In [1]:
```python
import csv
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
```

In [2]:
```python
# Load the data from csv file
df = pd.read_csv('shopping-data.csv')
df.head()
```

Out[2]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [3]:
```python
# Rename the columns
df.rename(columns={'Annual Income (k$)':'Annual Income', 'Spending Score (1-100)':'Spending Score'},inplace=True)
df.columns
```

Out[3]:
```
Index(['CustomerID', 'Genre', 'Age', 'Annual Income', 'Spending Score'], dtype='object')
```

In [4]:
```python
# Remove features other than Annual Income and Spending Score
df.drop(['CustomerID'],axis=1,inplace=True)
df.drop(['Age'],axis=1,inplace=True)
df.drop(['Genre'],axis=1,inplace=True)
```

In [5]:
```python
# Check the information and data types of the columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Annual Income   200 non-null    int64
 1   Spending Score  200 non-null    int64
dtypes: int64(2)
memory usage: 3.2 KB
```

In [6]:
```python
type(df)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

Import SSQ statistic functions

In [7]:
```python
import numpy as np
import scipy as sp
import sklearn.cluster
from scipy.spatial.distance import cdist, pdist

import pylab as pl

def ssq_statistics(data, ks=range(1,11), ssq_norm=True):
    """Computes the sum of squares for an nxm dataset.

    The sum of squares (SSQ) is a measure of within-cluster variation that measures the sum of
    squared distances from cluster prototypes.

    Each computation of the SSQ requires the clustering of the input dataset. To identify the
    optimal number of clusters k, the SSQ is computed over a range of possible values of k
    (via the parameter ks). For each value of k, within-cluster dispersion is calculated for the
    input dataset.

    The estimated optimal number of clusters, then, is defined as the value of k prior to an
    "elbow" point in the plot of SSQ values.

    Args:
      data ((n,m) SciPy array): The dataset on which to compute the gap statistics.
      ks (list, optional): The list of values k for which to compute the gap statistics.
        Defaults to range(1,11), which creates a list of values from 1 to 10.
```

```
    Returns:
      ssqs: an array of SSQs, one computed for each k.

    """
    ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)

    #n_samples, n_features = data.shape # the number of rows (samples) and columns (features)
    #if n_samples >= 2500:
    #    # Generate a small sub-sample of the data
    #    data_sample = shuffle(data, random_state=0)[:1000]
    #else:
    #    data_sample = data

    for (i,k) in enumerate(ks): # iterate over the range of k values
        # Fit the model on the data
        kmeans = sklearn.cluster.KMeans(n_clusters=k, random_state=0).fit(data)

        # Predict on the data (k-means) and get labels
        #labels = kmeans.predict(data)

        if ssq_norm:
            dist = np.min(cdist(data, kmeans.cluster_centers_, 'euclidean'), axis=1)

            tot_withinss = sum(dist**2) # Total within-cluster sum of squares
            totss = sum(pdist(data)**2) / data.shape[0] # The total sum of squares
            betweenss = totss - tot_withinss # The between-cluster sum of squares
            ssqs[i] = betweenss/totss*100
        else:
            # The sum of squared error (SSQ) for k
            ssqs[i] = kmeans.inertia_

    return ssqs

def plot_ssq_statistics(ssqs):
    """Generates and shows plots for the sum of squares (SSQ).

    A figure with one plot is generated. The plot is a bar plot of the SSQ computed for each
    value of k.

    Args:
      ssqs (SciPy array): An array of SSQs, one computed for each k.

    """
    # Create a figure
    fig = pl.figure(figsize=(6.75, 4))

    ind = range(1,len(ssqs)+1) # the x values for the ssqs
    width = 0.5 # the width of the bars

    # Create a bar plot
    #rects = pl.bar(ind, ssqs, width)
    pl.plot(ind, ssqs)

    # Add figure labels and ticks
    pl.title('Clustering Sum of Squared Distances', fontsize=16)
    pl.xlabel('Number of clusters k', fontsize=14)
    pl.ylabel('Sum of Squared Distance (SSQ)', fontsize=14)
    pl.xticks(ind)

    # Add text labels
    #for rect in rects:
    #    height = rect.get_height()
    #    pl.text(rect.get_x()+rect.get_width()/2., 1.05*height, '%d' % int(height), \
    #            ha='center', va='bottom')

    # Add figure bounds
    pl.ylim(0, max(ssqs)*1.2)
    pl.xlim(0, len(ssqs)+1.0)

    pl.show()
```

Import gap statistics function

In [8]:

```
import scipy as sp
import scipy as sp
import scipy.cluster.vq
import scipy.spatial.distance
import scipy.stats
import sklearn.cluster

import pylab as pl
```

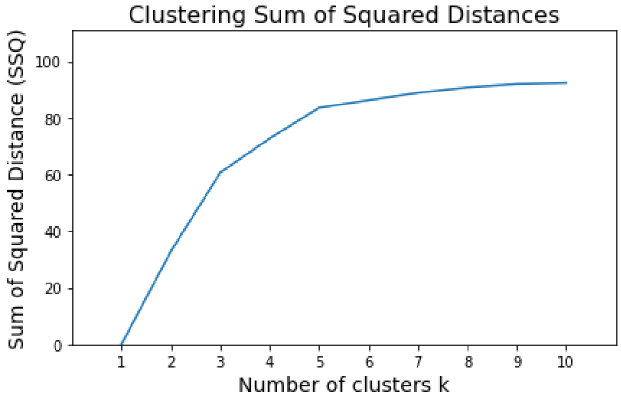
```python
dst = sp.spatial.distance.euclidean

def gap_statistics(data, refs=None, nrefs=20, ks=range(1,11)):
    """Computes the gap statistics for an nxm dataset.

    The gap statistic measures the difference between within-cluster dispersion on an input
    dataset and that expected under an appropriate reference null distribution.

    Computation of the gap statistic, then, requires a series of reference (null) distributions.
    One may either input a precomputed set of reference distributions (via the parameter refs)
    or specify the number of reference distributions (via the parameter nrefs) for automatic
    generation of uniform distributions within the bounding box of the dataset (data).

    Each computation of the gap statistic requires the clustering of the input dataset and of
    several reference distributions. To identify the optimal number of clusters k, the gap
    statistic is computed over a range of possible values of k (via the parameter ks).

    For each value of k, within-cluster dispersion is calculated for the input dataset and each
    reference distribution. The calculation of the within-cluster dispersion for the reference
    distributions will have a degree of variation, which we measure by standard deviation or
    standard error.

    The estimated optimal number of clusters, then, is defined as the smallest value k such that
    gap_k is greater than or equal to the sum of gap_k+1 minus the expected error err_k+1.

    Args:
      data ((n,m) SciPy array): The dataset on which to compute the gap statistics.
      refs ((n,m,k) SciPy array, optional): A precomputed set of reference distributions.
        Defaults to None.
      nrefs (int, optional): The number of reference distributions for automatic generation.
        Defaults to 20.
      ks (list, optional): The list of values k for which to compute the gap statistics.
        Defaults to range(1,11), which creates a list of values from 1 to 10.

    Returns:
      gaps: an array of gap statistics computed for each k.
      errs: an array of standard errors (se), with one corresponding to each gap computation.
      difs: an array of differences between each gap_k and the sum of gap_k+1 minus err_k+1.

    """
    shape = data.shape

    if refs==None:
        tops = data.max(axis=0) # maxima along the first axis (rows)
        bots = data.min(axis=0) # minima along the first axis (rows)
        dists = sp.matrix(sp.diag(tops-bots)) # the bounding box of the input dataset

        # Generate nrefs uniform distributions each in the half-open interval [0.0, 1.0)
        rands = sp.random.random_sample(size=(shape[0],shape[1], nrefs))

        # Adjust each of the uniform distributions to the bounding box of the input dataset
        for i in range(nrefs):
            rands[:,:,i] = rands[:,:,i]*dists+bots
    else:
        rands = refs

    gaps = sp.zeros((len(ks),))   # array for gap statistics (lenth ks)
    errs = sp.zeros((len(ks),))   # array for model standard errors (length ks)
    difs = sp.zeros((len(ks)-1,)) # array for differences between gaps (length ks-1)

    for (i,k) in enumerate(ks): # iterate over the range of k values
        # Cluster the input dataset via k-means clustering using the current value of k
        try:
            (kmc,kml) = sp.cluster.vq.kmeans2(data, k)
        except np.linalg.LinAlgError:
            kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(data)
            (kmc, kml) = kmeans.cluster_centers_, kmeans.labels_

        # Generate within-dispersion measure for the clustering of the input dataset
        disp = sum([dst(data[m,:],kmc[kml[m],:]) for m in range(shape[0])])

        # Generate within-dispersion measures for the clusterings of the reference datasets
        refdisps = sp.zeros((rands.shape[2],))
        for j in range(rands.shape[2]):
            # Cluster the reference dataset via k-means clustering using the current value of k
            try:
                (kmc,kml) = sp.cluster.vq.kmeans2(rands[:,:,j], k)
            except np.linalg.LinAlgError:
                kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(rands[:,:,j])
                (kmc, kml) = kmeans.cluster_centers_, kmeans.labels_

            refdisps[j] = sum([dst(rands[m,:,j],kmc[kml[m],:]) for m in range(shape[0])])
```

```python
        # Compute the (estimated) gap statistic for k
        gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))

        # Compute the expected error for k
        errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
                              for refdisp in refdisps)/float(nrefs)) * sp.sqrt(1+1/nrefs)

    # Compute the difference between gap_k and the sum of gap_k+1 minus err_k+1
    difs = sp.array([gaps[k] - (gaps[k+1]-errs[k+1]) for k in range(len(gaps)-1)])

    #print "Gaps: " + str(gaps)
    #print "Errs: " + str(errs)
    #print "Difs: " + str(difs)

    return gaps, errs, difs

def plot_gap_statistics(gaps, errs, difs):
    """Generates and shows plots for the gap statistics.

    A figure with two subplots is generated. The first subplot is an errorbar plot of the
    estimated gap statistics computed for each value of k. The second subplot is a barplot
    of the differences in the computed gap statistics.

    Args:
      gaps (SciPy array): An array of gap statistics, one computed for each k.
      errs (SciPy array): An array of standard errors (se), with one corresponding to each gap
        computation.
      difs (SciPy array): An array of differences between each gap_k and the sum of gap_k+1
        minus err_k+1.

    """
    # Create a figure
    fig = pl.figure(figsize=(16, 4))

    pl.subplots_adjust(wspace=0.35) # adjust the distance between figures

    # Subplot 1
    ax = fig.add_subplot(121)
    ind = range(1,len(gaps)+1) # the x values for the gaps

    # Create an errorbar plot
    rects = ax.errorbar(ind, gaps, yerr=errs, xerr=None, linewidth=1.0)

    # Add figure labels and ticks
    ax.set_title('Clustering Gap Statistics', fontsize=16)
    ax.set_xlabel('Number of clusters k', fontsize=14)
    ax.set_ylabel('Gap Statistic', fontsize=14)
    ax.set_xticks(ind)

    # Add figure bounds
    ax.set_ylim(0, max(gaps+errs)*1.1)
    ax.set_xlim(0, len(gaps)+1.0)

    # Subplot 2
    ax = fig.add_subplot(122)
    ind = range(1,len(difs)+1) # the x values for the difs

    max_gap = None
    if len(np.where(difs > 0)[0]) > 0:
        max_gap = np.where(difs > 0)[0][0] + 1 # the k with the first positive dif

    # Create a bar plot
    ax.bar(ind, difs, alpha=0.5, color='g', align='center')

    # Add figure labels and ticks
    if max_gap:
        ax.set_title('Clustering Gap Differences\n(k=%d Estimated as Optimal)' % (max_gap), \
                     fontsize=16)
    else:
        ax.set_title('Clustering Gap Differences\n', fontsize=16)
    ax.set_xlabel('Number of clusters k', fontsize=14)
    ax.set_ylabel('Gap Difference', fontsize=14)
    ax.xaxis.set_ticks(range(1,len(difs)+1))

    # Add figure bounds
    ax.set_ylim(min(difs)*1.2, max(difs)*1.2)
    ax.set_xlim(0, len(difs)+1.0)

    # Show the figure
    pl.show()
```

Generate and plot the SSQ statistics

In [9]:
```
ssqs = ssq_statistics(df, ks=range(1,11), ssq_norm=True)
plot_ssq_statistics(ssqs)
```

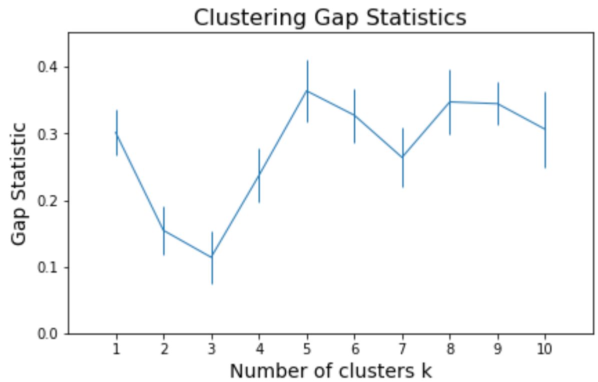
```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```

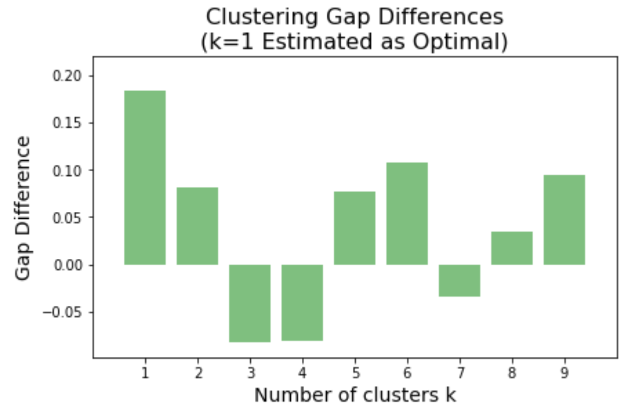


Generate and plot the gap statistics

In [10]:
```
gaps, errs, difs = gap_statistics(df.to_numpy(dtype=float), refs=None, nrefs=20, ks=range(1,11))
plot_gap_statistics(gaps, errs, difs)
```

```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:55: DeprecationWarning: scipy.diag i
s deprecated and will be removed in SciPy 2.0.0, use numpy.diag instead
  dists = sp.matrix(sp.diag(tops-bots)) # the bounding box of the input dataset
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:66: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  gaps = sp.zeros((len(ks),))   # array for gap statistics (lenth ks)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:67: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  errs = sp.zeros((len(ks),))   # array for model standard errors (length ks)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:68: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  difs = sp.zeros((len(ks)-1,)) # array for differences between gaps (length ks-1)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:82: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  refdisps = sp.zeros((rands.shape[2],))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:94: DeprecationWarning: scipy.log is
deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.log instead
  gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:94: DeprecationWarning: scipy.mean i
s deprecated and will be removed in SciPy 2.0.0, use numpy.mean instead
  gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.log is
deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.log instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.mean i
s deprecated and will be removed in SciPy 2.0.0, use numpy.mean instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.sqrt i
s deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:98: DeprecationWarning: scipy.sqrt i
s deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
  for refdisp in refdisps)/float(nrefs)) * sp.sqrt(1+1/nrefs)
/opt/homebrew/lib/python3.9/site-packages/scipy/cluster/vq.py:607: UserWarning: One of the clusters is empty. Re-ru
n kmeans with a different initialization.
  warnings.warn("One of the clusters is empty. "
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:101: DeprecationWarning: scipy.array
is deprecated and will be removed in SciPy 2.0.0, use numpy.array instead
  difs = sp.array([gaps[k] - (gaps[k+1]-errs[k+1]) for k in range(len(gaps)-1)])
```





Scatter plot of the data in 2d showing the clusters in different colors. Also show the cluster centers in the plot.

In [11]:
```python
# We use k=6 to plot the data in 2d
n = 6
kmeans6 = KMeans(n_clusters=n,n_init=10,max_iter=500)
kmeans6.fit(df)
```

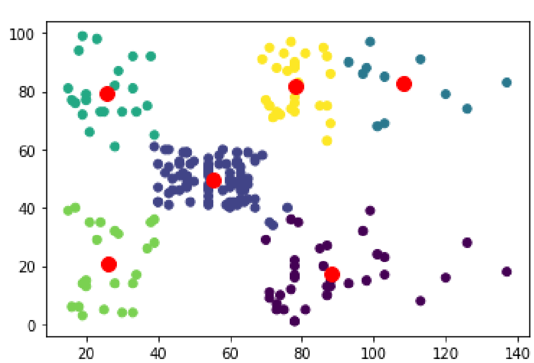
Out[11]:
```
▾                KMeans
KMeans(max_iter=500, n_clusters=6)
```

In [12]:
```python
df['clusters']=kmeans6.labels_
kmeans6.cluster_centers_
```

Out[12]:
```
array([[ 26.30434783,  20.91304348],
       [108.18181818,  82.72727273],
       [ 88.2       ,  17.11428571],
       [ 55.2962963 ,  49.51851852],
       [ 25.72727273,  79.36363636],
       [ 78.03571429,  81.89285714]])
```

In [13]:
```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# According to SSQ plot, the elbow point is 5, so set k=5.
# Reference: https://wellsr.com/python/python-kmeans-clustering-with-scikit-learn/
k=6
data_pred = KMeans(k, random_state=0).fit_predict(df)
plt.scatter(df["Annual Income"], df["Spending Score"], c=data_pred);

model = KMeans(n_clusters = 6)
model.fit(df)
plt.scatter(model.cluster_centers_[:, 0], model.cluster_centers_[:, 1], s=100, c='red')
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x13e346fa0>
```

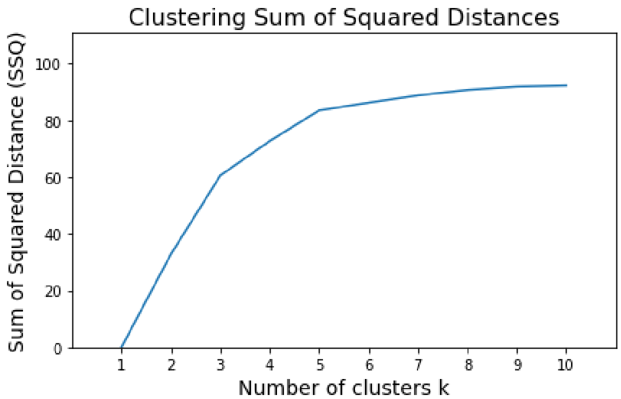


1. Where did you estimate the elbow point to be (between what values of k)? What value of k was typically estimated as optimal by the gap statistic? To adequately answer this question, consider generating both measures several (atleast 5) times, as there may be some amount of variation in the value of k that they each estimate as optimal.

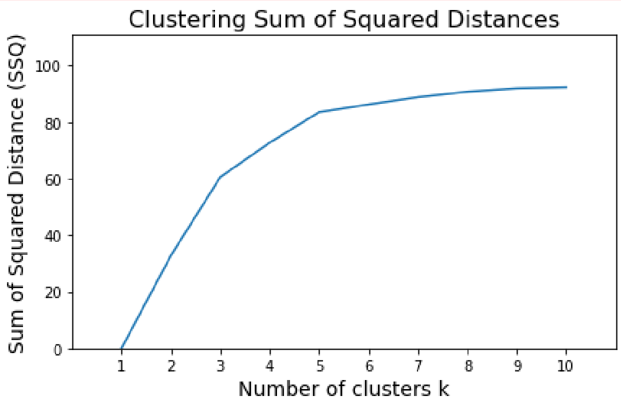
Using SSQ method 5 times

In [14]:
```
for i in range(5):
    ssqs = ssq_statistics(df, ks=range(1,11), ssq_norm=True)
    plot_ssq_statistics(ssqs)
```

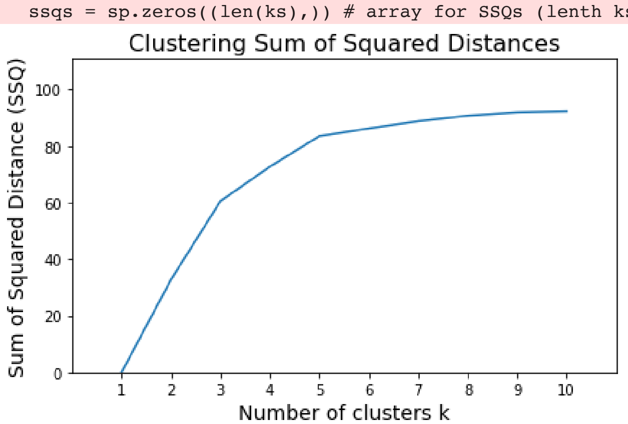
```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```



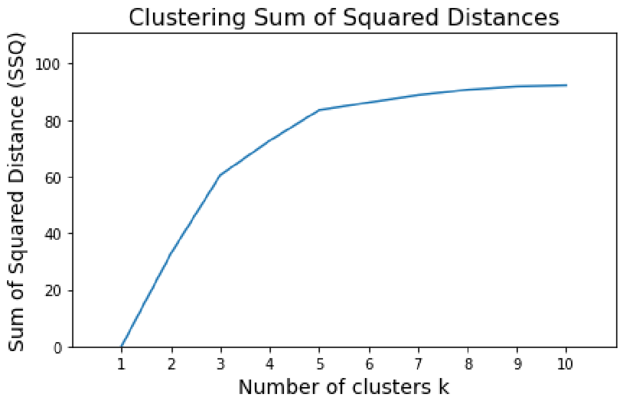

```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```



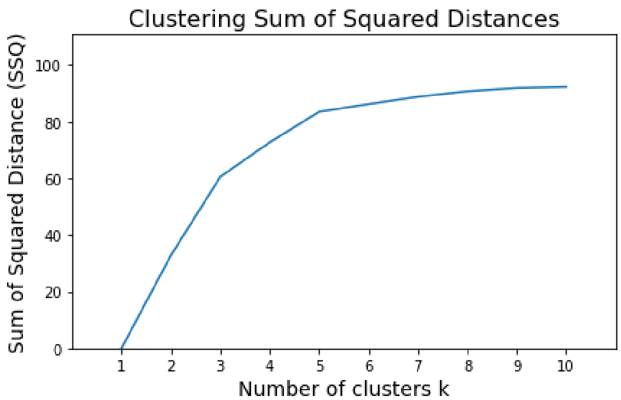

```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```




```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```




```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/231730143.py:31: DeprecationWarning: scipy.zeros i
s deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  ssqs = sp.zeros((len(ks),)) # array for SSQs (lenth ks)
```



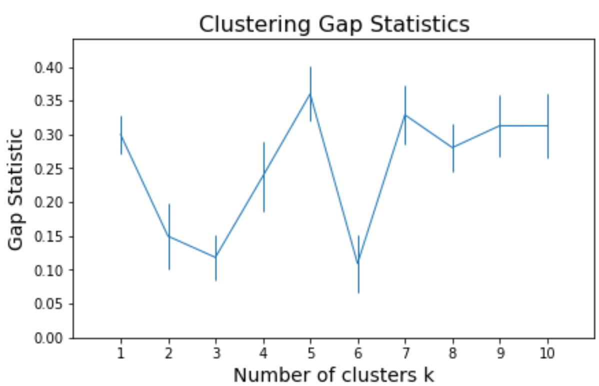
Note:

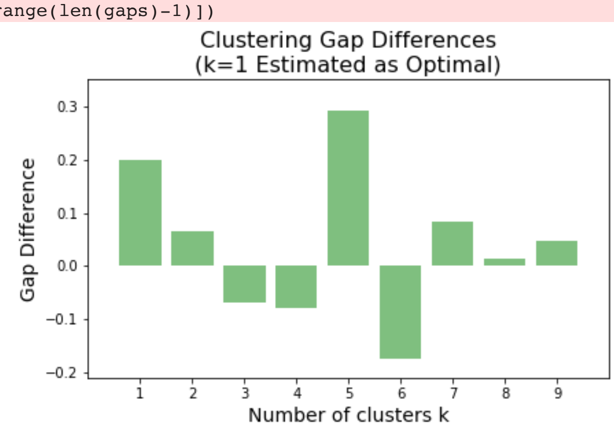
1. The sum of the squared deviation increases as the number of cluster increases.
2. The elbow point is the point in the graph when we notice a bend in the curve.
3. We notice 2 potential elbow points or "bends" i.e. one at approximately 3 and another at around 5.

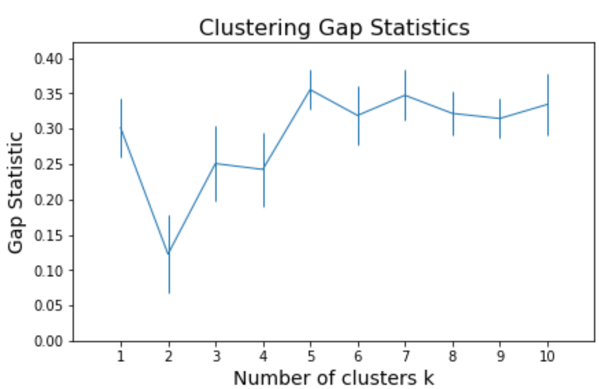
Using Gap statistics method 5 times

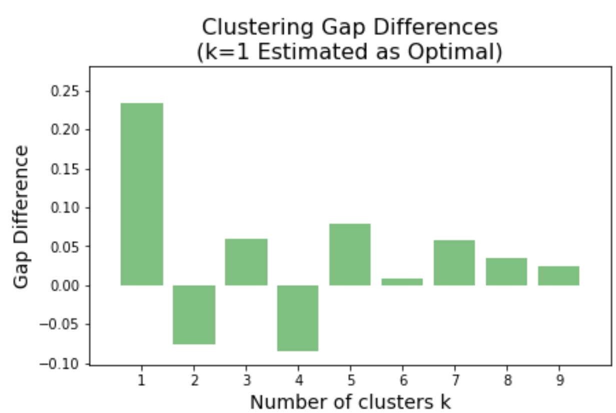
In [15]:
```
for i in range(5):
    gaps, errs, difs = gap_statistics(df.to_numpy(dtype=float), refs=None, nrefs=20, ks=range(1,11))
    plot_gap_statistics(gaps, errs, difs)
```

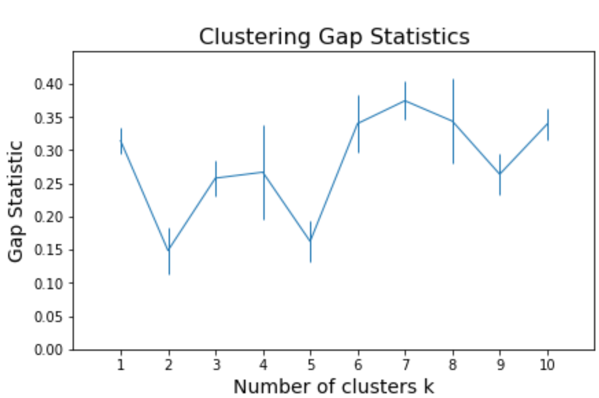
```
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:55: DeprecationWarning: scipy.diag i
s deprecated and will be removed in SciPy 2.0.0, use numpy.diag instead
  dists = sp.matrix(sp.diag(tops-bots)) # the bounding box of the input dataset
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:66: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  gaps = sp.zeros((len(ks),))   # array for gap statistics (lenth ks)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:67: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  errs = sp.zeros((len(ks),))   # array for model standard errors (length ks)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:68: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  difs = sp.zeros((len(ks)-1,)) # array for differences between gaps (length ks-1)
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:82: DeprecationWarning: scipy.zeros
is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  refdisps = sp.zeros((rands.shape[2],))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:94: DeprecationWarning: scipy.log is
deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.log instead
  gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:94: DeprecationWarning: scipy.mean i
s deprecated and will be removed in SciPy 2.0.0, use numpy.mean instead
  gaps[i] = sp.mean(sp.log(refdisps) - sp.log(disp))
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.log is
deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.log instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.mean i
s deprecated and will be removed in SciPy 2.0.0, use numpy.mean instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:97: DeprecationWarning: scipy.sqrt i
s deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
  errs[i] = sp.sqrt(sum(((sp.log(refdisp)-sp.mean(sp.log(refdisps)))**2) \
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:98: DeprecationWarning: scipy.sqrt i
s deprecated and will be removed in SciPy 2.0.0, use numpy.lib.scimath.sqrt instead
  for refdisp in refdisps)/float(nrefs)) * sp.sqrt(1+1/nrefs)
/opt/homebrew/lib/python3.9/site-packages/scipy/cluster/vq.py:607: UserWarning: One of the clusters is empty. Re-ru
n kmeans with a different initialization.
  warnings.warn("One of the clusters is empty. "
/var/folders/px/yn0kfz7s0_1fsvnqwr28n13w0000gn/T/ipykernel_13135/1660669241.py:101: DeprecationWarning: scipy.array
is deprecated and will be removed in SciPy 2.0.0, use numpy.array instead
  difs = sp.array([gaps[k] - (gaps[k+1]-errs[k+1]) for k in range(len(gaps)-1)])
```

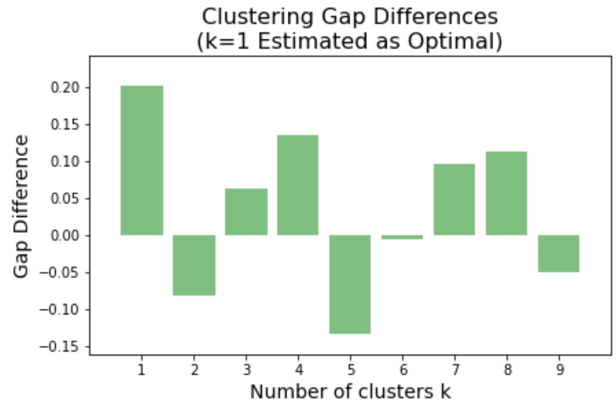


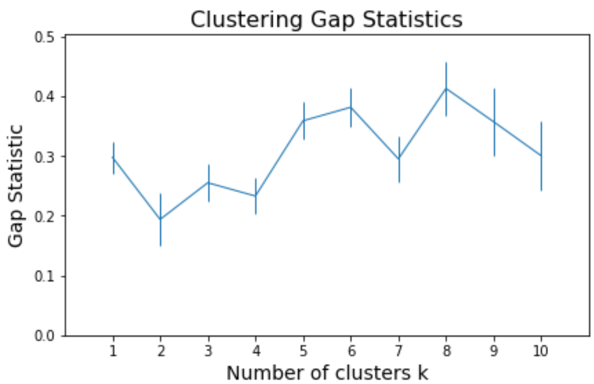


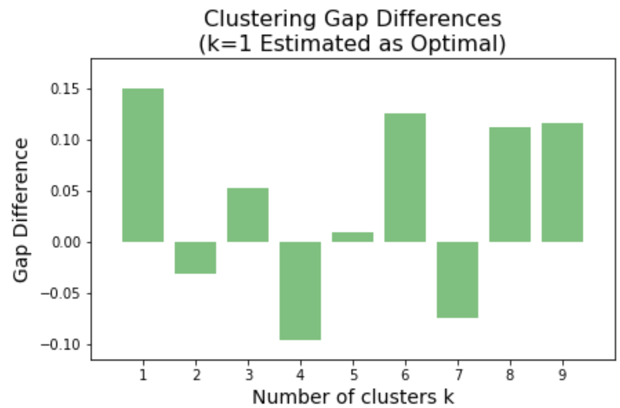


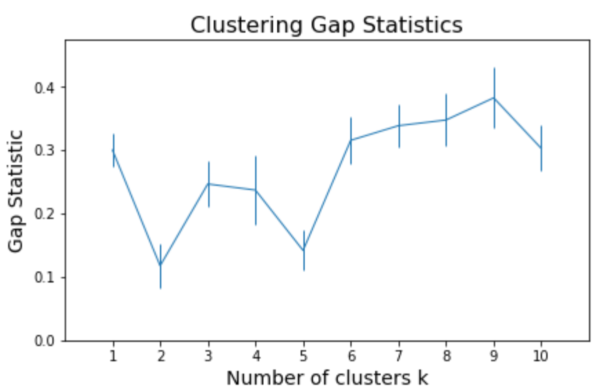


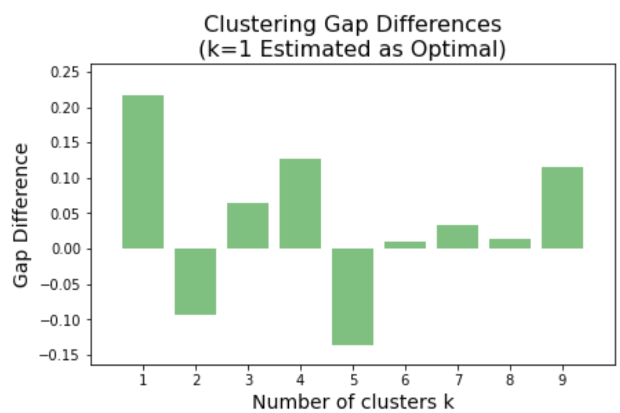












Note:

1. The lowest value of k with a corresponding gap statistic higher than or equal to the gap statistic of k+1 is the estimated optimal value of k.
2. Although there are some subtle variations among the gap difference/statiscs we generated above, k=1 always gives the optimal value.

2. Based on the scatter plot of the clustered data, what makes most sense? Give logical interpretation from visually inspecting the clusters.

Since we noticed 2 elbow points, we scatter plot of the clustered data and visualize the clusters.

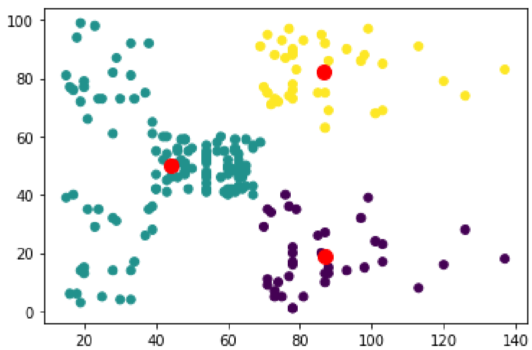
3-cluster

In [16]:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# According to SSQ plot, the elbow point is 5, so set k=5.
# Reference: https://wellsr.com/python/python-kmeans-clustering-with-scikit-learn/
k=3
data_pred = KMeans(k, random_state=0).fit_predict(df)
plt.scatter(df["Annual Income"], df["Spending Score"], c=data_pred);
```

```python
model = KMeans(n_clusters = 3)
model.fit(df)
plt.scatter(model.cluster_centers_[:, 0], model.cluster_centers_[:, 1], s=100, c='red')
```

Out[16]: <matplotlib.collections.PathCollection at 0x13e3b1a60>

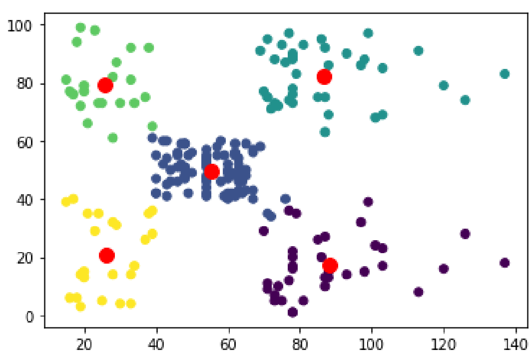


5 cluster

In [17]:

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# According to SSQ plot, the elbow point is 5, so set k=5.
# Reference: https://wellsr.com/python/python-kmeans-clustering-with-scikit-learn/
k=5
data_pred = KMeans(k, random_state=0).fit_predict(df)
plt.scatter(df["Annual Income"], df["Spending Score"], c=data_pred);

model = KMeans(n_clusters = 5)
model.fit(df)
plt.scatter(model.cluster_centers_[:, 0], model.cluster_centers_[:, 1], s=100, c='red')
```

Out[17]: <matplotlib.collections.PathCollection at 0x13e41acd0>



Logic interpretation from visualization

1. The points in the same cluster are scatterd in the same color. They are closer to the average point of that cluster , and they are visually shown together. Data points in the same groups are more similar to other data points in the same group than those in other groups.
2. When k=3, we see a clear line that sepreate one cluster and the other two where annual income is around 68.
3. When k=5, we have 5 clusters in top-right corner, bottom-right, top-left, bottom-left, and center.
4. Visually, it makes sense that we divide the data into clusters based on the spending score and annual income.

3. Between SSQ and Gap Statistics, does one measure seem to be a consistently better criterion for choosing the value of k than the other? Why or why not?

According to the visualization plotted above:

1. SSQ Statistics generated clusters make more sense, which can divide data into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. However, from Gap statistics, k=1 doesn't tell anything about clustering.
2. Therefore, SSQ seems to be a consistently better criterion for chossing the value of k.
3. Clustering depends on scale, among other things.

4. According to the clusGap help, the default method "firstSEmax" looks for the smallest k such that its value $f(k)$ is not more than 1 standard error away from the first local maximum. This criterion does not cause any of the gap statistics to stand out, resulting in an estimate of k=1.
5. Clustering result may involve various factors, including time complexity, effeciency, accuracy, simplicity, etc.

References:


```
1. https://stats.stackexchange.com/questions/140711/why-does-gap-statistic-for-k-means-suggest-one-cluster-even-though-there-are-ob
```