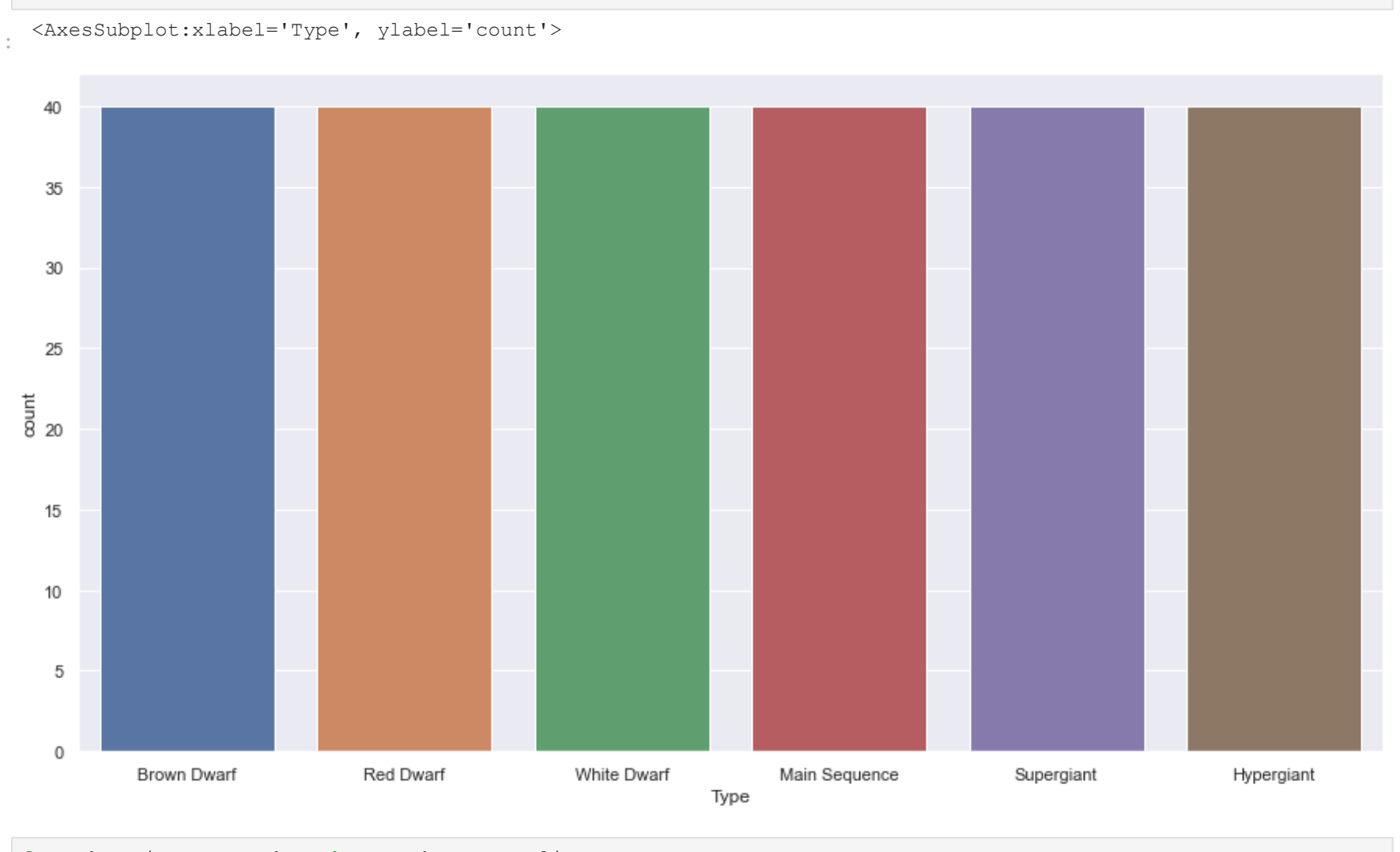
In [217…
```python
sns.countplot(data=df,x='Spectral_Class')
```

Out[2179]:
```
<AxesSubplot:xlabel='Spectral_Class', ylabel='count'>
```



In [218…
```python
sns.countplot(data=df,x='Type')
#it's balanaced
```

Out[2180]:
```
<AxesSubplot:xlabel='Type', ylabel='count'>
```



In [218…
```python
from datasist.structdata import detect_outliers
```

In [218…
```python
df.shape[0]
```

Out[2182]:
```
240
```

In [218…
```python
# only 11 row i will drop them
len(detect_outliers(df,0,['Temperature']))
```

Out[2183]:
```
11
```

In [218…
```python
idx=detect_outliers(df,0,['Temperature'])
```

In [218…
```python
df.shape
```

Out[2185]:
```
(240, 7)
```

In [218…
```python
df
```

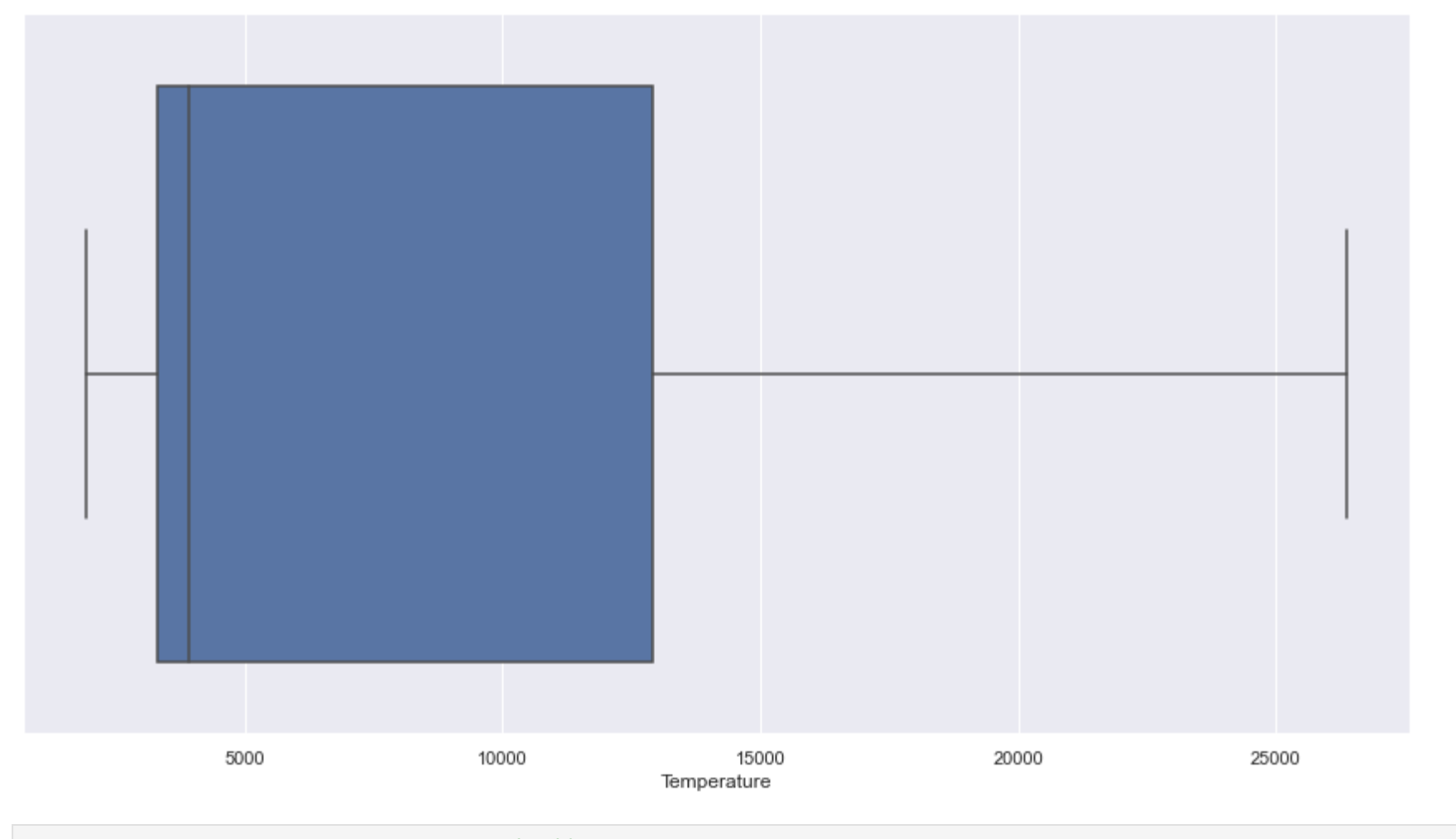
Out[2186]:

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	Brown Dwarf
1	3042	0.000500	0.1542	16.60	Red	M	Brown Dwarf
2	2600	0.000300	0.1020	18.70	Red	M	Brown Dwarf
3	2800	0.000200	0.1600	16.65	Red	M	Brown Dwarf
4	1939	0.000138	0.1030	20.06	Red	M	Brown Dwarf
...	...	...	...	...	...	...	...
235	38940	374830.000000	1356.0000	-9.93	Blue	O	Hypergiant
236	30839	834042.000000	1194.0000	-10.63	Blue	O	Hypergiant
237	8829	537493.000000	1423.0000	-10.73	White	A	Hypergiant
238	9235	404940.000000	1112.0000	-11.23	White	A	Hypergiant
239	37882	294903.000000	1783.0000	-7.80	Blue	O	Hypergiant

240 rows × 7 columns

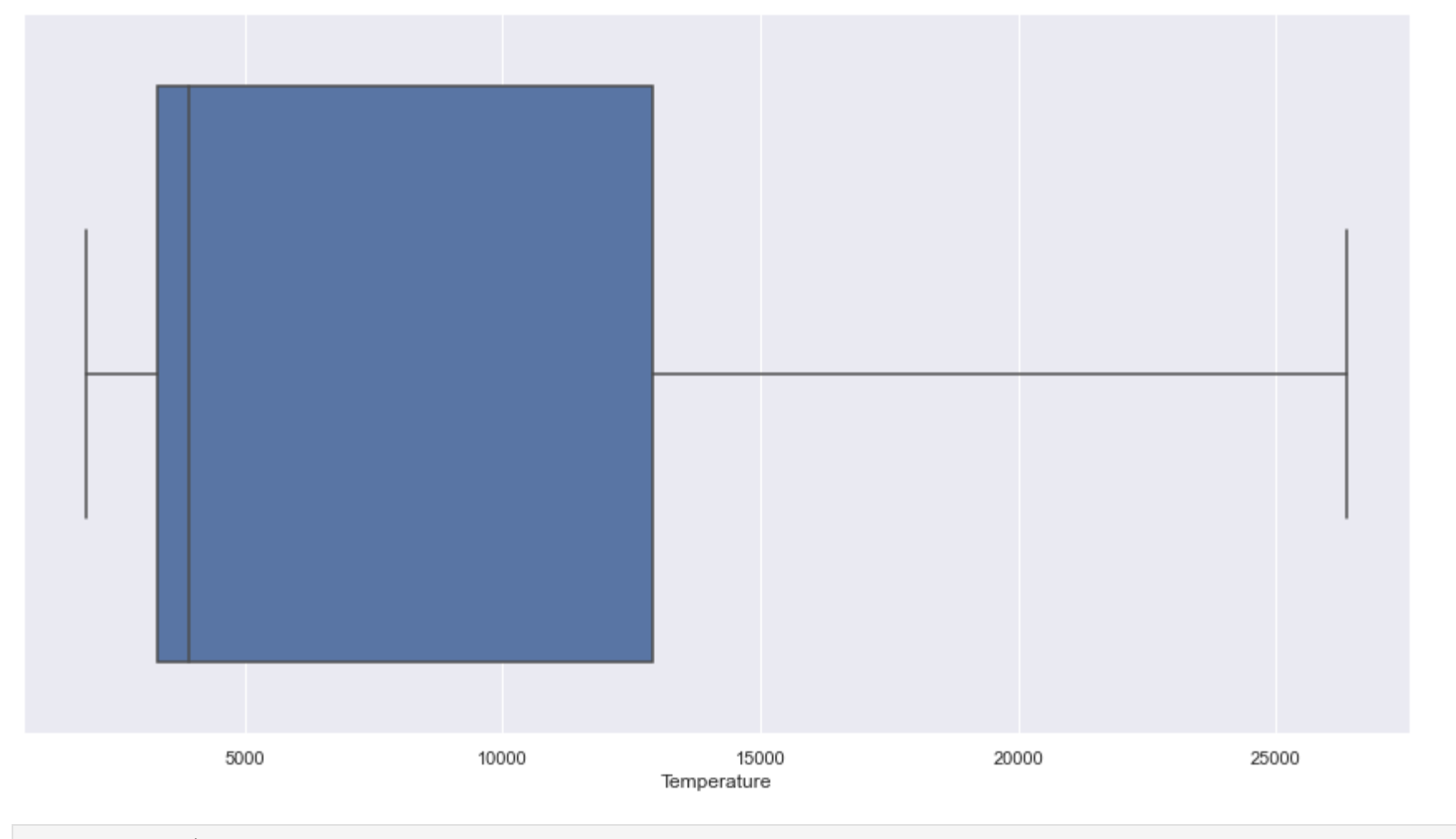
In [218…
```python
sns.boxplot(data=df[df['Temperature']<27700],x='Temperature')
```

Out[2187]:
```
<AxesSubplot:xlabel='Temperature'>
```



In [218…
```python
sns.boxplot(data=df[df['Temperature']<27700],x='Temperature')
```

Out[2188]:
```
<AxesSubplot:xlabel='Temperature'>
```



In [218…
```python
def temp_outliers(x):
    if x>27700:
        return np.nan
    else:
        return x
```

In [219…
```python
df['Temperature']=df['Temperature'].apply(temp_outliers)
```

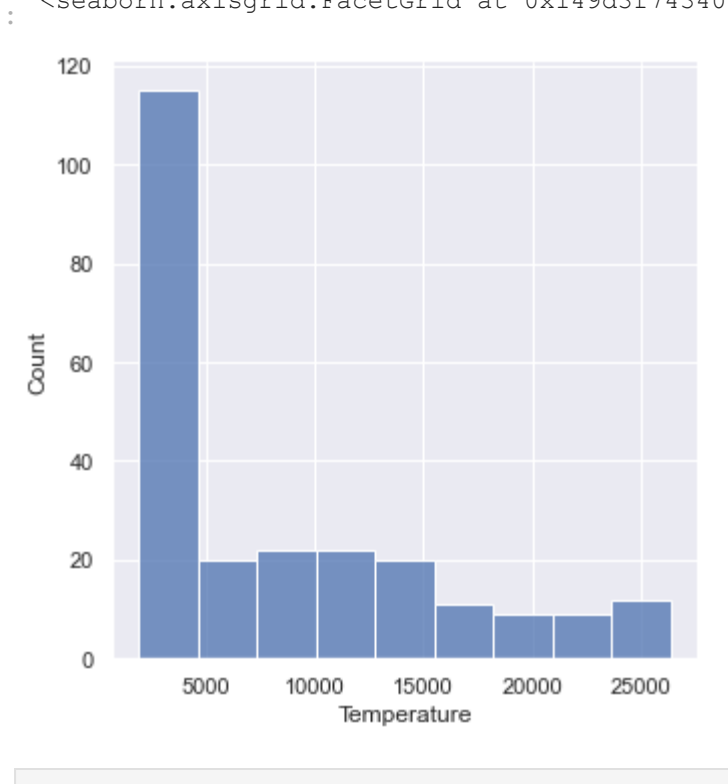
In [219…
```python
df.isnull().sum()
```

Out[2191]:
```
Temperature       18
L                  0
R                  0
A_M                0
Color              0
Spectral_Class     0
Type               0
dtype: int64
```

In [219…
```python
# now i will put these values by their type
df['Temperature']=df.groupby('Type')['Temperature'].apply(lambda x:x.fillna(x.mean()))
```

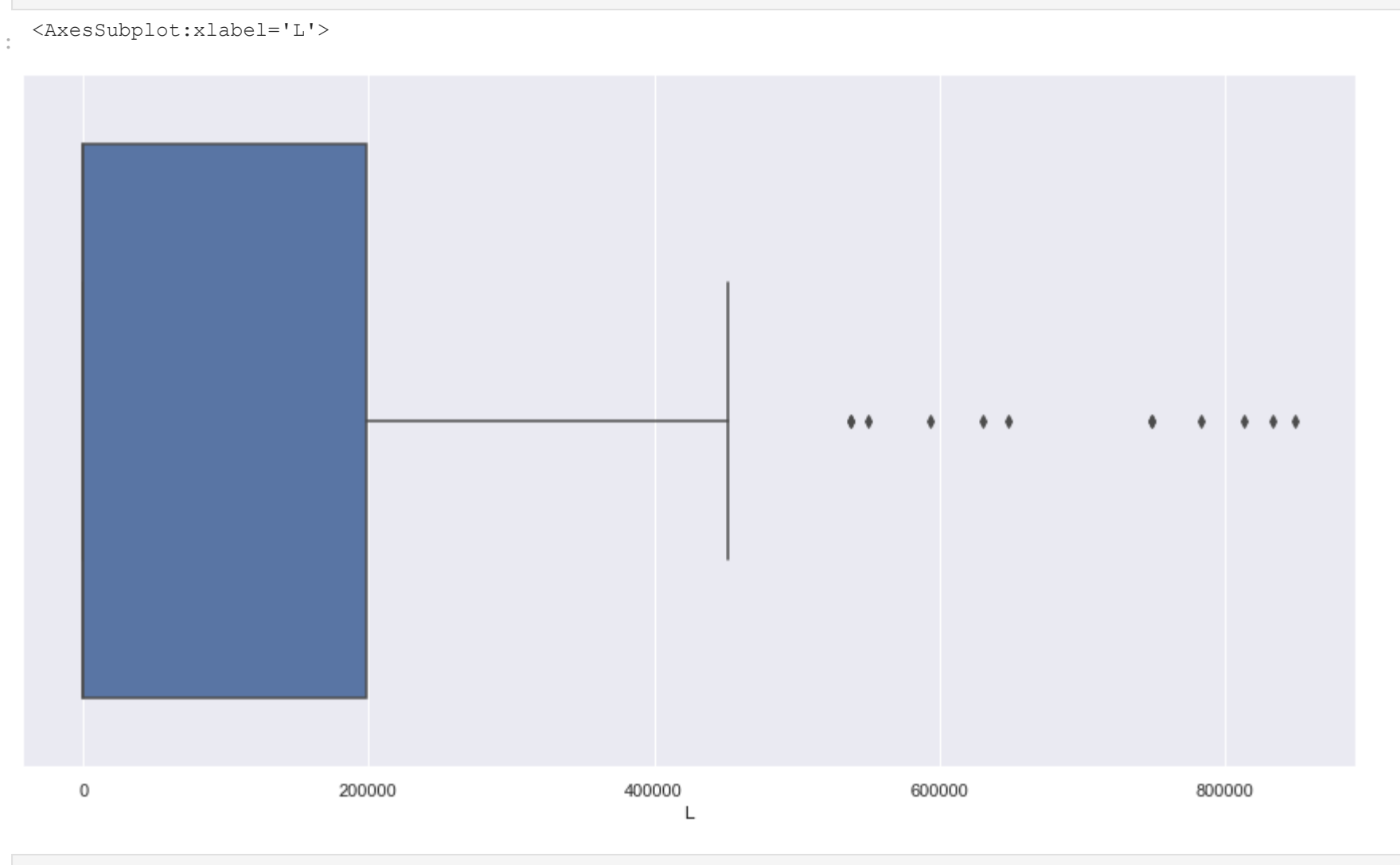
In [219…
```python
sns.displot(data=df,x='Temperature')
```

Out[2193]:
```
<seaborn.axisgrid.FacetGrid at 0x149d3f74340>
```



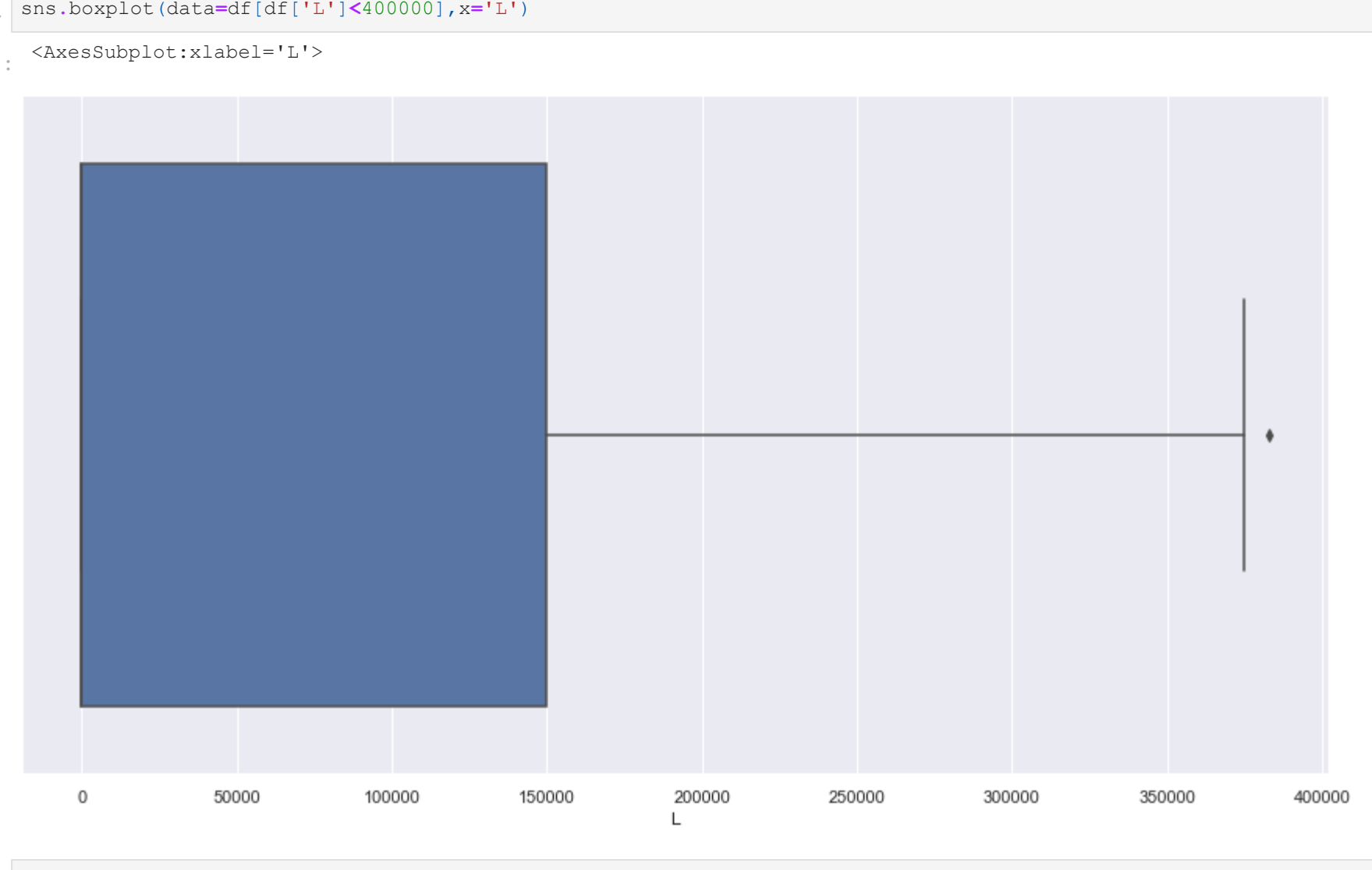
In [219…
```python
sns.boxplot(data=df,x='L')
```

Out[2194]:
```
<AxesSubplot:xlabel='L'>
```



In [219…
```python
sns.boxplot(data=df[df['L']<400000],x='L')
```

Out[2195]:
```
<AxesSubplot:xlabel='L'>
```



In [219…
```python
def L_outliers(x):
    if x>400000:
        return np.nan
    else:
        return x
```

In [219…
```python
df['L']=df['L'].apply(L_outliers)
```

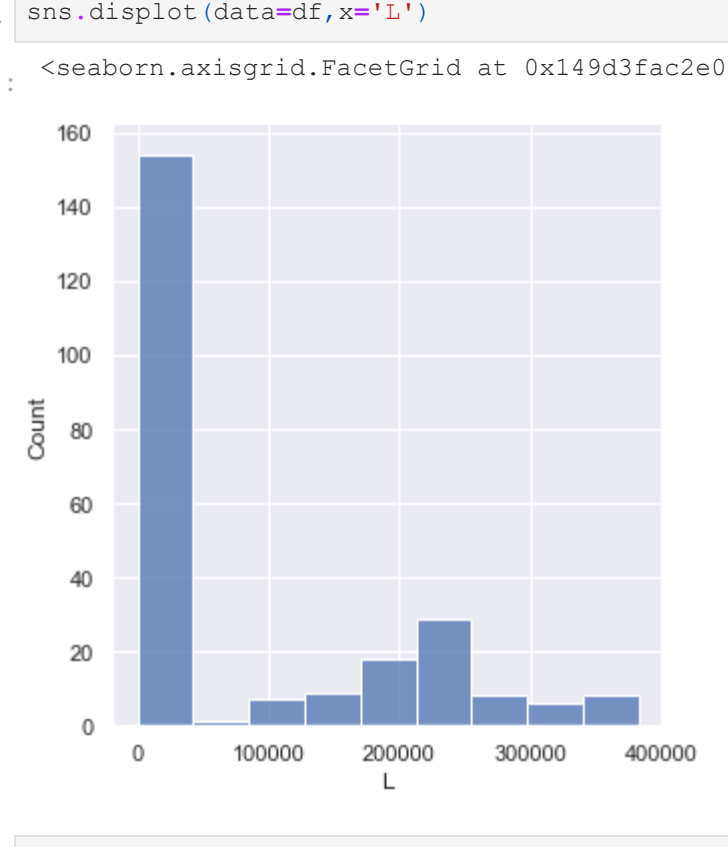
In [219…
```python
df['L'].isnull().sum()
```

Out[2198]:
```
15
```

In [219…
```python
df['L']=df.groupby('Type')['L'].apply(lambda l:l.fillna(l.mean()))
```

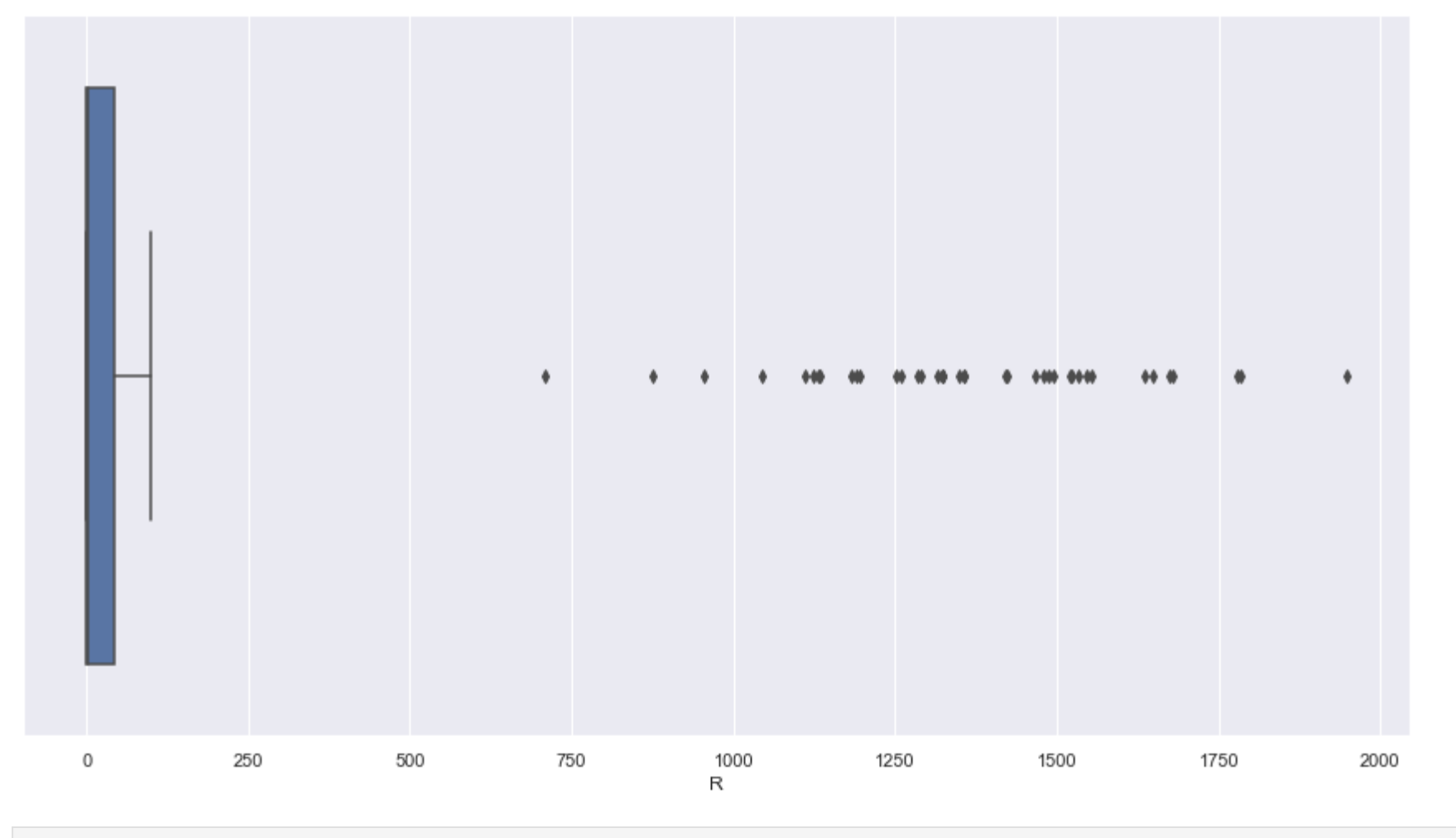
In [220…
```python
sns.displot(data=df,x='L')
```

Out[2200]:
```
<seaborn.axisgrid.FacetGrid at 0x149d3fac2e0>
```



In [220…
```python
sns.boxplot(data=df,x='R')
```

Out[2201]:
```
<AxesSubplot:xlabel='R'>
```



In [220…
```python
idx=detect_outliers(df,0,['R'])
```

In [220…
```python
for i in idx:
    df.loc[i,'R']=np.nan
```

In [220…
```python
df['R'].unique()
```

Out[2204]:
```
array([1.700e-01, 1.542e-01, 1.020e-01, 1.600e-01, 1.030e-01, 1.100e-01,
       1.270e-01, 9.600e-02, 1.300e-01, 5.100e-01, 3.761e-01, 1.960e-01,
       3.930e-01, 1.400e-01, 2.400e-01, 4.700e-01, 1.967e-01, 3.510e-01,
       8.400e-03, 1.234e-02, 1.100e-02, 1.000e-02, 1.400e-02, 9.840e-03,
       9.700e-03, 1.280e-02, 1.300e-02, 1.060e+01, 6.300e+00, 7.200e+00,
       2.890e+00, 9.000e-01, 1.800e+00, 1.120e+00, 9.800e-01, 1.106e+00,
       9.900e-01, 1.900e+01, 2.300e+01, 8.800e+01, 1.700e+01, 2.500e+01,
       2.900e+01, 4.500e+01, 8.900e+01, 8.400e+01, 2.600e+01,       nan,
       5.700e-02, 1.900e-01, 9.400e-02, 9.180e-02, 1.160e-01, 1.320e-01,
       9.300e-02, 9.110e-02, 1.180e-01, 1.200e-01, 2.730e-01, 3.800e-01,
       1.800e-01, 3.500e-01, 2.910e-01, 3.070e-01, 9.800e-02, 1.610e-01,
       1.200e-02, 9.200e-03, 9.500e-03, 1.500e-02, 8.900e-03, 1.090e-02,
       8.700e-03, 8.920e-03, 1.310e-02, 9.100e-01, 8.650e-01, 7.950e-01,
       1.130e+00, 2.630e+00, 6.980e+00, 8.760e-01, 1.340e+00, 7.010e+00,
       1.020e+01, 1.200e+01, 1.400e+01, 3.600e+01, 8.300e+01, 4.200e+01,
       6.300e+01, 5.700e+01, 6.700e+01, 9.120e-02, 9.760e-02, 9.320e-02,
       1.090e-01, 1.210e-01, 8.990e-02, 7.820e-02, 8.560e-02, 4.920e-01,
       3.100e-01, 3.400e-01, 6.200e-01, 7.300e-01, 5.600e-01, 2.560e-01,
       9.810e-03, 1.120e-02, 9.340e-03, 9.720e-03, 8.980e-03, 8.790e-03,
       9.450e-03, 8.780e-03, 9.960e-03, 6.020e+00, 2.487e+00, 5.745e+00,
       5.680e+00, 5.920e+00, 6.400e+00, 5.490e+00, 6.780e+00, 6.210e+00,
       6.860e+00, 9.700e+01, 3.500e+01, 2.400e+01, 7.300e+01, 3.900e+01,
       3.000e+01, 9.150e-02, 1.260e-01, 9.870e-02, 7.730e-02, 9.730e-02,
       1.190e-01, 9.980e-02, 1.120e-01, 1.290e-01, 1.480e-01, 4.600e-01,
       2.800e-01, 3.780e-01, 3.190e-01, 6.700e-01, 2.580e-01, 3.360e-01,
       4.710e-01, 6.750e-01, 1.210e-02, 1.140e-02, 1.130e-02, 1.160e-02,
       1.240e-02, 1.420e-02, 1.270e-02, 1.040e-02, 1.060e-02, 9.980e-03,
       6.360e+00, 5.856e+00, 5.992e+00, 6.640e+00, 6.390e+00, 6.237e+00,
       1.910e+00, 6.030e+00, 5.653e+00, 1.930e+00, 7.600e+01, 9.800e+01,
       8.100e+01, 6.200e+01, 4.600e+01, 8.000e+01, 8.600e+01, 9.200e+01])
```

In [220…
```python
df[df['R'].isna()==True]
```

Out[2205]:

	Temperature	L	R	A_M	Color	Spectral_Class	Type
50	3490.000000	270000.0000	NaN	-9.40	Red	M	Hypergiant
51	3750.000000	283000.0000	NaN	-7.63	Red	M	Hypergiant
52	3834.000000	272000.0000	NaN	-9.20	Red	M	Hypergiant
53	3749.000000	223970.1875	NaN	-8.05	Orange	M	Hypergiant
54	3650.000000	310000.0000	NaN	-7.79	Red	M	Hypergiant
55	3450.000000	263000.0000	NaN	-11.75	Red	M	Hypergiant
56	3660.000000	363000.0000	NaN	-11.92	Red	M	Hypergiant
57	3450.000000	174000.0000	NaN	-11.28	Red	M	Hypergiant
58	3752.000000	209000.0000	NaN	-11.24	Red	M	Hypergiant
59	3535.000000	195000.0000	NaN	-11.36	Red	M	Hypergiant
110	3459.000000	100000.0000	NaN	-10.70	Red	M	Hypergiant
111	3605.000000	126000.0000	NaN	-10.81	Red	M	Hypergiant
112	3615.000000	200000.0000	NaN	-11.33	Red	M	Hypergiant
113	3399.000000	117000.0000	NaN	-10.92	Red	M	Hypergiant
114	3610.000000	132000.0000	NaN	-10.86	Red	M	Hypergiant
115	3553.000000	145000.0000	NaN	-11.03	Red	M	Hypergiant
116	4015.000000	282000.0000	NaN	-11.39	Red	K	Hypergiant
117	3625.000000	74000.0000	NaN	-10.25	Red	M	Hypergiant
118	6850.000000	229000.0000	NaN	-10.07	Red	G	Hypergiant
119	3780.000000	200000.0000	NaN	-10.70	Red	M	Hypergiant
170	3570.000000	320000.0000	NaN	-7.58	Red	M	Hypergiant
171	3500.000000	138000.0000	NaN	-8.18	Red	M	Hypergiant
172	4287.000000	223970.1875	NaN	-9.20	Orange	K	Hypergiant
173	26000.000000	316000.0000	NaN	-9.10	Blue	B	Hypergiant
174	3600.000000	240000.0000	NaN	-7.89	Red	M	Hypergiant
175	3614.000000	145000.0000	NaN	-7.71	Red	M	Hypergiant
176	18000.000000	200000.0000	NaN	-8.30	Blue	O	Hypergiant
177	11000.000000	170000.0000	NaN	-9.90	Blue-white	B	Hypergiant
178	12100.000000	120000.0000	NaN	-7.84	Blue-white	B	Hypergiant
179	24490.000000	248490.0000	NaN	-8.24	Blue-white	B	Hypergiant
230	24145.000000	382993.0000	NaN	-8.84	Blue-white	B	Hypergiant
231	7356.029412	272830.0000	NaN	-9.29	Blue	O	Hypergiant
232	7356.029412	223970.1875	NaN	-10.84	Blue	O	Hypergiant
233	7356.029412	223970.1875	NaN	-7.59	Blue-white	B	Hypergiant
234	21904.000000	223970.1875	NaN	-7.67	Blue-white	B	Hypergiant
235	7356.029412	374830.0000	NaN	-9.93	Blue	O	Hypergiant
236	7356.029412	223970.1875	NaN	-10.63	Blue	O	Hypergiant
237	8829.000000	223970.1875	NaN	-10.73	White	A	Hypergiant
238	9235.000000	223970.1875	NaN	-11.23	White	A	Hypergiant
239	7356.029412	294903.0000	NaN	-7.80	Blue	O	Hypergiant

In [220…
```python
from sklearn.impute import KNNImputer
```

In [220…
```python
imputer=KNNImputer(n_neighbors=5)
```

In [220…
```python
df_impute=imputer.fit_transform(df[['R']])
```

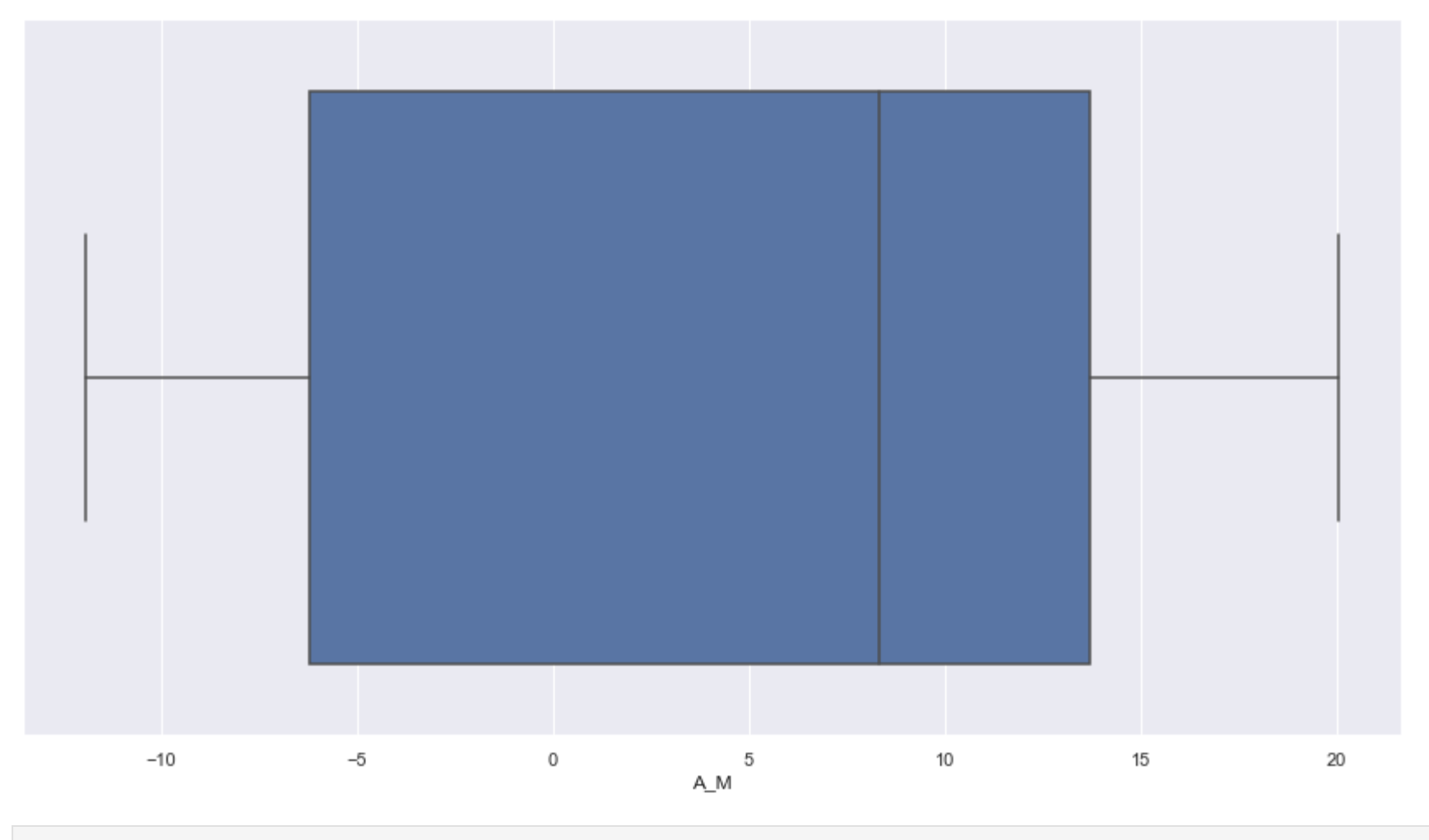
In [220…
```python
df['R']=pd.DataFrame(df_impute,columns=imputer.get_feature_names_out())
```

In [221…
```python
df.isnull().sum()
```

Out[2210]:
```
Temperature       0
L                 0
R                 0
A_M               0
Color             0
Spectral_Class    0
Type              0
dtype: int64
```

In [221…
```python
sns.boxplot(data=df,x='A_M')
#not outliers here
```

Out[2211]:
```
<AxesSubplot:xlabel='A_M'>
```



In [221…
```python
df['A_M']=np.abs(df['A_M'])
```

In [221…
```python
df['A_M'].describe()
```

Out[2213]:
```
count    240.000000
mean      10.308121
std        4.849497
min        0.013000
25%        6.243750
50%       10.880000
75%       13.697500
max       20.060000
Name: A_M, dtype: float64
```

In [221…
```python
#double check
df.duplicated().sum()
```

Out[2214]:
```
0
```

In [221…
```python
df.isnull().mean()
```

Out[2215]:
```
Temperature       0.0
L                 0.0
R                 0.0
A_M               0.0
Color             0.0
Spectral_Class    0.0
Type              0.0
dtype: float64
```

In [221…
```python
# let's check the correlation
```

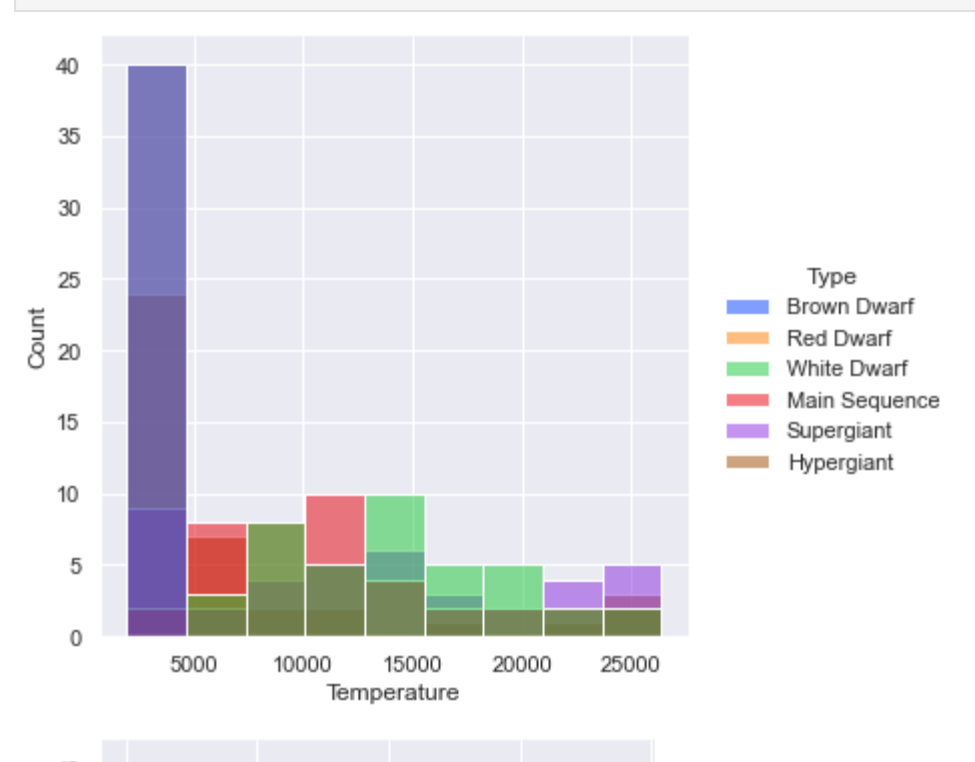
In [221…
```python
df.columns
```

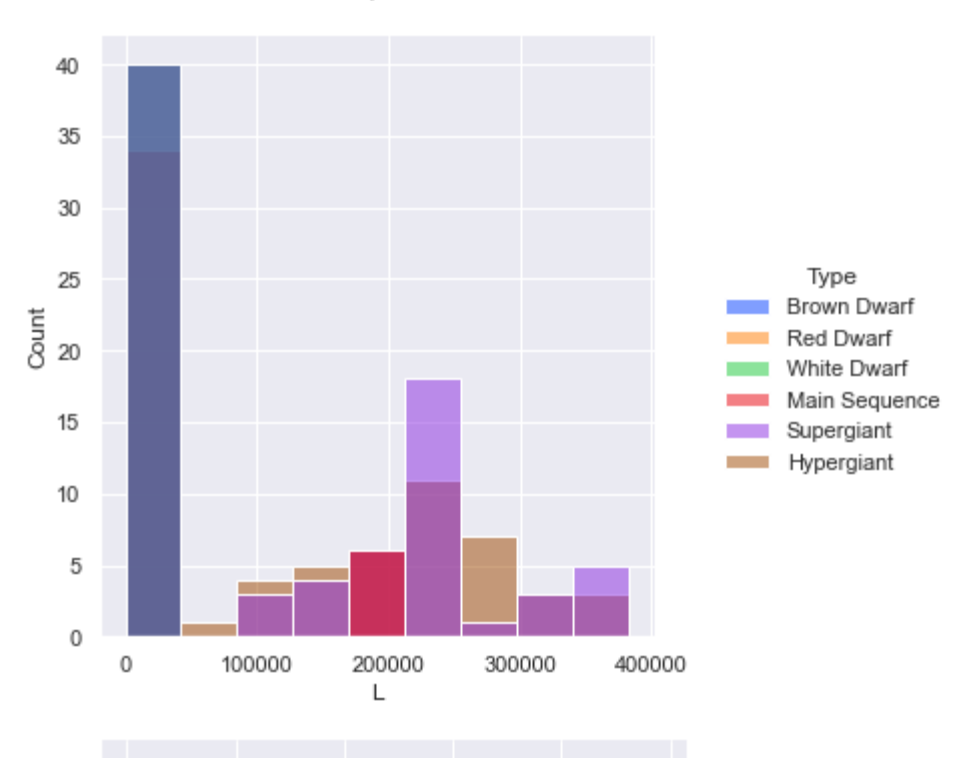
Out[2217]:
```
Index(['Temperature', 'L', 'R', 'A_M', 'Color', 'Spectral_Class', 'Type'], dtype='object')
```

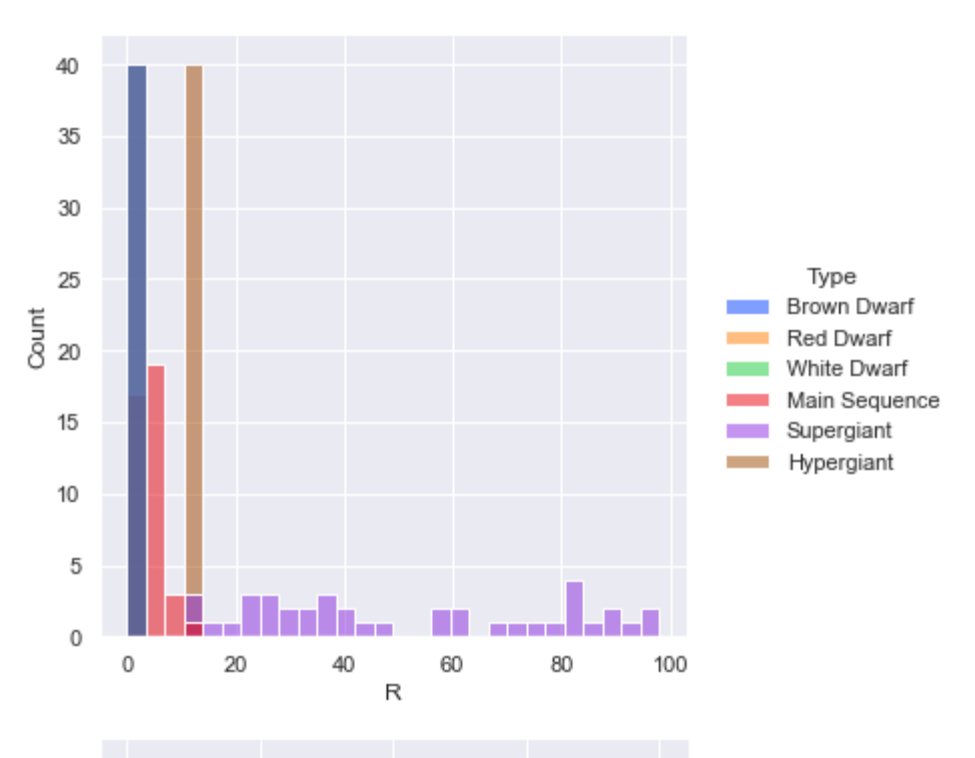
In [221…
```python
numeric_columns
```

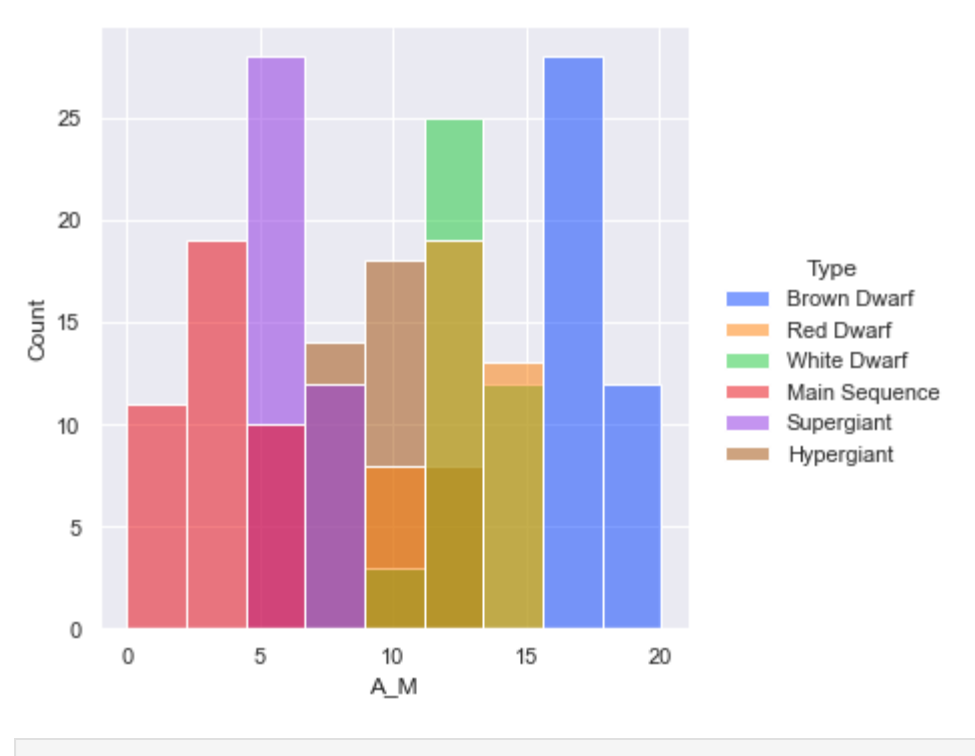
Out[2218]:
```
Index(['Temperature', 'L', 'R', 'A_M'], dtype='object')
```

In [221…
```python
for feature in numeric_columns:
    sns.displot(data=df,x=feature,hue='Type',palette='bright')
```









In [222…
```python
df.columns
```

Out[2220]:
```
Index(['Temperature', 'L', 'R', 'A_M', 'Color', 'Spectral_Class', 'Type'], dtype='object')
```

In [222…
```python
sns.scatterplot(data=df,x='Temperature',y='L',hue='Type',palette='bright')
```

```
Out[2221]: <AxesSubplot:xlabel='Temperature', ylabel='L'>
```

```
In [222… sns.scatterplot(data=df,x='Temperature',y='R',hue='Type',palette='bright')
Out[2222]: <AxesSubplot:xlabel='Temperature', ylabel='R'>
```

```
In [222… sns.scatterplot(data=df,x='Temperature',y='A_M',hue='Type',palette='bright')
Out[2223]: <AxesSubplot:xlabel='Temperature', ylabel='A_M'>
```

```
In [222… sns.scatterplot(data=df,x='R',y='L',hue='Type',palette='bright')
Out[2224]: <AxesSubplot:xlabel='R', ylabel='L'>
```

```
In [222… sns.countplot(data=df,x='Color',hue='Type')
         plt.xticks(rotation=90)
         plt.show()
```

```
In [222… sns.countplot(data=df,x='Spectral_Class',hue='Type')
Out[2226]: <AxesSubplot:xlabel='Spectral_Class', ylabel='count'>
```

```
In [222… ## Category Encoding

In [222… df['Spectral_Class'].unique()
Out[2228]: array(['M', 'B', 'A', 'F', 'O', 'K', 'G'], dtype=object)

In [222… spectral_dict={
             "A":1,
             "B":2,
             "F":3,
             "G":4,
             "O":5,
             "M":6,
             "K":7
         }

In [223… df['Spectral_Class']=df['Spectral_Class'].map(spectral_dict)

In [223… from sklearn.preprocessing import OneHotEncoder,LabelEncoder
         from category_encoders import BinaryEncoder

In [223… target_encoder=LabelEncoder()

In [223… df['Type']=target_encoder.fit_transform(df['Type'])

In [223… df['Type'].value_counts()
Out[2234]: 0    40
           3    40
           5    40
           2    40
           4    40
           1    40
           Name: Type, dtype: int64

In [223… color_encoder = BinaryEncoder()

In [223… df_=color_encoder.fit_transform(df[['Color']])

In [223… df=pd.concat([df,df_],axis=1)

In [223… df.drop('Color',axis=1,inplace=True)

In [223… df
Out[2239]:
```

	Temperature	L	R	A_M	Spectral_Class	Type	Color_0	Color_1	Color_2	Color_3	Color_4
0	3068.000000	0.002400	0.170000	16.12	6	0	0	0	0	0	1
1	3042.000000	0.000500	0.154200	16.60	6	0	0	0	0	0	1
2	2600.000000	0.000300	0.102000	18.70	6	0	0	0	0	0	1
3	2800.000000	0.000200	0.160000	16.65	6	0	0	0	0	0	1
4	1939.000000	0.000138	0.103000	20.06	6	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
235	7356.029412	374830.000000	11.209838	9.93	5	1	0	0	1	1	1
236	7356.029412	223970.187500	11.209838	10.63	5	1	0	0	1	1	1
237	8829.000000	223970.187500	11.209838	10.73	1	1	0	0	0	1	1
238	9235.000000	223970.187500	11.209838	11.23	1	1	0	0	0	1	1
239	7356.029412	294903.000000	11.209838	7.80	5	1	0	0	1	1	1

240 rows × 11 columns

```
In [224… from  sklearn.model_selection import train_test_split

In [224… x=df.drop('Type',axis=1)
         y=df['Type']

In [224… x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,stratify=y,random_state=7)

In [224… from sklearn.preprocessing import StandardScaler

In [224… scaler=StandardScaler()

In [224… columns_to_scaled=['Temperature','L','R','A_M']

In [224… x_train[columns_to_scaled]=scaler.fit_transform(x_train[columns_to_scaled])

In [224… x_test[columns_to_scaled]=scaler.transform(x_test[columns_to_scaled])

In [224… from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score,confusion_matrix,ConfusionMatrixDisplay

In [224… Logistic_model=LogisticRegression(multi_class='ovr',random_state=12)

In [225… Logistic_model.fit(x_train,y_train)
Out[2250]: ▾          LogisticRegression
           LogisticRegression(multi_class='ovr', random_state=12)

In [225… y_train_pre=Logistic_model.predict(x_train)
         y_test_pre=Logistic_model.predict(x_test)

In [225… print("The Train dataset",accuracy_score(y_train,y_train_pre)*100,"%")
         print("The Test dataset",accuracy_score(y_test,y_test_pre)*100,"%")

         The Train dataset 95.09803921568627 %
         The Test dataset 100.0 %

In [225… cm = confusion_matrix(y_train , y_train_pre )
         ConfusionMatrixDisplay(cm , display_labels=target_encoder.inverse_transform(Logistic_model.classes_)).plot()
         plt.show()
```

```
In [225… Logistic_model.predict_proba(x_test)
Out[2255]: array([[7.66959157e-01, 8.60101577e-02, 3.19457576e-05, 9.14915947e-02,
                    3.52020040e-04, 5.51551244e-02],
                   [1.17018402e-05, 1.62827037e-01, 4.50165188e-01, 5.12658726e-03,
                    3.81566404e-01, 3.03081641e-04],
                   [2.01656633e-05, 9.69630888e-02, 4.17095703e-02, 6.73652293e-05,
                    8.60769735e-01, 4.70075352e-04],
                   [1.28503340e-03, 9.98716698e-03, 1.84755115e-02, 1.56486337e-03,
                    5.87258681e-03, 9.62814838e-01],
                   [7.48783503e-02, 5.18023769e-02, 1.72590550e-02, 8.47133085e-01,
                    5.96714917e-03, 2.95998377e-03],
                   [1.21165729e-05, 4.09207822e-01, 2.45059301e-02, 1.79934116e-05,
                    5.66215234e-01, 4.09043302e-05],
                   [9.05269611e-08, 1.11921018e-02, 7.10035090e-01, 2.36663375e-01,
                    3.74440020e-02, 4.66534017e-03],
                   [3.31596416e-07, 1.31625939e-02, 9.20545667e-01, 1.83501486e-02,
                    2.47223026e-02, 2.32189566e-02],
                   [1.17111210e-02, 2.66503447e-02, 1.34769847e-03, 1.82477783e-03,
                    1.80764629e-03, 9.56658412e-01],
                   [6.91183709e-01, 6.43803066e-02, 3.27538067e-04, 2.25199141e-01,
                    9.34137012e-04, 1.79751682e-02],
                   [3.60615279e-08, 1.16113980e-02, 8.94856077e-01, 2.30983618e-02,
                    3.73760276e-02, 3.30580996e-02],
                   [1.34918370e-02, 8.55751166e-01, 7.01268047e-03, 3.88266321e-02,
                    8.48577803e-02, 5.99039980e-05],
                   [3.62450699e-05, 1.36289668e-01, 3.94512104e-02, 2.45531772e-04,
                    8.23854736e-01, 1.22608364e-04],
                   [1.86954431e-03, 2.98946529e-02, 2.53805457e-02, 9.86646075e-04,
                    1.70049766e-03, 9.40168113e-01],
                   [1.18832812e-05, 4.34015311e-03, 4.51414878e-03, 1.01627596e-05,
                    9.91018395e-01, 1.05256659e-04],
                   [4.58877091e-05, 1.90443988e-01, 8.38964925e-02, 7.01414354e-04,
                    7.24681486e-01, 2.30731883e-04],
                   [5.32602377e-04, 7.79355204e-01, 2.57546525e-02, 1.50941416e-02,
                    1.79260477e-01, 2.92227393e-06],
                   [3.33386282e-03, 1.20810918e-02, 9.41963358e-03, 1.47373401e-03,
                    4.58683108e-03, 9.69104847e-01],
                   [7.73869961e-01, 8.05270459e-02, 4.37609923e-05, 9.25303676e-02,
                    3.99088033e-04, 5.26297769e-02],
                   [6.35520220e-03, 4.00845525e-02, 2.86219519e-03, 2.21261788e-03,
                    1.75349528e-03, 9.46731937e-01],
                   [6.98795792e-04, 7.55610867e-01, 5.21694311e-03, 2.61898243e-05,
                    2.38322947e-01, 1.24257236e-04],
                   [2.96279977e-02, 4.26387329e-02, 3.43620509e-02, 8.83722659e-01,
                    7.83564895e-03, 1.81291091e-03],
                   [4.50798357e-08, 6.92073520e-03, 6.82350089e-01, 2.13308849e-01,
                    6.37227614e-02, 3.36975208e-02],
                   [2.72622029e-03, 8.53951481e-01, 8.82442362e-03, 1.17965367e-02,
                    1.22692996e-01, 8.34252599e-06],
                   [1.96836814e-02, 3.91931517e-02, 4.56677435e-02, 8.85521581e-01,
                    8.51457993e-03, 1.41926196e-03],
                   [3.88123953e-04, 2.18844139e-02, 2.00096188e-02, 1.02934649e-01,
                    8.54774898e-01, 8.29637995e-06],
                   [1.93086904e-02, 8.78618870e-01, 4.28274306e-03, 2.53929475e-02,
                    7.23345199e-02, 6.22288819e-05],
                   [4.52618694e-01, 6.29683526e-02, 1.85491468e-03, 4.71837116e-01,
                    2.16141555e-03, 8.55950727e-03],
                   [1.52908975e-01, 5.82537167e-02, 9.17580994e-03, 7.70400034e-01,
                    4.79292162e-03, 4.46854248e-03],
                   [6.68558508e-01, 6.33636854e-02, 4.00187417e-04, 2.51040261e-01,
                    1.03171107e-03, 1.56056466e-02],
                   [7.70410121e-01, 7.95457135e-02, 5.06076883e-05, 1.02582528e-01,
                    4.24665898e-04, 4.69863643e-02],
                   [2.07511831e-03, 3.22024803e-02, 2.69207111e-02, 1.32621068e-03,
                    1.80044993e-03, 9.35675030e-01],
                   [7.70860173e-01, 8.07167108e-02, 4.61149374e-05, 9.85217402e-02,
                    4.06041701e-04, 4.94492199e-02],
                   [9.32225707e-04, 7.72994128e-01, 1.21589475e-01, 3.52825937e-03,
                    1.00941914e-01, 1.39980060e-05],
                   [4.08234530e-08, 1.32501784e-02, 8.93200374e-01, 3.16830756e-02,
                    3.34941497e-02, 2.83721817e-02],
                   [3.72892994e-01, 6.33314058e-02, 2.95105369e-03, 5.49797013e-01,
                    2.93191548e-03, 8.09561825e-03]])
```