In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import vstack,array
from numpy.random import rand
from scipy.cluster.vq import kmeans,vq
from math import sqrt
from sklearn.cluster import KMeans
from pylab import plot, show
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [2]:
```python
stocks = pd.read_csv('data_stocks.csv')
```

In [3]:
```python
stocks.head()
```

Out[3]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDA(
0	1491226200	2363.6101	42.3300	143.6800	129.6300	82.040	10:
1	1491226260	2364.1001	42.3600	143.7000	130.3200	82.080	10:
2	1491226320	2362.6799	42.3100	143.6901	130.2250	82.030	10:
3	1491226380	2364.3101	42.3700	143.6400	130.0729	82.000	10:
4	1491226440	2364.8501	42.5378	143.6600	129.8800	82.035	10:

5 rows × 502 columns

In [4]:
```python
stocks.describe()
```

Out[4]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI
count	4.126600e+04	41266.000000	41266.000000	41266.000000	41266.00000	41266.000000
mean	1.497749e+09	2421.537882	47.708346	150.453566	141.31793	79.446873
std	3.822211e+06	39.557135	3.259377	6.236826	6.91674	2.000283
min	1.491226e+09	2329.139900	40.830000	140.160000	128.24000	74.800000
25%	1.494432e+09	2390.860100	44.945400	144.640000	135.19500	78.030000
50%	1.497638e+09	2430.149900	48.360000	149.945000	142.26000	79.410000
75%	1.501090e+09	2448.820100	50.180000	155.065000	147.10000	80.580000
max	1.504210e+09	2490.649900	54.475000	164.510000	155.33000	90.440000

8 rows × 502 columns

In [5]:
```
stocks.shape
```
Out[5]:
```
(41266, 502)
```

In [6]:
```
stocks.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41266 entries, 0 to 41265
Columns: 502 entries, DATE to NYSE.ZTS
dtypes: float64(501), int64(1)
memory usage: 158.0 MB
```

In [7]:
```
# creating deep copy of data frame
data_cor = stocks.copy()
data_cor.drop(['DATE','SP500'],inplace=True,axis=1)
```

Problem 1

There are various stocks for which we have collected a data set, which all stocks are apparently similar in performance

In [8]:
```
# finding correlation between variables - identify highly correlated variables
(stocks)
cor = data_cor.corr()
```

In [9]:
```
cor_dt = pd.DataFrame(data = cor.values, columns = cor.index, index = cor.inde
x)
cor_dt.head()
```

Out[9]:

	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP	NA
NASDAQ.AAL	1.000000	0.082065	0.542213	0.209446	0.245801	
NASDAQ.AAPL	0.082065	1.000000	0.714578	0.264269	0.265641	
NASDAQ.ADBE	0.542213	0.714578	1.000000	0.259282	0.476496	
NASDAQ.ADI	0.209446	0.264269	0.259282	1.000000	-0.085074	
NASDAQ.ADP	0.245801	0.265641	0.476496	-0.085074	1.000000	

5 rows × 500 columns

In [10]:
```
# Stocks that are similar in performance
cor_dt[cor_dt['NASDAQ.AAL'].values > 0.8].index
```

Out[10]:
```
Index(['NASDAQ.AAL', 'NASDAQ.AMZN', 'NASDAQ.EXPE', 'NASDAQ.HAS', 'NYSE.CMI',
       'NYSE.COH', 'NYSE.CSRA', 'NYSE.DAL', 'NYSE.DE', 'NYSE.FBHS', 'NYSE.GL
W',
       'NYSE.IT', 'NYSE.ITW', 'NYSE.IVZ', 'NYSE.MCK', 'NYSE.NWL', 'NYSE.PKI',
       'NYSE.ROP', 'NYSE.SWK', 'NYSE.TEL', 'NYSE.WRK', 'NYSE.XL'],
      dtype='object')
```

Problem 2

How many Unique patterns that exist in the historical stock data set, based on fluctuations in price.

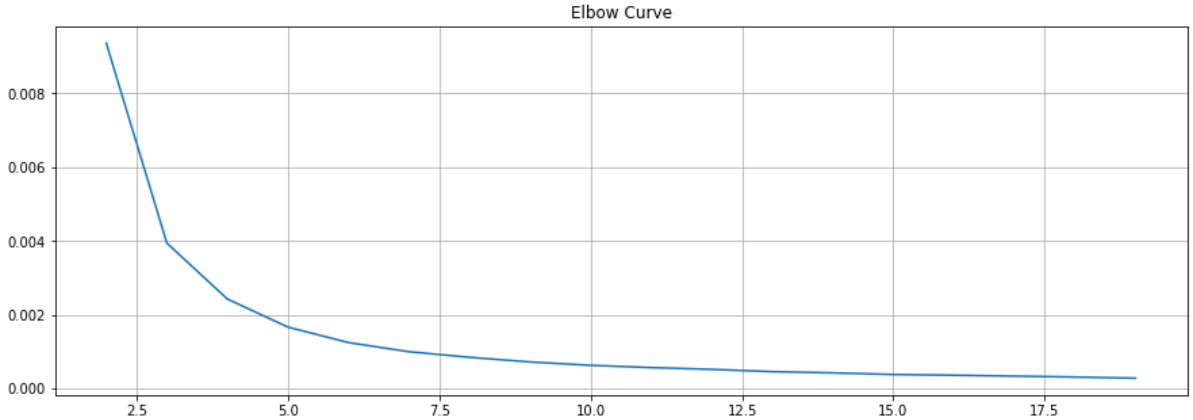
In [11]:
```
returns = stocks.pct_change().mean()*252
returns = pd.DataFrame(returns, columns=['Returns'])
# returns.columns = ['Returns']
returns['Volatility'] = stocks.pct_change().std()*sqrt(252)
```

Format the data into numpy array to feed into K-means algorithm

In [12]:
```
data_kmeans = np.asarray([np.asarray(returns.Returns), np.asarray(returns.Vola
tility)]).T
```

In [13]:
```
X = data_kmeans
distortions=[]
for i in range(2, 20):
    k_means = KMeans(n_clusters=i)
    k_means.fit(X)
    distortions.append(k_means.inertia_)
```

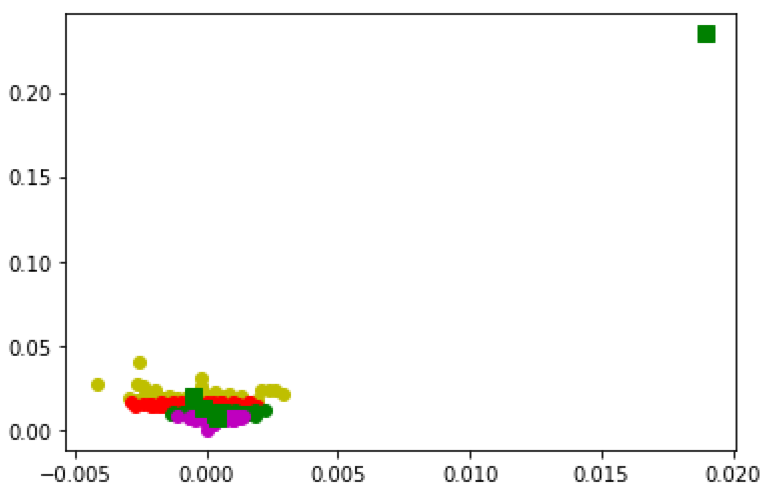
In [14]:
```
fig = plt.figure(figsize=(15, 5))
plt.plot(range(2, 20), distortions)
plt.grid(True)
plt.title('Elbow Curve')
plt.show()
```



From the above Elbow curve we see that the curve has a steep at cluster no. 5.

In [15]:
```
# Computing k-means with 5 clusters
centroids, _ = kmeans(data_kmeans, 5)
idx, _ = vq(data_kmeans, centroids)
```

In [16]:
```
# Some plotting using numpy's logical indexing
plot(data_kmeans[idx==0,0], data_kmeans[idx==0,1], 'ob',
     data_kmeans[idx==1,0], data_kmeans[idx==1,1], 'oy',
     data_kmeans[idx==2,0], data_kmeans[idx==2,1], 'or',
     data_kmeans[idx==3,0], data_kmeans[idx==3,1], 'og',
     data_kmeans[idx==4,0], data_kmeans[idx==4,1], 'om')
plot(centroids[:,0], centroids[:,1], 'sg', markersize=8)
show()
```



Ok, so it looks like we have an outlier in the data which is skewing the results and making it difficult to actually see what is going on for all the other stocks. Let’s take the easy route and just delete the outlier from our data set and run this again.

In [17]:
```
#identify the outlier
print(returns.idxmax())
```

```
Returns       NYSE.XRX
Volatility    NYSE.XRX
dtype: object
```

Ok so let’s drop the stock ‘NYSE.XRX and recreate the necessary data arrays.

In [18]:
```
# Drop the outlier stock from our data
stocks.drop(['NYSE.XRX', 'DATE', 'SP500'], inplace=True, axis=1)
stocks.head()
```

Out[18]:

	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP	NASDAQ.ADSK
0	42.3300	143.6800	129.6300	82.040	102.2300	85.2200
1	42.3600	143.7000	130.3200	82.080	102.1400	85.6500
2	42.3100	143.6901	130.2250	82.030	102.2125	85.5100
3	42.3700	143.6400	130.0729	82.000	102.1400	85.4872
4	42.5378	143.6600	129.8800	82.035	102.0600	85.7001

5 rows × 499 columns

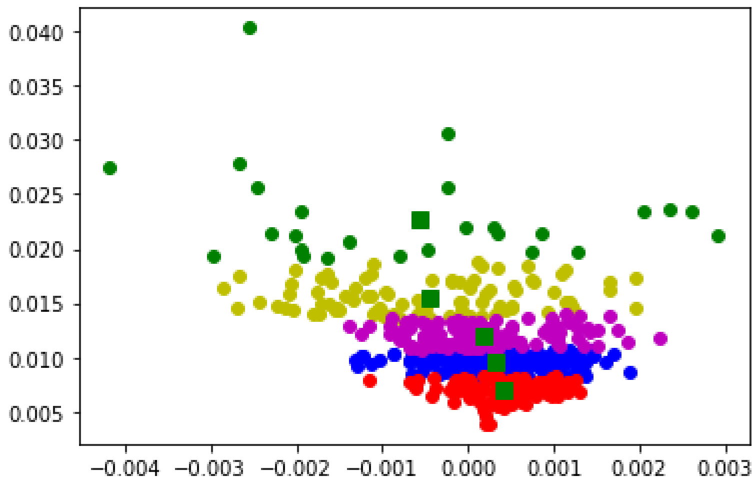
In [19]:
```
returns = stocks.pct_change().mean()*252
returns = pd.DataFrame(returns, columns=['Returns'])
returns['Volatility'] = stocks.pct_change().std()*sqrt(252)
```

Format the data into numpy array to feed into K-means algorithm

In [20]:
```
data_kmeans = np.asarray([np.asarray(returns.Returns), np.asarray(returns.Vola
tility)]).T
```

In [21]:
```
# Computing k-means with 5 clusters
centroids, _ = kmeans(data_kmeans, 5)
idx, _ = vq(data_kmeans, centroids)
```

In [22]:
```
# Some plotting using numpy's logical indexing
plot(data_kmeans[idx==0,0], data_kmeans[idx==0,1], 'ob',
     data_kmeans[idx==1,0], data_kmeans[idx==1,1], 'oy',
     data_kmeans[idx==2,0], data_kmeans[idx==2,1], 'or',
     data_kmeans[idx==3,0], data_kmeans[idx==3,1], 'og',
     data_kmeans[idx==4,0], data_kmeans[idx==4,1], 'om')
plot(centroids[:,0], centroids[:,1], 'sg', markersize=8)
show()
```



Finally to get the details of which stock is actually in which cluster we can run the following line of code to carry out a list comprehension to create a list of tuples in the (Stock Name, Cluster Number) format:

In [23]:
```
details = [(name,cluster) for name, cluster in zip(returns.index,idx)]
details[:5]
```

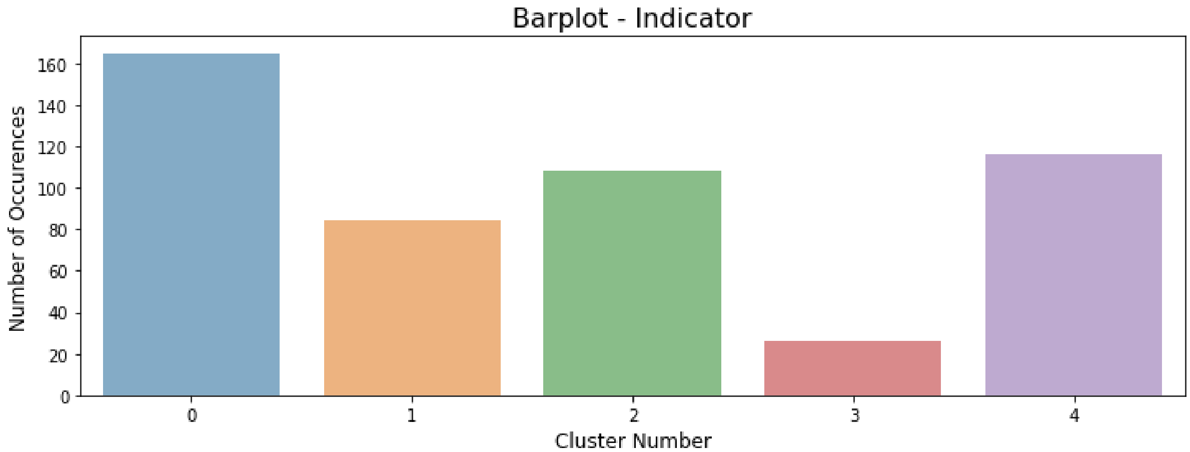
Out[23]:
```
[('NASDAQ.AAL', 1),
 ('NASDAQ.AAPL', 0),
 ('NASDAQ.ADBE', 0),
 ('NASDAQ.ADI', 4),
 ('NASDAQ.ADP', 4)]
```

So there you have it, we now have a list of each of the stocks in the S&P 500, along with which one of 5 clusters they belong to with the clusters being defined by their return and volatility characteristics. We also have a visual representation of the clusters in chart format.

In [24]:
```
df = pd.DataFrame(details, columns=['Stock_Name','Cluster_No'])
df.Cluster_No.value_counts()
```

Out[24]:
```
0    165
4    116
2    108
1     84
3     26
Name: Cluster_No, dtype: int64
```

In [25]:
```
ind = df.Cluster_No.value_counts()
plt.figure(figsize=(12,4))
sns.barplot(x=ind.index, y=ind.values, alpha=0.6)
plt.ylabel('Number of Occurences',fontsize=12)
plt.xlabel('Cluster Number',fontsize=12)
plt.title('Barplot - Indicator',fontsize =16)
plt.show()
```



Problem 3

Identify which all stocks are moving together and which all stocks are different from each other.

In [26]:
```
stocks = pd.read_csv('data_stocks.csv')
stocks.head()
```

Out[26]:

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAC
0	1491226200	2363.6101	42.3300	143.6800	129.6300	82.040	10:
1	1491226260	2364.1001	42.3600	143.7000	130.3200	82.080	10:
2	1491226320	2362.6799	42.3100	143.6901	130.2250	82.030	10:
3	1491226380	2364.3101	42.3700	143.6400	130.0729	82.000	10:
4	1491226440	2364.8501	42.5378	143.6600	129.8800	82.035	10:

5 rows × 502 columns

In [27]:
```
data_pca = stocks.copy()
data_pca.drop(['DATE', 'SP500'], inplace=True, axis=1)
```

In [28]:
```
# Convert data into numpy array
X = data_pca.values
```

In [29]:
```
# Scaling the values
sc = StandardScaler()
X_scaled = sc.fit_transform(X)
```

In [30]:
```
pca = PCA(n_components=150)
pca.fit(X_scaled)
```

Out[30]:
```
PCA(copy=True, iterated_power='auto', n_components=150, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

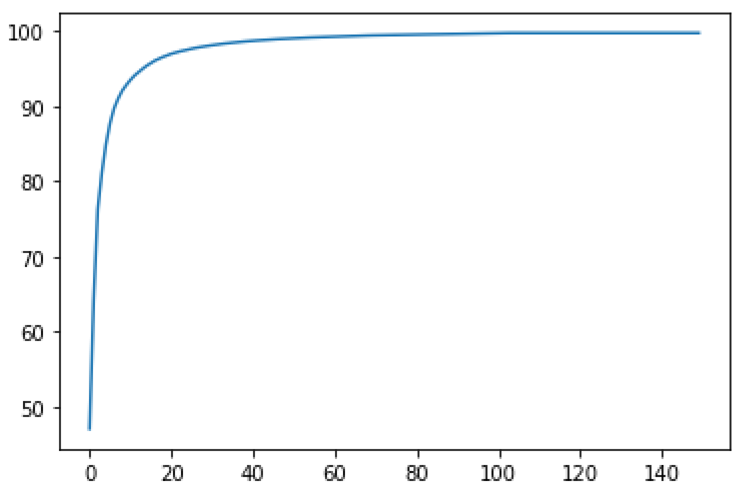
In [31]:
```
# The amount of variance that each PCA explains
var = pca.explained_variance_ratio_

# cumulative varince explains
var1 = np.cumsum(np.round(var, decimals=4)*100)
print(var1)
```

```
[47.03 64.26 76.28 81.13 84.93 87.74 89.74 91.05 92.1  92.85 93.53 94.11
 94.57 95.02 95.43 95.78 96.11 96.37 96.6  96.82 97.01 97.17 97.32 97.45
 97.58 97.7  97.81 97.91 98.   98.08 98.16 98.24 98.31 98.38 98.44 98.5
 98.56 98.61 98.66 98.71 98.75 98.79 98.83 98.87 98.91 98.94 98.97 99.
 99.03 99.06 99.09 99.12 99.15 99.17 99.19 99.21 99.23 99.25 99.27 99.29
 99.31 99.33 99.35 99.37 99.39 99.41 99.43 99.45 99.46 99.47 99.48 99.49
 99.5  99.51 99.52 99.53 99.54 99.55 99.56 99.57 99.58 99.59 99.6  99.61
 99.62 99.63 99.64 99.65 99.66 99.67 99.68 99.69 99.7  99.71 99.72 99.73
 99.74 99.75 99.76 99.77 99.78 99.79 99.79 99.79 99.79 99.79 99.79 99.79
 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79
 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79
 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79 99.79
 99.79 99.79 99.79 99.79 99.79 99.79]
```

In [32]:
```
plt.plot(var1)
```

Out[32]: [<matplotlib.lines.Line2D at 0x21980050448>]



In [33]:
```
# Looking at above plot I can consider 25 variables
pca = PCA(n_components = 25)
X1 = pca.fit_transform(X_scaled)
print(X1)
```

```
[[ 25.55018064  10.00580482  -9.38207446 ...   0.54989216   0.09822323
    2.41815712]
 [ 25.64880185   9.89282687  -9.8023104  ...   0.45251862   0.18287394
    2.31119141]
 [ 25.56345929   9.82533675  -9.67570287 ...   0.52932995   0.05562211
    2.05371665]
 ...
 [-22.76894921  13.32753802   6.56220278 ...  -2.15019881   1.19339642
   -0.31219387]
 [-22.61319638  13.41831515   6.6755356  ...  -2.13605041   1.19947637
   -0.33862978]
 [-22.72127837  13.36292841   6.60406294 ...  -2.17308504   1.17922968
   -0.29849097]]
```

In [34]:
```
print('Number of PCA:: ', len(pca.components_))
print(abs(pca.components_))
```

```
Number of PCA::  25
[[0.03925756 0.04106421 0.0629084  ... 0.06247664 0.00253829 0.05169773]
 [0.06428354 0.033861   0.00186129 ... 0.02040637 0.08122924 0.05950068]
 [0.03985758 0.06416494 0.01207933 ... 0.02101011 0.06637293 0.02356977]
 ...
 [0.0115967  0.00535958 0.00471095 ... 0.01175626 0.04450505 0.0119977 ]
 [0.07420279 0.02162452 0.0091008  ... 0.00118054 0.01838588 0.0483524 ]
 [0.02139236 0.03316466 0.03697724 ... 0.00437137 0.02091392 0.02517418]]
```

In [35]:
```
comp = pd.DataFrame(pca.components_, columns = data_pca.columns)
comp.head()
```

Out[35]:

	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP	NASDAQ.ADSK
0	-0.039258	-0.041064	-0.062908	-0.009788	-0.035866	-0.054668
1	-0.064284	0.033861	0.001861	-0.032453	0.043511	-0.029519
2	-0.039858	0.064165	0.012079	0.043266	-0.037239	0.040506
3	0.007578	0.077164	0.008564	-0.027896	-0.017418	0.008973
4	-0.033303	-0.016981	0.002438	-0.038330	-0.102023	-0.034831

5 rows × 500 columns

In [ ]: