In [4]:
```
import numpy as np
import matplotlib.pyplot as mp
from pylab import show
from sklearn.cluster import KMeans
```

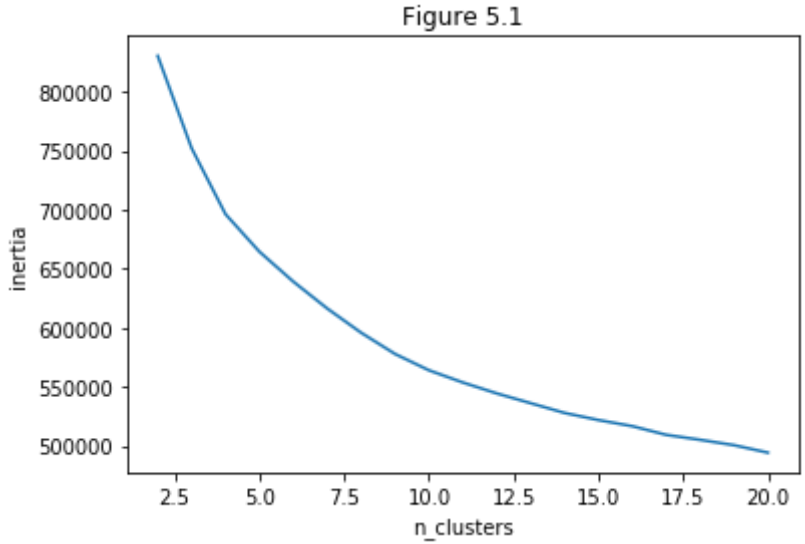
In [34]:
```
data = np.loadtxt("data.csv")

#remove the first item, which is the label
labels = []
for row in data:
    labels.append(row[0])
    row = row[1:]
```

In [29]:
```
x_val = []
y_val = []

for i in range (2,21,1):
    x_val.append(i)
    kmeans = KMeans(n_clusters=i, init="random").fit(data)
    y_val.append(kmeans.inertia_)
```

In [32]:
```
mp.plot(x_val,y_val)
mp.title("Figure 5.1")
mp.xlabel("n_clusters")
mp.ylabel("inertia")
mp.show()
```

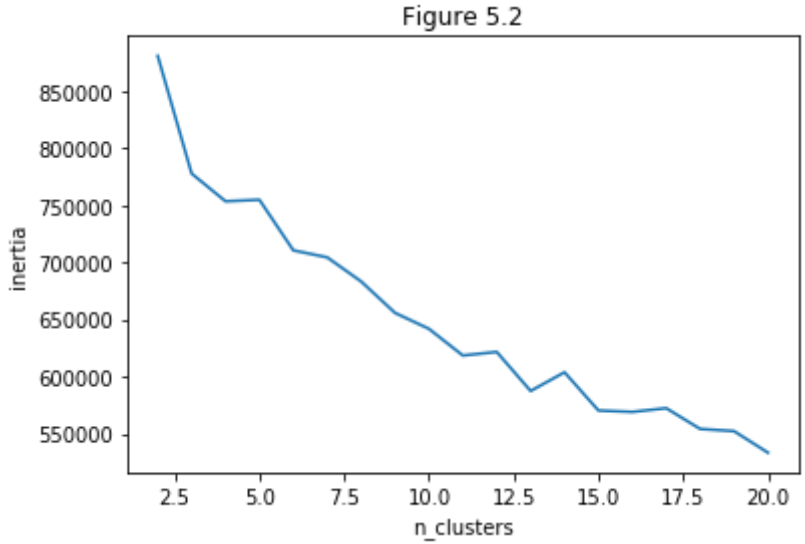


b.The optimal number of clusters would be approximately 10, which coincides with our expectations, since we should expect to only see 10 distinct digits.

In [35]:
```
x_val = []
y_val = []

for i in range (2,21,1):
    x_val.append(i)
    kmeans = KMeans(n_clusters=i, init="random", n_init=1, max_iter=1).fit(data)
    y_val.append(kmeans.inertia_)
```

In [36]:
```
mp.plot(x_val,y_val)
mp.title("Figure 5.2")
mp.xlabel("n_clusters")
mp.ylabel("inertia")
mp.show()
```

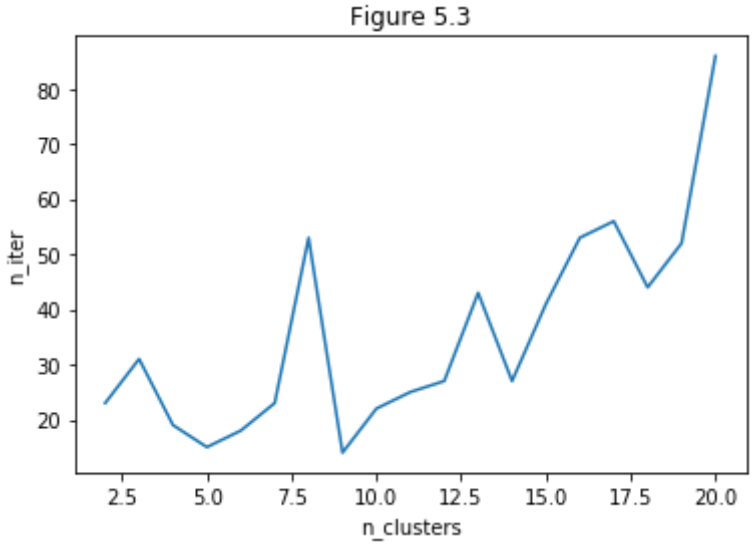


d. Figure 5.2 is significantly more jagged than Figure 5.1. This is understandable in that our number of iterations has significantly decreased. Also, the number of times we initialize kmeans has also decreased, giving us a more varied view of the inertia. Figure 5.1 is a more smoothed out version.

In [37]:
```
x_val = []
y_val = []

for i in range (2,21,1):
    x_val.append(i)
    kmeans = KMeans(n_clusters=i, init="random").fit(data)
    y_val.append(kmeans.n_iter_)
```

In [38]:
```
mp.plot(x_val,y_val)
mp.title("Figure 5.3")
mp.xlabel("n_clusters")
mp.ylabel("n_iter")
mp.show()
```



f. As we increase the number of clusters, we would expect to need more iterations to be able to align our data points closer to their respective clusters. While the Figure 5.3 shows a bit of jagged nature, it does show a trendline upward where n_clusters and n_iter are positively related. This is within our expectations.