Problem Statement

```
Predictive Modeling for Financial Market Analysis
```

Importing important libraries

In [1]:

```
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
import category_encoders as ce
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.metrics import accuracy_score , confusion_matrix
```

In [2]:
```
df = pd.read_csv("healthcare_finance.csv")
df
```

Out[2]:

	Unnamed: 0	Symbol	Name	Price (Intraday)	Change	% Change	Volume	Av Vol mont
0	0	BFAFX	American Funds Bond Fund of Amer F1	-0.05	-0.45%	11.01	11.17	11.
1	1	ABNFX	American Funds Bond Fund of Amer F2	-0.05	-0.45%	11.01	11.17	11.
2	2	CFAEX	American Funds Bond Fund of Amer 529E	-0.05	-0.45%	11.01	11.17	11.
3	3	CFAFX	American Funds Bond Fund of Amer 529F	-0.05	-0.45%	11.01	11.17	11.
4	4	RBFEX	American Funds Bond Fund of Amer R4	-0.05	-0.45%	11.01	11.17	11.
...	...	...	...	...	...	...	...	
445	445	0P0001M384	Polar Capital Fut Healthcare A acc USD	0.83	+0.85%	96.93	100.68	93.(
446	446	0P0001LUQZ	Invesco Funds - Invesco China Health Care Equi...	-0.01	-0.23%	4.2800	4.22	4.5
447	447	0P00000BON	Candriam Equities L Biotechnology	5.16	+0.71%	728.37	765.35	718.(
448	448	0P0001HJ7Z	Bellevue (Lux) Bellevue Dgtl Hthl AI2USD	1.53	+1.18%	129.78	135.55	133.3
449	449	0P0001BM5E	Sector Healthcare Value B USD	1.47	+0.91%	162.39	167.81	165.

450 rows × 11 columns

Checking null values

In [3]:
```
df.isnull().mean()*100>0
```

Out[3]:
```
Unnamed: 0           False
Symbol               False
Name                 False
Price (Intraday)     False
Change               False
% Change             False
Volume               False
Avg Vol (3 month)    False
Market Cap            True
PE Ratio (TTM)        True
52 Week Range         True
dtype: bool
```

In [4]:
```
df.isna().sum()
```

Out[4]:
```
Unnamed: 0             0
Symbol                 0
Name                   0
Price (Intraday)       0
Change                 0
% Change               0
Volume                 0
Avg Vol (3 month)      0
Market Cap             5
PE Ratio (TTM)         5
52 Week Range        450
dtype: int64
```

In [5]:
```
df.drop(columns=['52 Week Range'], inplace=True)
df.drop(columns=["Unnamed: 0"], inplace=True)
df.drop(columns=["Symbol"], inplace=True)
df.drop(columns=["PE Ratio (TTM)"], inplace=True)
```

In [6]:
```
df
```

Out[6]:

	Name	Price (Intraday)	Change	% Change	Volume	Avg Vol (3 month)	Market Cap
0	American Funds Bond Fund of Amer F1	-0.05	-0.45%	11.01	11.17	11.14	-0.84%
1	American Funds Bond Fund of Amer F2	-0.05	-0.45%	11.01	11.17	11.14	-0.76%
2	American Funds Bond Fund of Amer 529E	-0.05	-0.45%	11.01	11.17	11.14	-0.88%
3	American Funds Bond Fund of Amer 529F	-0.05	-0.45%	11.01	11.17	11.14	-0.79%
4	American Funds Bond Fund of Amer R4	-0.05	-0.45%	11.01	11.17	11.14	-0.82%
...	...	...	...	...	...	...	...
445	Polar Capital Fut Healthcare A acc USD	0.83	+0.85%	96.93	100.68	93.09	9.32%
446	Invesco Funds - Invesco China Health Care Equi...	-0.01	-0.23%	4.2800	4.22	4.52	-11.42%
447	Candriam Equities L Biotechnology	5.16	+0.71%	728.37	765.35	718.69	2.78%
448	Bellevue (Lux) Bellevue Dgtl Hthl AI2USD	1.53	+1.18%	129.78	135.55	133.36	1.49%
449	Sector Healthcare Value B USD	1.47	+0.91%	162.39	167.81	165.17	3.27%

450 rows × 7 columns

Encoding

In [7]:

```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Select the column you want to scale
column_to_scale = ['Price (Intraday)']

# Remove commas from the column
df[column_to_scale] = df[column_to_scale].replace({',': ''}, regex=True

# Convert the column to float
df[column_to_scale] = df[column_to_scale].astype(float)

# Apply the scaler to the selected column
df[column_to_scale] = scaler.fit_transform(df[column_to_scale])

# Display the updated DataFrame
print(df)
```

```
                                                  Name  Price (Intrada
y)  \
0                American Funds Bond Fund of Amer F1          0.3008
78
1                American Funds Bond Fund of Amer F2          0.3008
78
2              American Funds Bond Fund of Amer 529E          0.3008
78
3              American Funds Bond Fund of Amer 529F          0.3008
78
4                American Funds Bond Fund of Amer R4          0.3008
78
..                                             ...
...
445            Polar Capital Fut Healthcare A acc USD          0.3092
78
446  Invesco Funds - Invesco China Health Care Equi...          0.3012
60
447                Candriam Equities L Biotechnology          0.3506
11
448          Bellevue (Lux) Bellevue Dgtl Hthl AI2USD          0.3159
60
449                    Sector Healthcare Value B USD          0.3153
88

     Change % Change  Volume Avg Vol (3 month) Market Cap
0    -0.45%    11.01   11.17             11.14     -0.84%
1    -0.45%    11.01   11.17             11.14     -0.76%
2    -0.45%    11.01   11.17             11.14     -0.88%
3    -0.45%    11.01   11.17             11.14     -0.79%
4    -0.45%    11.01   11.17             11.14     -0.82%
..      ...      ...     ...               ...        ...
445  +0.85%    96.93  100.68             93.09      9.32%
446  -0.23%   4.2800    4.22              4.52    -11.42%
447  +0.71%   728.37  765.35            718.69      2.78%
448  +1.18%   129.78  135.55            133.36      1.49%
449  +0.91%   162.39  167.81            165.17      3.27%

[450 rows x 7 columns]
```

In [8]:
```
def frequency_encoding(colName):
        enc_Name = (df.groupby(colName).size())/len(df)
        df['Name_encode']= df[colName].apply(lambda x : enc_Name[x])
```

In [9]:
```
frequency_encoding("Name")
```

In [10]:

```
# Print unique values in 'Market Cap' before conversion
print("Unique values in 'Market Cap' before conversion:")
print(df['Market Cap'].unique())

# Perform the conversion operations, handling percentage signs
df['Market Cap'] = df['Market Cap'].str.rstrip('%').astype(str).str.rep

# Check unique values in 'Market Cap' after conversion
print("Unique values in 'Market Cap' after conversion:")
print(df['Market Cap'].unique())
```

```
Unique values in 'Market Cap' before conversion:
['-0.84%' '-0.76%' '-0.88%' '-0.79%' '-0.82%' '-0.75%' '-0.74%' '-1.0
2%'
 '-1.00%' '-0.93%' '-0.89%' '-0.83%' '-1.01%' '-0.77%' '5.07%' '5.08%'
 '8.32%' '7.95%' '7.98%' '8.75%' '8.65%' '8.57%' '8.46%' '8.69%' '8.7
8%'
 '6.82%' '-0.57%' '-0.32%' '-0.34%' '-0.28%' '-0.54%' '-0.25%' '-0.3
0%'
 '-0.22%' '9.05%' '8.47%' '8.53%' '8.56%' '8.52%' '6.71%' '6.52%' '6.6
5%'
 '6.58%' '6.39%' '10.54%' '8.60%' '0.75%' '0.76%' '0.62%' '0.85%' '0.8
8%'
 '0.81%' '-0.15%' '-0.23%' '0.04%' '-0.04%' '0.02%' '-0.11%' '1.28%'
 '1.51%' '1.46%' '1.50%' '1.47%' '1.40%' '1.53%' '8.20%' '8.30%' '-0.3
6%'
 '-0.47%' '-0.48%' '0.23%' '0.12%' '0.25%' '0.18%' '0.13%' '0.22%' '0.
17%'
 '0.01%' '-0.08%' '0.06%' '0.07%' '-0.09%' '0.20%' '13.08%' '12.97%'
 '12.80%' '13.15%' '13.10%' '13.17%' '12.93%' '1.32%' '8.62%' '8.36%'
 '1.19%' '1.08%' '1.14%' '1.29%' '1.20%' '1.13%' '1.04%' '4.67%' '4.6
5%'
 '4.47%' '4.76%' nan '13.16%' '13.27%' '13.26%' '3.13%' '10.35%' '10.3
8%'
 '10.16%' '18.51%' '10.43%' '3.08%' '7.51%' '7.49%' '7.39%' '7.47%'
 '7.48%' '7.15%' '7.30%' '7.96%' '7.82%' '7.92%' '11.55%' '11.58%'
 '11.47%' '11.28%' '2.44%' '8.12%' '2.43%' '2.30%' '2.37%' '11.30%'
 '11.12%' '10.98%' '11.21%' '4.73%' '4.53%' '4.54%' '4.39%' '4.60%'
 '4.64%' '4.48%' '2.75%' '2.79%' '11.22%' '11.57%' '7.79%' '7.77%' '7.
70%'
 '9.96%' '10.19%' '10.12%' '1.11%' '1.36%' '1.84%' '-0.26%' '2.82%'
 '2.67%' '2.49%' '2.84%' '-0.27%' '8.87%' '8.84%' '8.92%' '9.81%' '9.6
3%'
 '9.90%' '7.50%' '-0.06%' '6.83%' '9.29%' '9.46%' '9.13%' '7.57%' '0.5
5%'
 '0.73%' '0.80%' '-0.95%' '-0.78%' '6.99%' '5.10%' '18.06%' '18.43%'
 '3.95%' '3.82%' '1.16%' '1.60%' '7.13%' '7.19%' '7.11%' '6.93%' '7.0
2%'
 '11.91%' '12.18%' '9.40%' '9.53%' '1.34%' '1.52%' '5.64%' '5.78%' '5.
50%'
 '-7.91%' '11.25%' '11.17%' '8.54%' '8.33%' '8.61%' '8.63%' '8.66%'
 '-10.14%' '-10.37%' '-10.62%' '-10.41%' '-13.40%' '-13.56%' '-15.26%'
 '4.75%' '4.46%' '-14.26%' '-14.06%' '-13.80%' '-13.79%' '2.04%' '2.0
3%'
 '1.74%' '6.05%' '5.68%' '5.85%' '4.29%' '5.30%' '5.84%' '5.88%' '6.1
1%'
 '5.98%' '3.41%' '0.14%' '13.68%' '0.84%' '7.04%' '2.12%' '11.61%' '0.
59%'
 '1.07%' '0.09%' '9.55%' '0.87%' '7.81%' '7.83%' '4.27%' '-1.55%' '9.7
4%'
 '-0.65%' '-0.00%' '5.20%' '1.56%' '1.27%' '7.73%' '3.23%' '1.81%'
 '10.00%' '11.75%' '10.78%' '9.23%' '6.89%' '7.75%' '4.41%' '4.33%'
 '7.71%' '3.70%' '7.78%' '1.31%' '2.26%' '-1.12%' '9.52%' '3.76%' '8.4
8%'
 '7.22%' '2.78%' '7.65%' '-9.25%' '1.55%' '13.96%' '3.11%' '9.54%' '4.
57%'
 '6.72%' '1.26%' '6.73%' '9.32%' '-11.42%' '1.49%' '3.27%']
Unique values in 'Market Cap' after conversion:
[-8.400e+05 -7.600e+05 -8.800e+05 -7.900e+05 -8.200e+05 -7.500e+05
 -7.400e+05 -1.020e+06 -1.000e+06 -9.300e+05 -8.900e+05 -8.300e+05
 -1.010e+06 -7.700e+05  5.070e+06  5.080e+06  8.320e+06  7.950e+06
  7.980e+06  8.750e+06  8.650e+06  8.570e+06  8.460e+06  8.690e+06
```

```
 8.780e+06  6.820e+06 -5.700e+05 -3.200e+05 -3.400e+05 -2.800e+05
-5.400e+05 -2.500e+05 -3.000e+05 -2.200e+05  9.050e+06  8.470e+06
 8.530e+06  8.560e+06  8.520e+06  6.710e+06  6.520e+06  6.650e+06
 6.580e+06  6.390e+06  1.054e+07  8.600e+06  7.500e+05  7.600e+05
 6.200e+05  8.500e+05  8.800e+05  8.100e+05 -1.500e+05 -2.300e+05
 4.000e+04 -4.000e+04  2.000e+04 -1.100e+05  1.280e+06  1.510e+06
 1.460e+06  1.500e+06  1.470e+06  1.400e+06  1.530e+06  8.200e+06
 8.300e+06 -3.600e+05 -4.700e+05 -4.800e+05  2.300e+05  1.200e+05
 2.500e+05  1.800e+05  1.300e+05  2.200e+05  1.700e+05  1.000e+04
-8.000e+04  6.000e+04  7.000e+04 -9.000e+04  2.000e+05  1.308e+07
 1.297e+07  1.280e+07  1.315e+07  1.310e+07  1.317e+07  1.293e+07
 1.320e+06  8.620e+06  8.360e+06  1.190e+06  1.080e+06  1.140e+06
 1.290e+06  1.200e+06  1.130e+06  1.040e+06  4.670e+06  4.650e+06
 4.470e+06  4.760e+06         nan  1.316e+07  1.327e+07  1.326e+07
 3.130e+06  1.035e+07  1.038e+07  1.016e+07  1.851e+07  1.043e+07
 3.080e+06  7.510e+06  7.490e+06  7.390e+06  7.470e+06  7.480e+06
 7.150e+06  7.300e+06  7.960e+06  7.820e+06  7.920e+06  1.155e+07
 1.158e+07  1.147e+07  1.128e+07  2.440e+06  8.120e+06  2.430e+06
 2.300e+06  2.370e+06  1.130e+07  1.112e+07  1.098e+07  1.121e+07
 4.730e+06  4.530e+06  4.540e+06  4.390e+06  4.600e+06  4.640e+06
 4.480e+06  2.750e+06  2.790e+06  1.122e+07  1.157e+07  7.790e+06
 7.770e+06  7.700e+06  9.960e+06  1.019e+07  1.012e+07  1.110e+06
 1.360e+06  1.840e+06 -2.600e+05  2.820e+06  2.670e+06  2.490e+06
 2.840e+06 -2.700e+05  8.870e+06  8.840e+06  8.920e+06  9.810e+06
 9.630e+06  9.900e+06  7.500e+06 -6.000e+04  6.830e+06  9.290e+06
 9.460e+06  9.130e+06  7.570e+06  5.500e+05  7.300e+05  8.000e+05
-9.500e+05 -7.800e+05  6.990e+06  5.100e+06  1.806e+07  1.843e+07
 3.950e+06  3.820e+06  1.160e+06  1.600e+06  7.130e+06  7.190e+06
 7.110e+06  6.930e+06  7.020e+06  1.191e+07  1.218e+07  9.400e+06
 9.530e+06  1.340e+06  1.520e+06  5.640e+06  5.780e+06  5.500e+06
-7.910e+06  1.125e+07  1.117e+07  8.540e+06  8.330e+06  8.610e+06
 8.630e+06  8.660e+06 -1.014e+07 -1.037e+07 -1.062e+07 -1.041e+07
-1.340e+07 -1.356e+07 -1.526e+07  4.750e+06  4.460e+06 -1.426e+07
-1.406e+07 -1.380e+07 -1.379e+07  2.040e+06  2.030e+06  1.740e+06
 6.050e+06  5.680e+06  5.850e+06  4.290e+06  5.300e+06  5.840e+06
 5.880e+06  6.110e+06  5.980e+06  3.410e+06  1.400e+05  1.368e+07
 8.400e+05  7.040e+06  2.120e+06  1.161e+07  5.900e+05  1.070e+06
 9.000e+04  9.550e+06  8.700e+05  7.810e+06  7.830e+06  4.270e+06
-1.550e+06  9.740e+06 -6.500e+05 -0.000e+00  5.200e+06  1.560e+06
 1.270e+06  7.730e+06  3.230e+06  1.810e+06  1.000e+07  1.175e+07
 1.078e+07  9.230e+06  6.890e+06  7.750e+06  4.410e+06  4.330e+06
 7.710e+06  3.700e+06  7.780e+06  1.310e+06  2.260e+06 -1.120e+06
 9.520e+06  3.760e+06  8.480e+06  7.220e+06  2.780e+06  7.650e+06
-9.250e+06  1.550e+06  1.396e+07  3.110e+06  9.540e+06  4.570e+06
 6.720e+06  1.260e+06  6.730e+06  9.320e+06 -1.142e+07  1.490e+06
 3.270e+06]
```

In [11]:
```
df.drop(columns=['Name'], inplace=True)
```

In [12]:
```
# Remove ',' and 'M' from 'Volume' column and convert to float
df['Volume'] = df['Volume'].astype(str).str.replace(',', '').str.rstrip
```

In [13]:
```
# Remove ',' from 'Avg Vol (3 month)' column and convert to float
df['Avg Vol (3 month)'] = df['Avg Vol (3 month)'].astype(str).str.repla
```

In [14]:
```
df
```

Out[14]:

	Price (Intraday)	Change	% Change	Volume	Avg Vol (3 month)	Market Cap	Name_encode
0	0.300878	-0.45%	11.01	11170000.0	11140000.0	-840000.0	0.002222
1	0.300878	-0.45%	11.01	11170000.0	11140000.0	-760000.0	0.002222
2	0.300878	-0.45%	11.01	11170000.0	11140000.0	-880000.0	0.002222
3	0.300878	-0.45%	11.01	11170000.0	11140000.0	-790000.0	0.002222
4	0.300878	-0.45%	11.01	11170000.0	11140000.0	-820000.0	0.002222
...	...	...	...	...	...	...	...
445	0.309278	+0.85%	96.93	100680000.0	93090000.0	9320000.0	0.002222
446	0.301260	-0.23%	4.2800	4220000.0	4520000.0	-11420000.0	0.002222
447	0.350611	+0.71%	728.37	765350000.0	718690000.0	2780000.0	0.004444
448	0.315960	+1.18%	129.78	135550000.0	133360000.0	1490000.0	0.002222
449	0.315388	+0.91%	162.39	167810000.0	165170000.0	3270000.0	0.002222

450 rows × 7 columns

Scaling

In [15]:
```
df.drop(columns=['Change'], inplace=True)
```

In [16]:

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Select only the numerical columns you want to scale
numerical_columns = ['Price (Intraday)', '% Change', 'Volume', 'Avg Vol

# Remove commas from all numerical columns
df[numerical_columns] = df[numerical_columns].replace({',': ''}, regex=

# Convert all numerical columns to float
df[numerical_columns] = df[numerical_columns].astype(float)

# Apply the scaler to the selected columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Display the updated DataFrame
print(df)
```

```
     Price (Intraday)  % Change    Volume  Avg Vol (3 month)  Market C
ap  \
0            0.300878  0.001304  0.001315           0.001196    0.4270
06
1            0.300878  0.001304  0.001315           0.001196    0.4293
75
2            0.300878  0.001304  0.001315           0.001196    0.4258
22
3            0.300878  0.001304  0.001315           0.001196    0.4284
87
4            0.300878  0.001304  0.001315           0.001196    0.4275
98
..                ...       ...       ...                ...
...
445          0.309278  0.012323  0.012728           0.010718    0.7278
65
446          0.301260  0.000441  0.000428           0.000426    0.1137
10
447          0.350611  0.093304  0.097477           0.083409    0.5342
02
448          0.315960  0.016536  0.017174           0.015397    0.4960
02
449          0.315388  0.020718  0.021287           0.019093    0.5487
12

     Name_encode
0       0.002222
1       0.002222
2       0.002222
3       0.002222
4       0.002222
..           ...
445     0.002222
446     0.002222
447     0.004444
448     0.002222
449     0.002222

[450 rows x 6 columns]
```

In [17]:
```
df
```

Out[17]:

	Price (Intraday)	% Change	Volume	Avg Vol (3 month)	Market Cap	Name_encode
0	0.300878	0.001304	0.001315	0.001196	0.427006	0.002222
1	0.300878	0.001304	0.001315	0.001196	0.429375	0.002222
2	0.300878	0.001304	0.001315	0.001196	0.425822	0.002222
3	0.300878	0.001304	0.001315	0.001196	0.428487	0.002222
4	0.300878	0.001304	0.001315	0.001196	0.427598	0.002222
...	...	...	...	...	...	...
445	0.309278	0.012323	0.012728	0.010718	0.727865	0.002222
446	0.301260	0.000441	0.000428	0.000426	0.113710	0.002222
447	0.350611	0.093304	0.097477	0.083409	0.534202	0.004444
448	0.315960	0.016536	0.017174	0.015397	0.496002	0.002222
449	0.315388	0.020718	0.021287	0.019093	0.548712	0.002222

450 rows × 6 columns

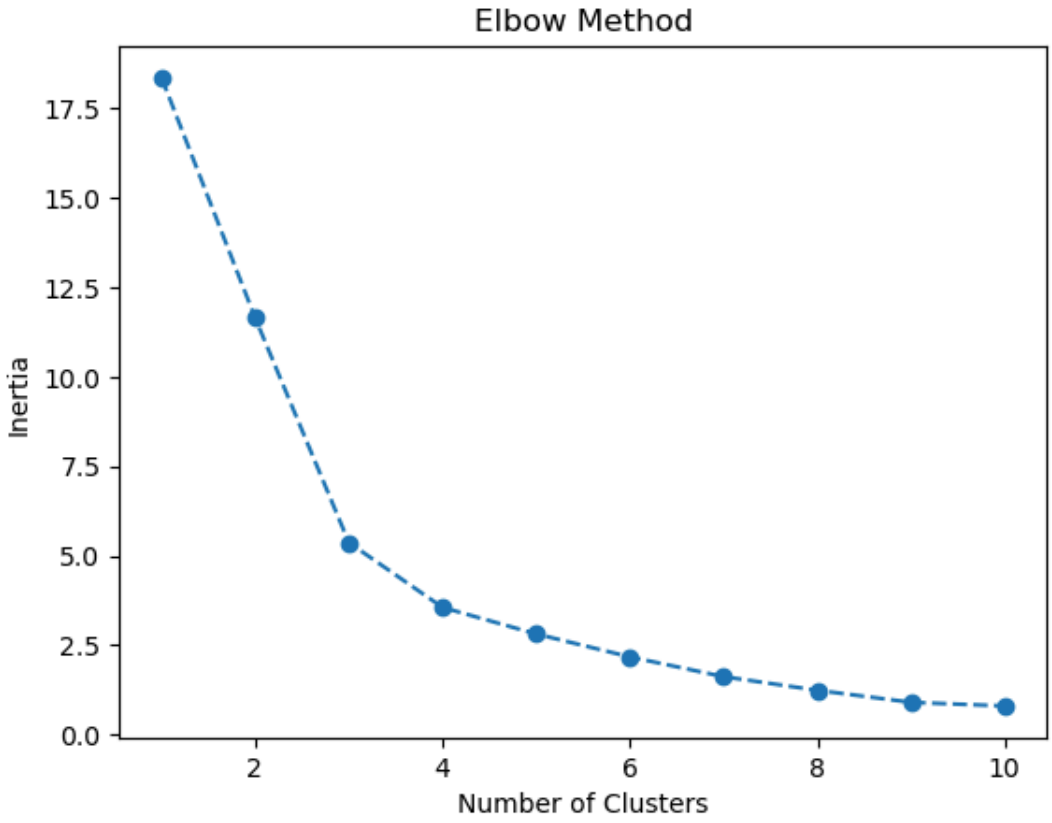
Clustering

In [18]:

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

# Select the relevant columns for clustering
cols_to_cluster = ['Price (Intraday)', 'Volume', 'Avg Vol (3 month)', '
data_for_clustering = df[cols_to_cluster]

# Impute NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')
data_for_clustering_imputed = imputer.fit_transform(data_for_clustering

# Determine the optimal number of clusters using the elbow method
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(data_for_clustering_imputed)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
```

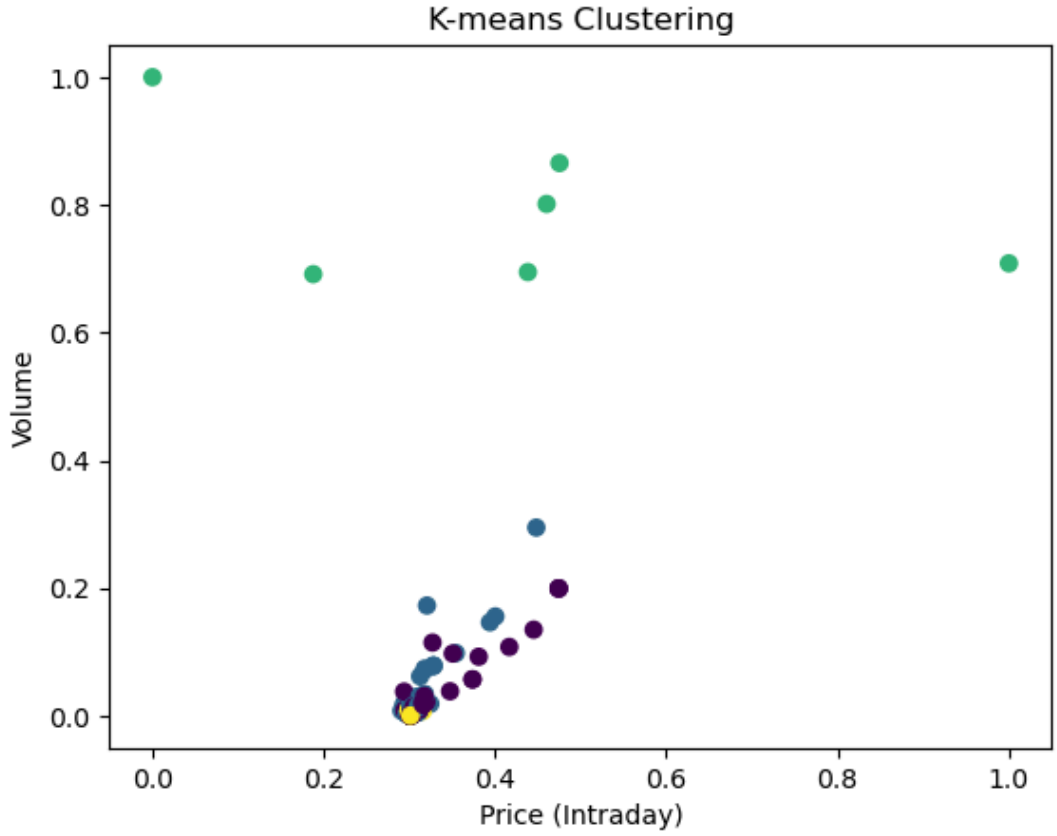
```
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```


Elbow Method
Inertia
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2
4
6
8
10
Number of Clusters

In [19]:

```
# Perform k-means clustering
kmeans = KMeans(n_clusters=4, init='k-means++', random_state=42)
kmeans.fit(data_for_clustering_imputed)

# Add cluster labels to the original DataFrame
df['Cluster'] = kmeans.labels_

# Display the cluster centers
cluster_centers = pd.DataFrame(
    imputer.statistics_.reshape(1, -1),  # Using imputer.statistics_ in
    columns=cols_to_cluster
)
print("Cluster Centers:")
print(cluster_centers)

# Visualize the clusters (for 2D data)
# Assuming you want to visualize the 'Price' and 'Volume' columns
plt.scatter(df['Price (Intraday)'], df['Volume'], c=df['Cluster'], cmap
plt.xlabel('Price (Intraday)')
plt.ylabel('Volume')
plt.title('K-means Clustering')
plt.show()
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1412: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.p
y:1436: UserWarning: KMeans is known to have a memory leak on Windows
with MKL, when there are less chunks than available threads. You can a
void it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(

Cluster Centers:
   Price (Intraday)    Volume  Avg Vol (3 month)  Market Cap
0          0.307318  0.021013           0.018255     0.57772
```



In [20]:
```
df
```

Out[20]:

	Price (Intraday)	% Change	Volume	Avg Vol (3 month)	Market Cap	Name_encode	Cluster
0	0.300878	0.001304	0.001315	0.001196	0.427006	0.002222	0
1	0.300878	0.001304	0.001315	0.001196	0.429375	0.002222	0
2	0.300878	0.001304	0.001315	0.001196	0.425822	0.002222	0
3	0.300878	0.001304	0.001315	0.001196	0.428487	0.002222	0
4	0.300878	0.001304	0.001315	0.001196	0.427598	0.002222	0
...	...	...	...	...	...	...	...
445	0.309278	0.012323	0.012728	0.010718	0.727865	0.002222	1
446	0.301260	0.000441	0.000428	0.000426	0.113710	0.002222	3
447	0.350611	0.093304	0.097477	0.083409	0.534202	0.004444	0
448	0.315960	0.016536	0.017174	0.015397	0.496002	0.002222	0
449	0.315388	0.020718	0.021287	0.019093	0.548712	0.002222	0

450 rows × 7 columns

KNN

```
1. KNN is a simple and intuitive classification algorithm that works by
finding the K nearest data points in the training set to a given test
point and then assigns the label of the majority class among those K
neighbors.
```

```
2. It's a non-parametric algorithm, meaning it doesn't make any
assumptions about the underlying data distribution.
3. KNN's performance heavily depends on the choice of K and the distance
metric used to measure the similarity between data points.
```

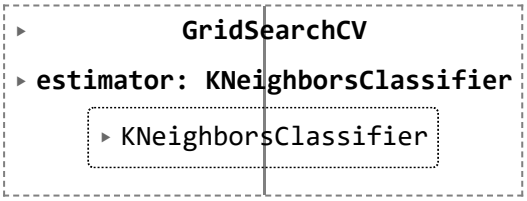
In [21]:
```
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [22]:
```
X = df.drop('Cluster', axis=1)
y = df['Cluster']

# Impute missing values in X
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_
```

In [23]:
```
# Hyperparameter tuning using GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9]}
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\model_selection\_
split.py:725: UserWarning: The least populated class in y has only 4 m
embers, which is less than n_splits=5.
  warnings.warn(
```

Out[23]:



In [24]:
```
# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Evaluating the model on the test set
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Best Parameters: {'n_neighbors': 3}
Best Score: 0.9944444444444445
Accuracy: 0.9777777777777777
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        47
           1       1.00      0.95      0.97        39
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2

    accuracy                           0.98        90
   macro avg       0.99      0.99      0.99        90
weighted avg       0.98      0.98      0.98        90
```

Logestic Regression

1. Despite its name, logistic regression is a linear model for binary classification that predicts the probability that a given input belongs to a certain class.
2. It models the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function.
3. Logistic regression is often used as a baseline model for binary classification tasks due to its simplicity and interpretability.

In [25]:
```
from sklearn.linear_model import LogisticRegression

# Hyperparameter tuning using GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
logistic_reg = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(logistic_reg, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Evaluating the model on the test set
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\model_selection\_
split.py:725: UserWarning: The least populated class in y has only 4 m
embers, which is less than n_splits=5.
  warnings.warn(

Best Parameters: {'C': 100}
Best Score: 0.986111111111111
Accuracy: 0.9888888888888889
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        47
           1       1.00      0.97      0.99        39
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2

    accuracy                           0.99        90
   macro avg       0.99      0.99      0.99        90
weighted avg       0.99      0.99      0.99        90
```

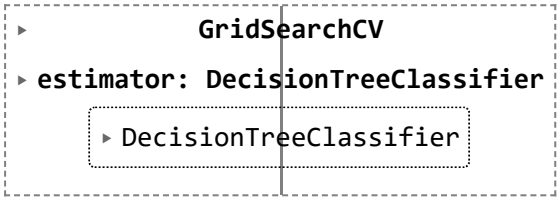
Decision Tree

1. Decision trees are a popular and powerful machine learning algorithm used for both classification and regression tasks.
2. They work by recursively partitioning the feature space into subsets based on the values of the features, with each partition aiming to minimize impurity or maximize information gain.
3. Decision trees are easy to interpret and visualize, making them useful for understanding the decision-making process of the model.

In [26]:
```
from sklearn.tree import DecisionTreeClassifier

# Hyperparameter tuning using GridSearchCV
param_grid = {'max_depth': [None, 5, 10, 15, 20]}
decision_tree = DecisionTreeClassifier()
grid_search = GridSearchCV(decision_tree, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\model_selection\_
split.py:725: UserWarning: The least populated class in y has only 4 m
embers, which is less than n_splits=5.
  warnings.warn(
```

Out[26]:



In [27]:
```
# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'max_depth': None}
Best Score: 0.9944444444444445
```

In [28]:
```
# Evaluating the model on the test set
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        39
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2

    accuracy                           1.00        90
   macro avg       1.00      1.00      1.00        90
weighted avg       1.00      1.00      1.00        90
```

Random Forest

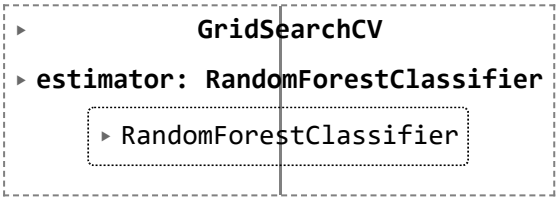
```
1. Random Forest is an ensemble learning method that builds multiple
decision trees during training and outputs the class that is the mode of
the classes (classification) or mean prediction (regression) of the
individual trees.
```

2. It combines the predictions of multiple decision trees to improve the overall accuracy and generalization of the model.
3. Random Forests are robust to overfitting and handle high-dimensional data well, making them popular for a wide range of classification and regression tasks.

In [29]:
```
from sklearn.ensemble import RandomForestClassifier

# Hyperparameter tuning using GridSearchCV
param_grid = {'n_estimators': [100, 200, 300],
              'max_depth': [None, 5, 10, 15, 20]}
random_forest = RandomForestClassifier()
grid_search = GridSearchCV(random_forest, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\model_selection\_
split.py:725: UserWarning: The least populated class in y has only 4 m
embers, which is less than n_splits=5.
  warnings.warn(
```

Out[29]:



In [30]:
```
# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'max_depth': 10, 'n_estimators': 100}
Best Score: 0.9972222222222221
```

In [31]:
```
# Evaluating the model on the test set
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        39
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2

    accuracy                           1.00        90
   macro avg       1.00      1.00      1.00        90
weighted avg       1.00      1.00      1.00        90
```

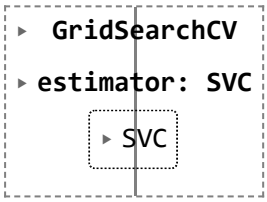
SVM

1. SVM is a powerful supervised learning algorithm used for classification and regression tasks.
2. It works by finding the hyperplane that best separates the classes in the feature space while maximizing the margin between the classes.
3. SVM can handle linear and non-linear decision boundaries using different kernel functions such as linear, polynomial, and radial basis function (RBF) kernels.

In [32]:
```
from sklearn.svm import SVC

# Hyperparameter tuning using GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
C:\Users\Kashish\anaconda3\Lib\site-packages\sklearn\model_selection\_
split.py:725: UserWarning: The least populated class in y has only 4 m
embers, which is less than n_splits=5.
  warnings.warn(
```

Out[32]:



In [33]:
```
# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'linear'}
Best Score: 0.9944444444444445
```

In [34]:
```
# Evaluating the model on the test set
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9888888888888889
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        47
           1       1.00      0.97      0.99        39
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2

    accuracy                           0.99        90
   macro avg       0.99      0.99      0.99        90
weighted avg       0.99      0.99      0.99        90
```

Naive Bayes

1. Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features.
2. Despite its simplicity, Naive Bayes often performs surprisingly well in practice, especially for text classification tasks.
3. It's particularly useful when dealing with high-dimensional datasets and has low computational overhead, making it efficient for large-scale applications.

In [35]:
```
from sklearn.naive_bayes import GaussianNB

# Creating and fitting the model
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)

# Evaluating the model on the test set
y_pred = naive_bayes.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9222222222222223
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.87      0.93        47
           1       0.86      0.97      0.92        39
           2       0.67      1.00      0.80         2
           3       1.00      1.00      1.00         2

    accuracy                           0.92        90
   macro avg       0.88      0.96      0.91        90
weighted avg       0.93      0.92      0.92        90
```

ROC

The ROC curve provides a visual representation of the trade-off between the true positive rate and the false positive rate across different threshold settings. It's a useful tool for evaluating and comparing the performance of binary classification algorithms.

In [37]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for each class
y_prob_svm = grid_search.best_estimator_.predict(X_test)

# Compute ROC curve and AUC for each class
fpr_svm = dict()
tpr_svm = dict()
roc_auc_svm = dict()
for i in range(len(grid_search.best_estimator_.classes_)):
    fpr_svm[i], tpr_svm[i], _ = roc_curve(y_test, y_prob_svm[:, i])
    roc_auc_svm[i] = auc(fpr_svm[i], tpr_svm[i])

# Plot ROC curve for each class
plt.figure(figsize=(8, 6))
for i in range(len(grid_search.best_estimator_.classes_)):
    plt.plot(fpr_svm[i], tpr_svm[i], label='ROC curve (area = %0.2f) fo
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for SVM')
plt.legend(loc="lower right")
plt.show()
```

```
---------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
Cell In[37], line 12
     10 roc_auc_svm = dict()
     11 for i in range(len(grid_search.best_estimator_.classes_)):
---> 12     fpr_svm[i], tpr_svm[i], _ = roc_curve(y_test, y_prob_svm
[:, i])
     13     roc_auc_svm[i] = auc(fpr_svm[i], tpr_svm[i])
     15 # Plot ROC curve for each class

IndexError: too many indices for array: array is 1-dimensional, but 2
were indexed
```

In [ ]:
```
```