KMeans

In [14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
import seaborn as sns
```

In [15]:
```
def kmean(k,x,random_state=0):
    s = x.shape[0]
    m = np.random.seed(random_state)
    r = np.random.choice(s,k,replace = False)
    centroids = x[r]
    while True:
        old_centroids = centroids.copy()
        dist = cdist(x,centroids)
        label = np.argmin(dist,axis=1)
        for cluster_id in range(k):
            cluster = x[label == cluster_id]
            if len(cluster):
                centroids[cluster_id] = cluster.mean(axis=0)
        if np.all(old_centroids != centroids):
            break
    return (centroids,label)
```

Evaluation Matrics

In [16]:
```
def distortion(x,label,centroids):
    return np.linalg.norm(x-(centroids[label]),axis=-1).mean()

def silhouette_score(k,x,label):
    y = np.array([])
    for i in range(k):
        c = x[label == i]
        if len(c):
            y = np.append(y,cdist(c,c).mean())
    a=y.mean()
    z = np.array([])
    for i in range(k):
        c = x[label == i]
        v = np.array([])
        for j in range(k):
            if i != j:
                d = x[label == j]
                if len(d) and len(c):
                    v = np.append(v,cdist(c,d).mean())
        z = np.append(z,np.min(v))
    b=z.mean()
    return (b-a)/np.max(np.dstack([a,b]),axis=-1).mean()
```

Loading the Iris dataset

In [17]:
```
names = ["s_len","s_wid","p_len","p_wid","species"]
```

In [18]:
```
x_data = pd.read_csv("C:\\Users\\ASUS\\Iris.csv",names = names)
x = x_data.drop("species",axis=1).values
```

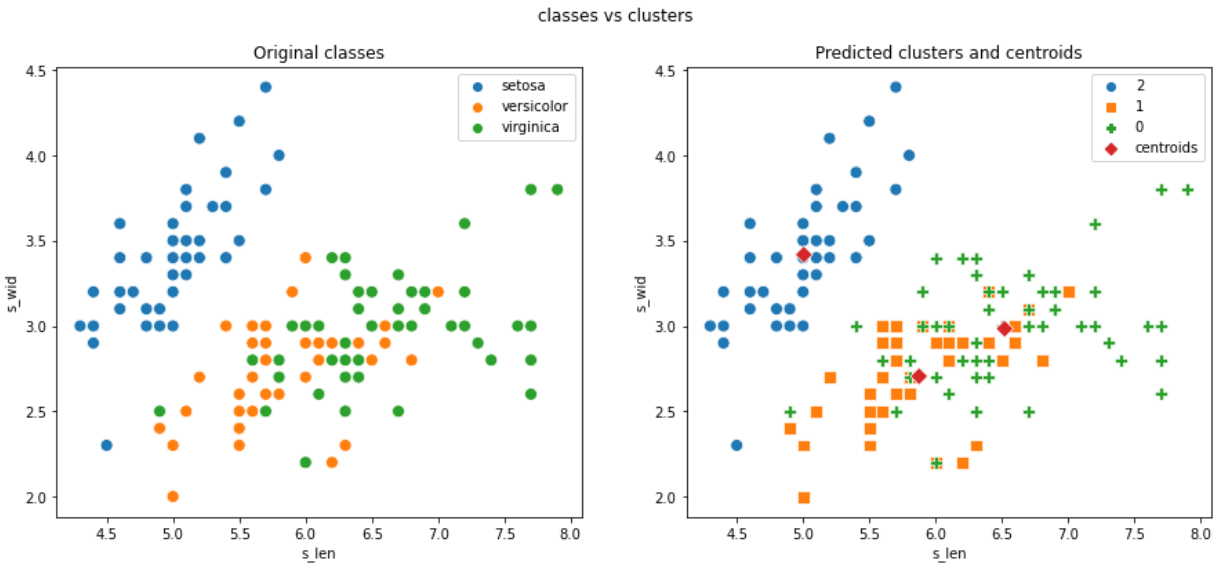
Implementation of KMeans

In [19]:
```
centroids,label = kmean(3,x)
```

In [20]:
```
all_df = x_data.copy(deep=True)
clusters = pd.DataFrame(centroids,columns=names[:-1])
clusters["cluster"] = "centroids"
all_df["cluster"] = label.astype("str")
all_df = pd.concat([all_df,clusters])
```

Comparing the clustering results with original classes

In [21]:
```
plt.figure(figsize=(15,6))
plt.suptitle("classes vs clusters")
plt.subplot(121)
plt.title("Original classes")
sns.scatterplot(data=all_df,x="s_len",y="s_wid",hue="species",s=80)
plt.legend(loc="upper right")
plt.subplot(122)
plt.title("Predicted clusters and centroids")
sns.scatterplot(data=all_df,x="s_len",y="s_wid",hue="cluster",style = "clust
plt.legend(loc="upper right")
plt.show()
```



Testing with multiple values of k (Hyperparameter tuning)

In [22]:
```
sil_coefs = []
distortions = []
K = np.arange(2,6)
for k in K:
    centroids,label = kmean(k,x)
    avg_sil_coef = silhouette_score(k,x,label)
    dist = distortion(x,label,centroids)
    print(f"For k={k:<4} Avg.Sil.Coef: {avg_sil_coef:<10.5f} Distortion: {di
    distortions.append(dist)
    sil_coefs.append(avg_sil_coef)
```

```
For k=2    Avg.Sil.Coef: 0.53721    Distortion: 1.30604
For k=3    Avg.Sil.Coef: 0.59123    Distortion: 0.67576
For k=4    Avg.Sil.Coef: 0.55830    Distortion: 0.59402
For k=5    Avg.Sil.Coef: 0.43412    Distortion: 0.56721
```

Using Elbow method to find the optimal value of k

In [23]:
```
plt.plot(K, distortions, 's-', markersize=8)
plt.xlabel('k')
plt.xticks(K)
plt.ylabel('Distortion')
plt.title('Elbow Method for Finding Optimal k')
plt.show()
```