Cluster the given points based on their colors

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters.

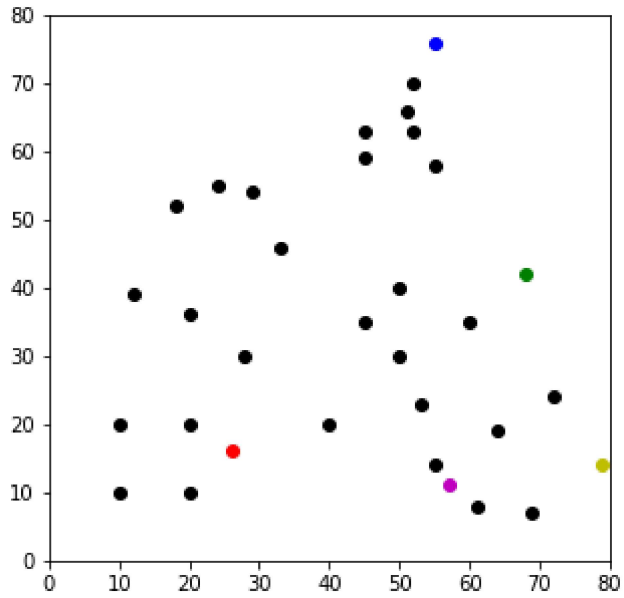
There are 3 steps:

Initialisation – K initial “means” (centroids) are generated at random Assignment – K clusters are created by associating each observation with the nearest centroid Update – The centroid of the clusters becomes the new mean Assignment and Update are repeated iteratively until convergence

The end result is that the sum of squared errors is minimised between points and their respective centroids.

We’ll do this manually first, then show how it’s done using scikit-learn

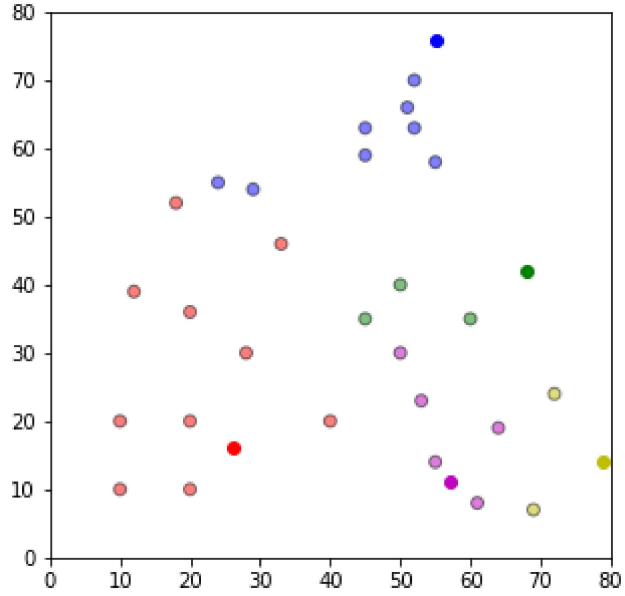
In [3]:
```
## Initialisation

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.DataFrame({
    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64,
69, 72, 50, 40, 45, 50, 60, 10, 10, 20, 20],
    'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7
, 24, 30, 20, 35, 40, 35, 20, 10, 10, 20]
})


np.random.seed(200)
k = 5
# centroids[i] = [x, y]
centroids = {
    i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]
    for i in range(k)
}

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap = {1: 'r', 2: 'g', 3: 'b', 4:'y', 5:'m'}
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```



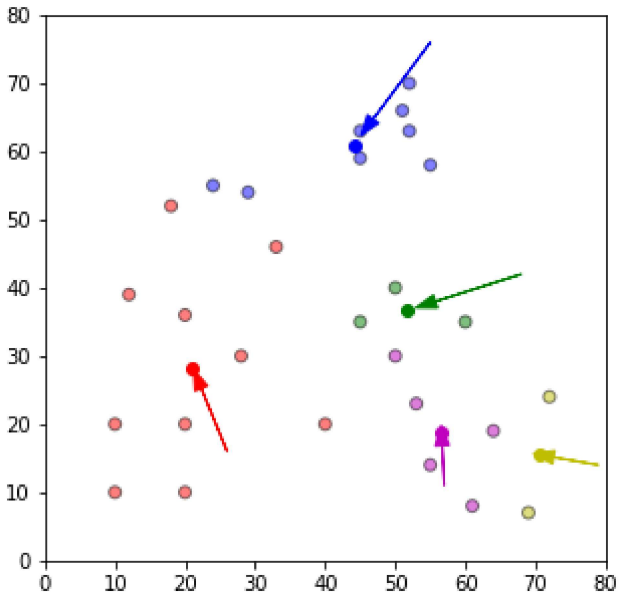
In [4]:
```
## Assignment Stage

def assignment(df, centroids):
    for i in centroids.keys():
        # sqrt((x1 - x2)^2 - (y1 - y2)^2)
        df['distance_from_{}'.format(i)] = (
            np.sqrt(
                (df['x'] - centroids[i][0]) ** 2
                + (df['y'] - centroids[i][1]) ** 2
            )
        )
    centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.
keys()]
    df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
    df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_'
)))
    df['color'] = df['closest'].map(lambda x: colmap[x])
    return df

df = assignment(df, centroids)
print(df.head())

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

	x	y	distance_from_1	...	distance_from_5	closest	color
0	12	39	26.925824	...	53.000000	1	r
1	20	36	20.880613	...	44.654227	1	r
2	28	30	14.142136	...	34.669872	1	r
3	18	52	36.878178	...	56.586217	1	r
4	29	54	38.118237	...	51.312766	3	b

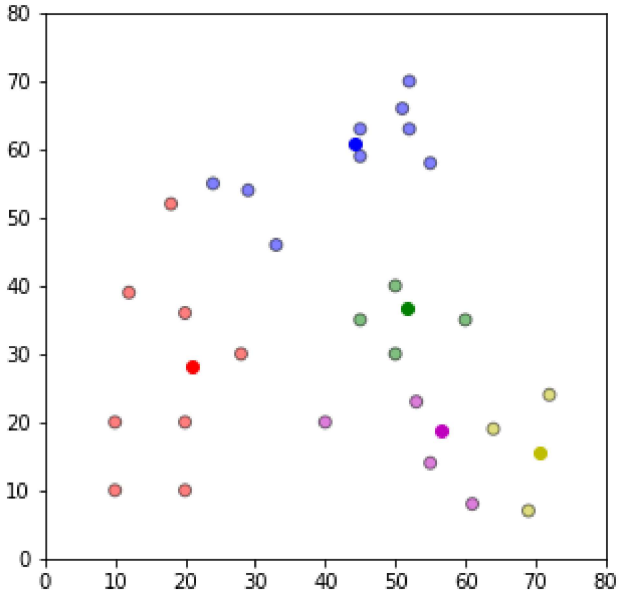
[5 rows x 9 columns]



In [5]:
```
## Update Stage

import copy

old_centroids = copy.deepcopy(centroids)

def update(k):
    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])
    return k

centroids = update(centroids)

fig = plt.figure(figsize=(5, 5))
ax = plt.axes()
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
for i in old_centroids.keys():
    old_x = old_centroids[i][0]
    old_y = old_centroids[i][1]
    dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
    dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
    ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i],
ec=colmap[i])
plt.show()
```



In [6]:
```
## Repeat Assigment Stage

df = assignment(df, centroids)

# Plot results
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

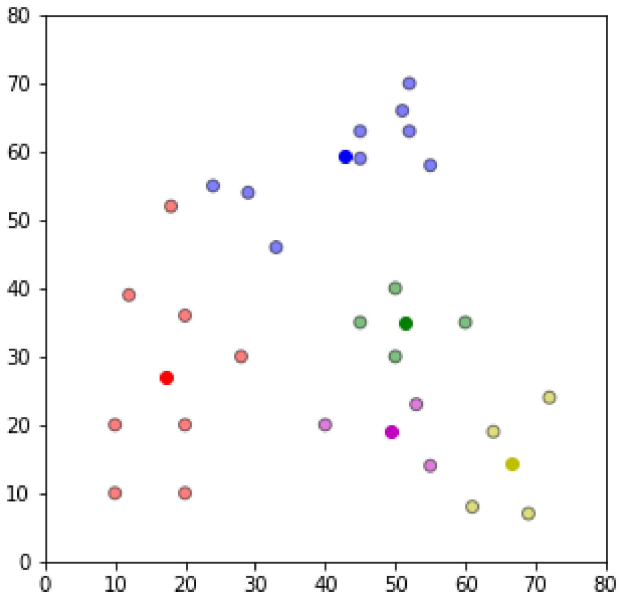


Note that one of the reds is now green and one of the blues is now red.

We are getting closer.

We now repeat until there are no changes to any of the clusters.

In [7]:
```
# Continue until all assigned categories don't change any more
while True:
    closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
    df = assignment(df, centroids)
    if closest_centroids.equals(df['closest']):
        break

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```



So we have 3 clear clusters with 3 means at the centre of these clusters.

We will now repeat the above using scikit-learn, we first fit to our data

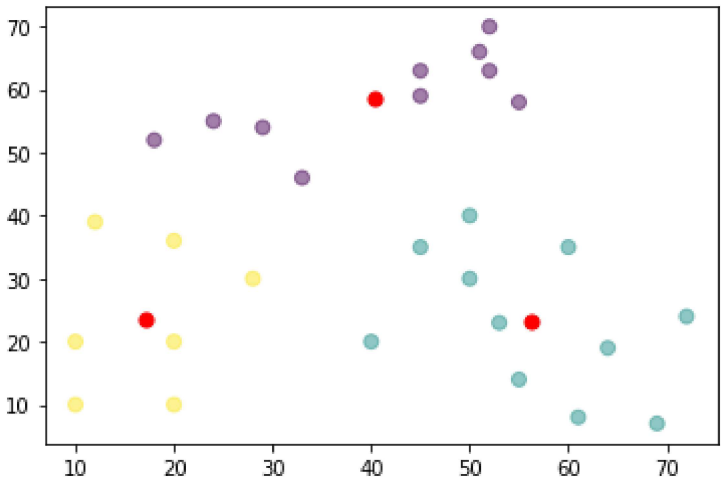
In [0]:
```
from sklearn.cluster import KMeans
```

In [0]:
```
df = pd.DataFrame({
    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64,
69, 72, 50, 40, 45, 50, 60, 10, 10, 20, 20],
    'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7
, 24, 30, 20, 35, 40, 35, 20, 10, 10, 20]
})
```

K-Means Clustering - 3 Clusters

In [0]:
```
kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
```

In [15]:
```
plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5
)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

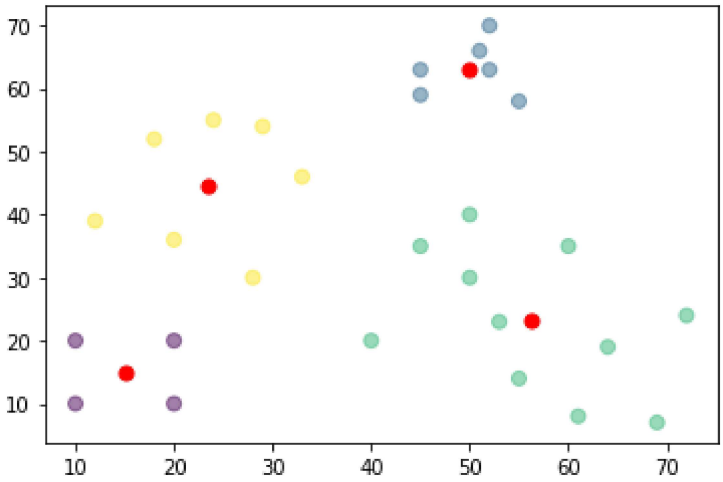


K-Means Clustering – 4 clusters

In [16]:
```
kmeans = KMeans(n_clusters=4).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[15.         15.        ]
 [50.         63.16666667]
 [56.27272727 23.18181818]
 [23.42857143 44.57142857]]
```

In [17]:
```
plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5
)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```



K-Means Clustering – 5 clusters

In [18]:
```
kmeans = KMeans(n_clusters=5).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[49.66666667 30.5       ]
 [23.42857143 44.57142857]
 [64.2        14.4       ]
 [50.         63.16666667]
 [15.         15.        ]]
```

In [19]:
```
plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5
)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```