In [1]:

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

In [2]:

```
cd Desktop
```

```
C:\Users\user\Desktop
```

# Conventional Relative Grading System

```
taking mean =m
standard deviation=s
marks=x
for
cutoff for Fail--------->F
m-s>x>=m-1.5s----------->C-
m-.5s>x>=m-s------------>C
m>x>=m-.5s-------------->B-
m+.5s>x>=m ------------->B
m+ s>x>=m+.5s----------->A-
m+1.5s>x>=m+s----------->A
Cutoff for A*----------->A*
```

In [3]:

```
file = input('file name')
```

```
file namedata.txt
```

There are 8 grades : A*, A, A-, B, B-, C, C-, F

among which A* and F is chosen absolutely unlike other grades.

In [4]:

```
astar = input('cutoff for a*')
```

```
cutoff for a*91
```

In [5]:

```
f = input('passing cutoff')
```

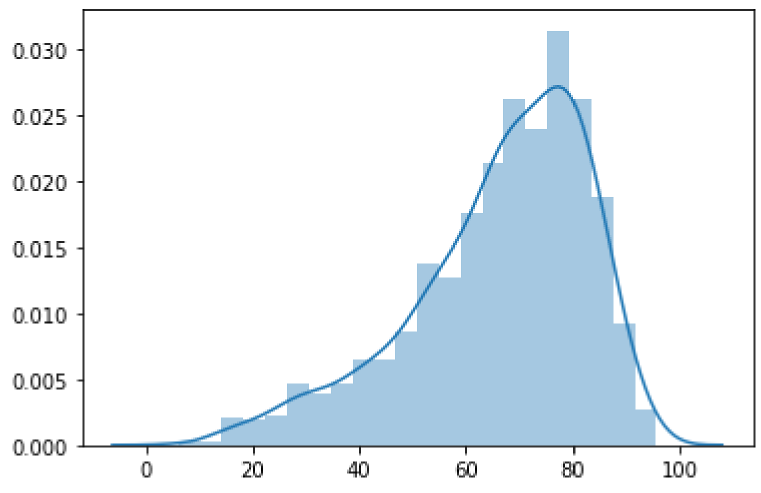
```
passing cutoff27
```

In [6]:

```
marks = pd.read_csv('data.txt')
sns.distplot(marks)
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x241ffba30c8>
```

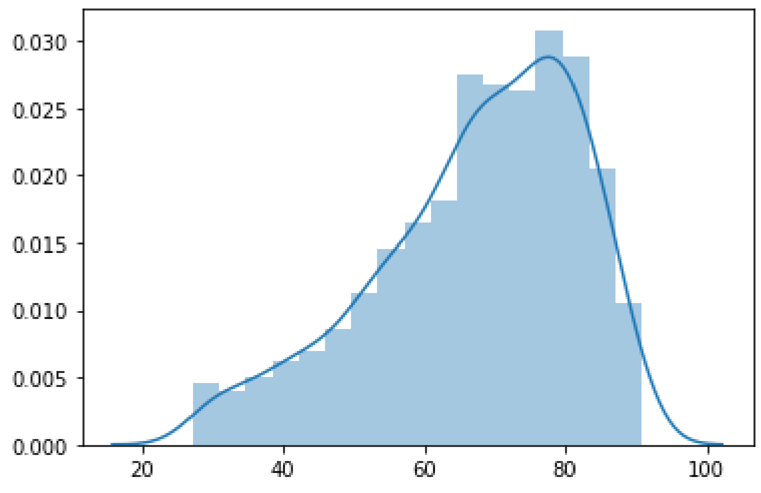


In [41]:

```
marks = marks[(marks['Marks'] > 27) & (marks['Marks'] < 91)]
sns.distplot(marks)
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x241892a2448>
```



Insight to the data

1. The above histogram shows us that majority of the marks is lying arround 67. This means that B grade, which is considered as average should be assigned somewhere over here.
2. A slight variation in marks near the average segment increases the percentile by a huge amount rather than that in the lower segment. This means that it's difficult to score any more marks past the average barrier, so it will not be fair to assign grades in a uniformly divided marks metrics.

Straightaway applying unsupervised learning algorithm to the marks with 6 clusters will not be appropriate because:

1. One unit marks will hold more value towards the 80-100 range than in 27-60 range.
2. Change in the percentile per unit marks is higher in the later range

Should the grading be solely dependent on the marks unit?

It seems there are 2 factors at work, (i) marks, (ii) Density of Marks(can be given by percentile)

Let's have a percentile vs marks data

In [42]:

```
m = pd.Series(sorted(marks['Marks']))
l = len(m)
```

In [43]:

```
df = pd.DataFrame(m, columns = ['Marks'])
```

In [44]:

```
df['percentile']  =m.reset_index()['index'].apply(lambda x: ((x/l))*100)
```

In [91]:

```
df['pfa'] =np.sqrt( (df.Marks)**2 + (df.percentile)**2) /100
```

In [53]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 6)
```

In [76]:

```
df['dimension'] = np.ones(l)
kmeans.fit(df[['Marks', 'percentile']])
```

Out[76]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [77]:

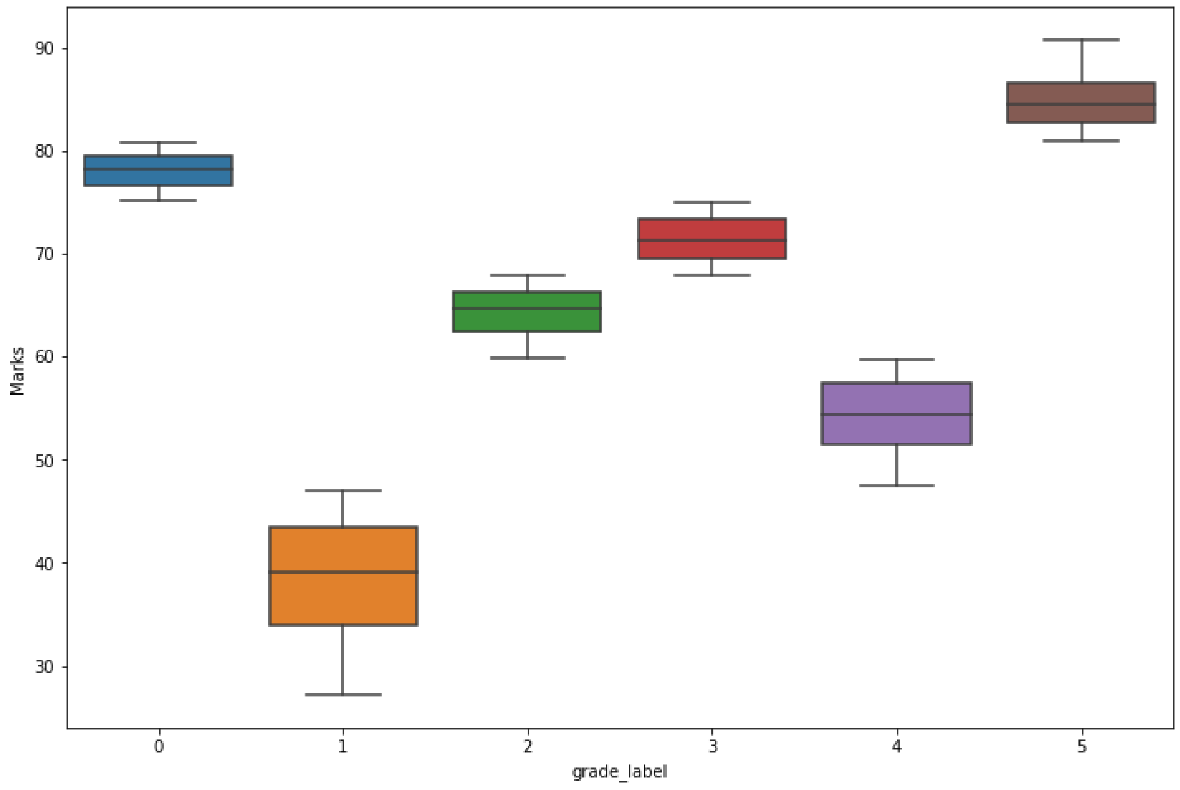
```
df['grade_label'] = kmeans.labels_
```

In [80]:

```
plt.figure(figsize = (12,8))
sns.boxplot(x="grade_label", y="Marks", data=df)
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2418be51288>
```



In [72]:

```
kmeans.fit(df[['Marks', 'dimension']])
```

Out[72]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [73]:

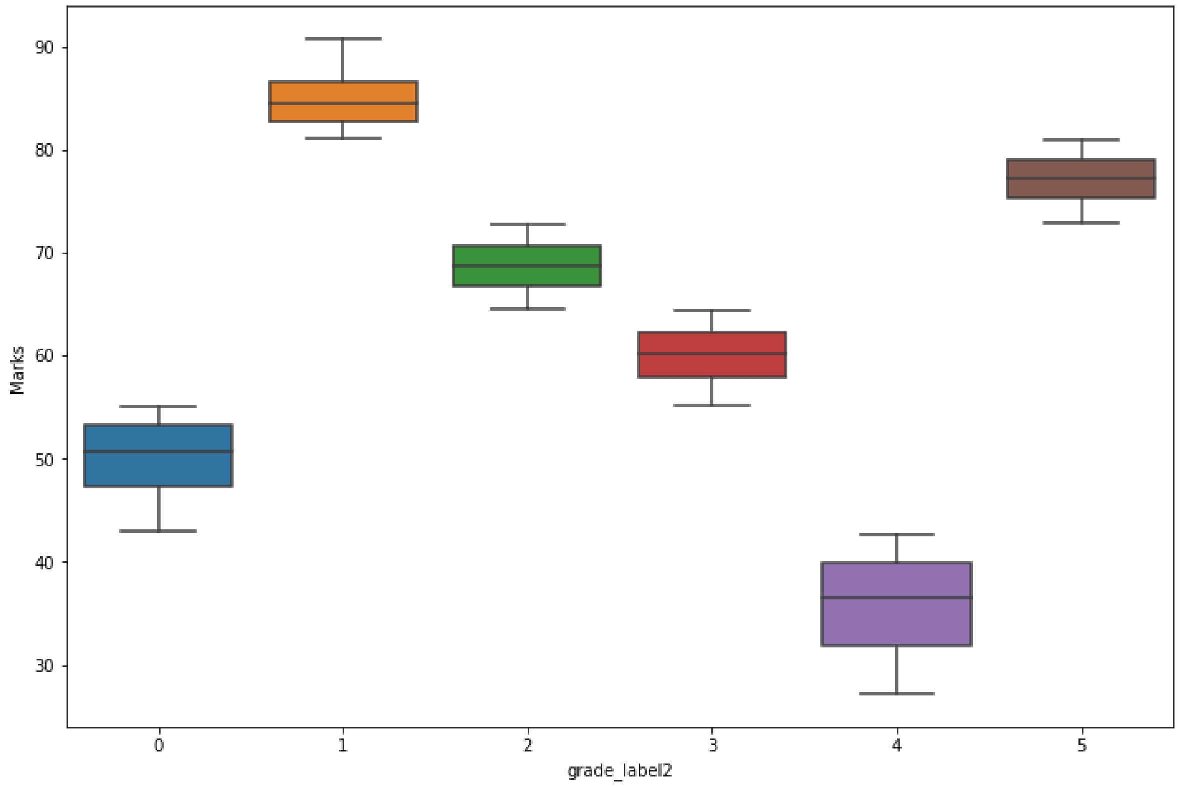
```
df['grade_label2'] = kmeans.labels_
```

In [83]:

```
plt.figure(figsize = (12,8))
sns.boxplot(x="grade_label2", y="Marks", data=df)
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2418c306b48>
```



In [ ]:

```

```