In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans   #安裝 conda install scikit-learn
```

In [2]:
```
#使用pandas套件的read_csv讀入資料,並以DataFram儲存(2D)
rfm = pd.read_csv('clustering_Ex1.csv')
```

In [3]:
```
#檢查rfm的資料型態(DataFrame是二維的結構)
#在Spyder要使用 print(type(rfm))
type(rfm)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

In [4]:
```
#印出rfm的內容 (43672 rows × 5 columns)
#在Spyder要使用 print(rfm)
#顯示的結果在最前面多一個索引的欄位(pandas預設的),流水號, 從0開始
rfm
```

Out[4]:

	cid	gender	R	F	M
0	0S3670071489	F	586	1	1380
1	0S3687895473	M	12	1	1186
2	0S3690675977	F	75	2	2850
3	2009S2044237	F	657	1	1099
4	2009S2044261	M	145	1	1360
...	...	...	...	...	...
43667	S201S242852053	F	211	1	678
43668	S201S280812053	F	109	2	3120
43669	S201S280816053	M	313	1	780
43670	S201S280840053	M	33	1	150
43671	S201S282920053	F	24	2	2590

43672 rows × 5 columns

In [5]:
```
#印出rfm的資訊, 查看各個欄位的基本資料:名稱,非空值數, 資料型態。
#欄位同樣有預設的索引(pandas預設的),流水號, 從0開始
rfm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43672 entries, 0 to 43671
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   cid     43672 non-null  object
 1   gender  43672 non-null  object
 2   R       43672 non-null  int64 
 3   F       43672 non-null  int64 
 4   M       43672 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 1.7+ MB
```

In [6]:
```
#使用iloc裁切需要的資料。
#[:,]表示所有列的資料都要。
#[ ,2:5]表示切出欄位索引編號 2,3,4 的資料
data = rfm.iloc[:, 2:5]
print(data)
```

```
         R  F     M
0      586  1  1380
1       12  1  1186
2       75  2  2850
3      657  1  1099
4      145  1  1360
...    ... ..   ...
43667  211  1   678
43668  109  2  3120
43669  313  1   780
43670   33  1   150
43671   24  2  2590

[43672 rows x 3 columns]
```

In [7]:
```
#另一種選擇欄位的方法是使用欄位名稱
df = rfm[['R', 'F', 'M']]
df.head()
```

Out[7]:

	R	F	M
0	586	1	1380
1	12	1	1186
2	75	2	2850
3	657	1	1099
4	145	1	1360

In [8]:
```
#查看裁切出的data各個欄位的統計值
#觀察結果可以看出 R, F, M 的數值差距很大
data.describe()
```

Out[8]:

	R	F	M
count	43672.000000	43672.000000	43672.000000
mean	312.230560	1.788835	2595.994298
std	197.701405	2.262211	4945.038093
min	1.000000	1.000000	29.000000
25%	141.000000	1.000000	870.000000
50%	296.000000	1.000000	1380.000000
75%	484.000000	2.000000	2580.000000
max	669.000000	112.000000	233136.000000

In [9]:
```
#若直接用原始資料進行分群，分群的依據受到大數值的影響較大，而有不公平的現象
#俗稱小尺度的資料被大尺度的資料 "吃掉"
#因此通常會將資料進行標準化(或歸一化)。
#常用的方式有: 最小-最大值標準化、Z-score標準化。
#最小-最大值標準化: 資料轉換為 (0,1) 範圍的數, [(原始資料 x)-(原始資料中最小值)]/[(原始資料中最大值 x)-(原始資料中最小值)]
#Z-score標準化: 資料轉換為標準常態分配(平均數=0,標準差=1)
```

In [10]:
```
#最小值最大值正規化(Min-Max Normalization)
from sklearn.preprocessing import MinMaxScaler

#建立轉換資料的框架 scaler, 範圍在 0~1 之間
scaler = MinMaxScaler(feature_range=(0,1))
```

In [11]:
```
#正式做資料轉換,其中先將資料(dataFrame)轉換為 numpy,再用 scale框架轉換資料
df_minmax = scaler.fit_transform(data.to_numpy())

#檢視轉換後的內容(介於0~1之間), 此時的資料型態是 numpy(2d)
df_minmax
```

Out[11]:
```
array([[8.75748503e-01, 0.00000000e+00, 5.79562175e-03],
       [1.64670659e-02, 0.00000000e+00, 4.96338591e-03],
       [1.10778443e-01, 9.00900901e-03, 1.21017387e-02],
       ...,
       [4.67065868e-01, 0.00000000e+00, 3.22169647e-03],
       [4.79041916e-02, 0.00000000e+00, 5.19074931e-04],
       [3.44311377e-02, 9.00900901e-03, 1.09863711e-02]])
```

In [12]:
```
#後面集群分析時需要, 因此再轉回pandas的 DataFrame, 同時指定欄位名稱 R, F, M
#轉換完後顯示資料, 會發現主動給予列索引
df_minmax = pd.DataFrame(df_minmax, columns = ['R', 'F', 'M'])
df_minmax
```

Out[12]:

	R	F	M
0	0.875749	0.000000	0.005796
1	0.016467	0.000000	0.004963
2	0.110778	0.009009	0.012102
3	0.982036	0.000000	0.004590
4	0.215569	0.000000	0.005710
...	...	...	...
43667	0.314371	0.000000	0.002784
43668	0.161677	0.009009	0.013260
43669	0.467066	0.000000	0.003222
43670	0.047904	0.000000	0.000519
43671	0.034431	0.009009	0.010986

43672 rows × 3 columns

In [13]:
```
#使用前面的 from sklearn.cluster import KMeans 套件的集群分析函數 KMeans()
#先建立模型的框架。
#KMeans有很多參數可以設定, 不一定每個參數都需要, 若未設定則使自動使用預設值
# n_clusters=4 分群的數量,預設8群,在此設定4群。
# max_iter=500 分群過程中演算法執行最大迭代數, 在k-means中，如果執行結果收斂的話，是有可能提前中止，而不會執行到最大迭代次數。
#random_state 指定隨機亂數種子, 確保每次分群結果都一樣(若是要調參數,才能比較調整前後的差別)

model_KMC = KMeans(n_clusters = 4, max_iter = 500, random_state = 42)
```

In [14]:
```
# Batch size: 批次,考慮資料量很大，電腦記憶體不夠用,把 資料分批送進模型訓練, 分批的數量就是 Batch size
# Epoch: 期, 訓練模型過程裡，演算法完整使用過資料集每筆資料的狀態。
# Iteration：迭代，訓練過程迭代（重複）了幾次，才能完成了1個 Epoch 的訓練
# 例如:有100筆資料, Batch size=20, 則要5次迭代(Iteration)才能完成一個 Epoch
```

In [15]:
```
#將資料進行分群(計算相似度高(距離較近)的點會被分為同一群)
#分群完後, 每一筆資料會給予一個群編號標籤,0, 1, 2, 3 =>4群
model_KMC.fit_predict(df_minmax)
```

Out[15]:
```
array([1, 2, 2, ..., 0, 2, 2])
```

In [16]:
```
#Series是Pandas的一維陣列結構。
#將model內的標籤存入 Series,再以不同的值分別計數, 印出每一群的個數

r1 = pd.Series(model_KMC.labels_).value_counts()
r1
```

Out[16]:
```
2    12062
0    11837
3    10187
1     9586
dtype: int64
```

In [17]:
```
#將model內的中心點(座標)印出

r2 = pd.DataFrame(model_KMC.cluster_centers_)
r2
```

Out[17]:

	0	1	2
0	0.358468	0.006232	0.010220
1	0.877883	0.001400	0.006335
2	0.105587	0.015541	0.018300
3	0.630672	0.003496	0.007696

In [18]:
```
#合併上述兩個矩陣 (axis=1表示欄合併),得到集群中心和筆數的矩陣
r21 = pd.concat([r2, r1], axis = 1)
r21
```

Out[18]:

	0	1	2	0
0	0.358468	0.006232	0.010220	11837
1	0.877883	0.001400	0.006335	9586
2	0.105587	0.015541	0.018300	12062
3	0.630672	0.003496	0.007696	10187

In [19]:
```
#重新命名 欄位名稱
# list+list
r21.columns = list(df_minmax.columns) + ['size_of_Group']
r21
```

Out[19]:

	R	F	M	size_of_Group
0	0.358468	0.006232	0.010220	11837
1	0.877883	0.001400	0.006335	9586
2	0.105587	0.015541	0.018300	12062
3	0.630672	0.003496	0.007696	10187

In [20]:
```
#將標準化資料 df_minmax 增加群別的欄位 model_KMC.labels_ (axis=1表示欄合併)
r = pd.concat([df_minmax, pd.Series(model_KMC.labels_, index = df_minmax.index)], axis = 1)
r
```

Out[20]:

	R	F	M	0
0	0.875749	0.000000	0.005796	1
1	0.016467	0.000000	0.004963	2
2	0.110778	0.009009	0.012102	2
3	0.982036	0.000000	0.004590	1
4	0.215569	0.000000	0.005710	2
...	...	...	...	...
43667	0.314371	0.000000	0.002784	0
43668	0.161677	0.009009	0.013260	2
43669	0.467066	0.000000	0.003222	0
43670	0.047904	0.000000	0.000519	2
43671	0.034431	0.009009	0.010986	2

43672 rows × 4 columns

In [21]:
```
#重新命名 欄位名稱
r.columns = list(df_minmax.columns) + ['GroupID']
r
```

Out[21]:

	R	F	M	GroupID
0	0.875749	0.000000	0.005796	1
1	0.016467	0.000000	0.004963	2
2	0.110778	0.009009	0.012102	2
3	0.982036	0.000000	0.004590	1
4	0.215569	0.000000	0.005710	2
...	...	...	...	...
43667	0.314371	0.000000	0.002784	0
43668	0.161677	0.009009	0.013260	2
43669	0.467066	0.000000	0.003222	0
43670	0.047904	0.000000	0.000519	2
43671	0.034431	0.009009	0.010986	2

43672 rows × 4 columns

In [22]:
```
#原始資料 rfm 增加一個欄位 "k-means", 其值為分群後的群別
rfm["k-means"] = model_KMC.labels_
rfm
```

Out[22]:

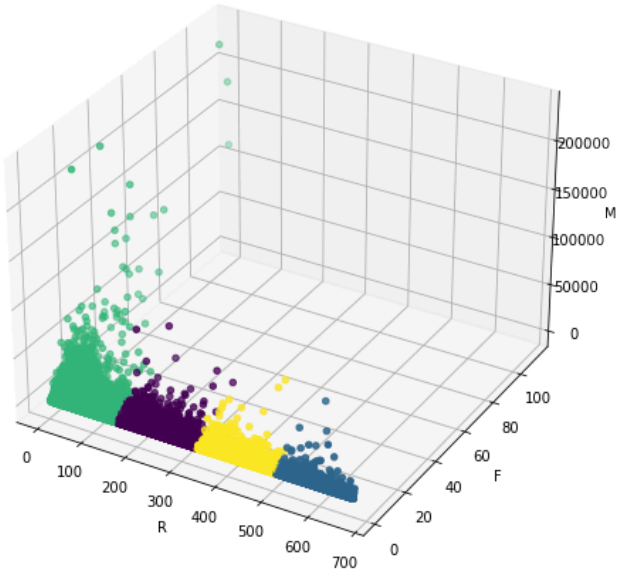
	cid	gender	R	F	M	k-means
0	0S3670071489	F	586	1	1380	1
1	0S3687895473	M	12	1	1186	2
2	0S3690675977	F	75	2	2850	2
3	2009S2044237	F	657	1	1099	1
4	2009S2044261	M	145	1	1360	2
...	...	...	...	...	...	...
43667	S201S242852053	F	211	1	678	0
43668	S201S280812053	F	109	2	3120	2
43669	S201S280816053	M	313	1	780	0
43670	S201S280840053	M	33	1	150	2
43671	S201S282920053	F	24	2	2590	2

43672 rows × 6 columns

In [23]:
```
#將上述結果儲存為 csv 檔
rfm.to_csv("rfm_KMC_Output.csv",index = False)
```

In [24]:
```
#3D散佈圖_精簡方法

import matplotlib.pyplot as plt
dft = pd.read_csv('rfm_KMC_Output.csv')   #讀取csv資料

#設定 10*8 大小的空畫布
fig = plt.figure(figsize = (10, 8))

#設定 3D 圖
ax = fig.gca(projection = '3d')

#繪製散佈圖, 給予三軸資料,
# c(color)使用預設的顏色編號,可以指定 cmap='',不同顏色表示不同群
# marker = 'o' 標記為圖形
ax.scatter(dft['R'], dft['F'], dft['M'], c = dft['k-means'], marker = 'o')
ax.set_xlabel('R')
ax.set_ylabel('F')
ax.set_zlabel('M')

plt.show()
```

```
C:\Users\user\AppData\Local\Temp/ipykernel_22088/948710055.py:10: MatplotlibDeprecationWarning: Calling gca() with
keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword a
rguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with
default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection = '3d')
```

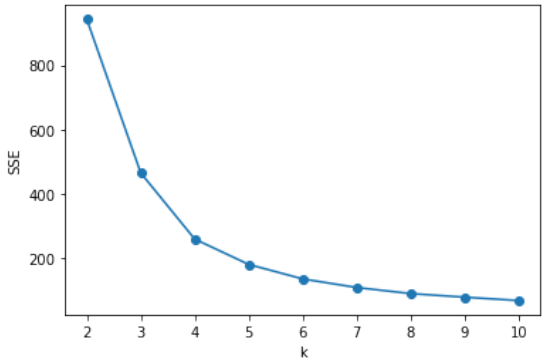


In [25]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

In [26]:
```
#設定不同的分群數, 觀察SSE值, 找出適合的分群數
#模型.inertia_ 記錄分群後的 SSE
SSE = []
for k in range(2, 11):
    est = KMeans(n_clusters = k)
    est.fit(df_minmax)
    SSE.append(est.inertia_)

SSE
```

Out[26]:
```
[943.1063556920799,
 464.78486985131036,
 258.9527506447147,
 180.10469769507122,
 135.07142989780726,
 108.3601385705478,
 89.48352501520392,
 78.22025485890656,
 67.80809473978984]
```

In [27]:
```
# 繪製散佈圖, 找到 Elbow(手肘)點
# 從圖中可以看出 elbow 出現在 4
X = range(2, 11)
plt.xlabel('k')
plt.ylabel('SSE')
plt.plot(X, SSE, 'o-')

plt.show()
```



In [28]:
```
#將每一群客戶資料抽出另外儲存
group0 = r[r['GroupID'] == 0]
group1 = r[r['GroupID'] == 1]
group2 = r[r['GroupID'] == 2]
group3 = r[r['GroupID'] == 3]
```

In [29]:
```
#每一群的基本統計量描述
group0.describe()
```

Out[29]:

	R	F	M	GroupID
count	11837.000000	11837.000000	11837.000000	11837.0
mean	0.358425	0.006233	0.010220	0.0
std	0.073903	0.012515	0.014020	0.0
min	0.232036	0.000000	0.000017	0.0
25%	0.294910	0.000000	0.004080	0.0
50%	0.351796	0.000000	0.005796	0.0
75%	0.423653	0.009009	0.011711	0.0
max	0.494012	0.243243	0.411356	0.0

In [30]:
```
group1.describe()
```

Out[30]:

	R	F	M	GroupID
count	9586.000000	9586.000000	9586.000000	9586.0
mean	0.877445	0.001413	0.006344	1.0
std	0.069899	0.005864	0.006788	0.0
min	0.754491	0.000000	0.000039	1.0
25%	0.820359	0.000000	0.002857	1.0
50%	0.874251	0.000000	0.004895	1.0
75%	0.938623	0.000000	0.007083	1.0
max	1.000000	0.252252	0.184426	1.0

In [31]:
```
group2.describe()
```

Out[31]:

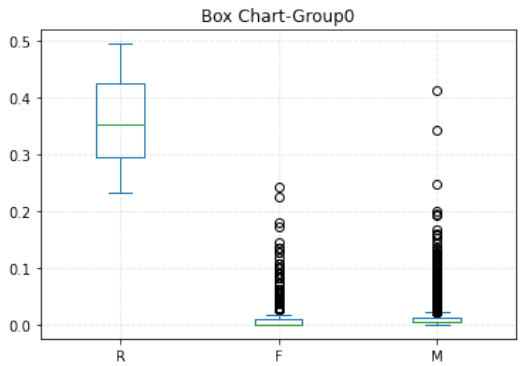
	R	F	M	GroupID
count	12062.000000	12062.000000	12062.000000	12062.0
mean	0.105545	0.015544	0.018302	2.0
std	0.070047	0.033832	0.035288	0.0
min	0.000000	0.000000	0.000000	2.0
25%	0.037425	0.000000	0.004161	2.0
50%	0.100299	0.000000	0.007973	2.0
75%	0.167665	0.018018	0.019346	2.0
max	0.235030	1.000000	1.000000	2.0

In [32]:
```
group3.describe()
```

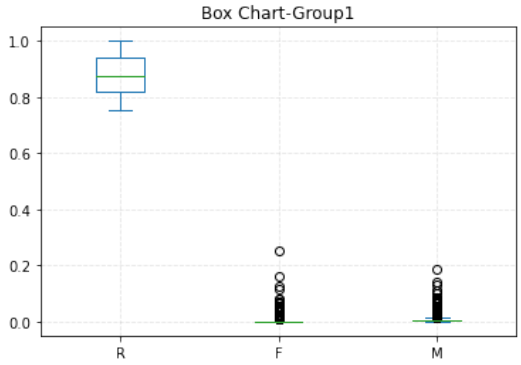
Out[32]:

	R	F	M	GroupID
count	10187.000000	10187.000000	10187.000000	10187.0
mean	0.630258	0.003491	0.007693	3.0
std	0.071988	0.009076	0.009380	0.0
min	0.495509	0.000000	0.000026	3.0
25%	0.573353	0.000000	0.003121	3.0
50%	0.627246	0.000000	0.005367	3.0
75%	0.691617	0.000000	0.008713	3.0
max	0.754491	0.252252	0.199449	3.0

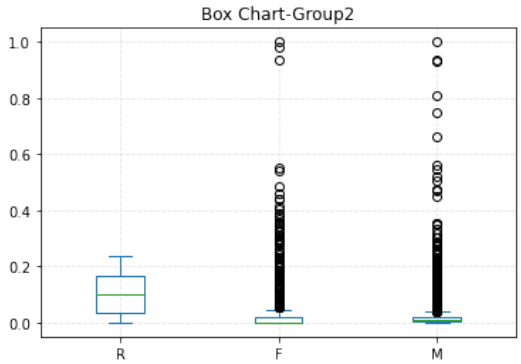
In [33]:
```
#繪製盒鬚圖(箱型圖)
#重點在觀察中位數的高低和離群值(outlier)的大小
#盒鬚圖的頂端至底端分別表示了: 最大值,上四分位數,中位數,下四分位數,最小值。超出圖形的範圍為離群值
group0 = group0[['R', 'F', 'M']]
group0.plot.box(title = "Box Chart-Group0")
plt.grid(linestyle = "--", alpha = 0.3)
```



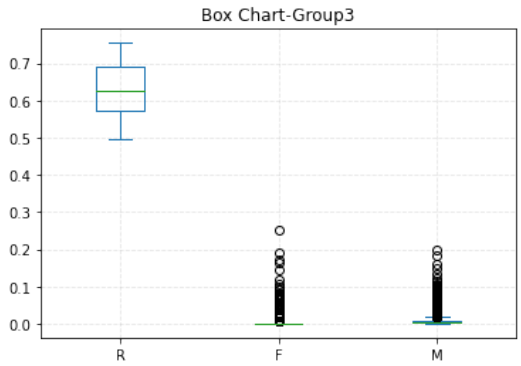
In [34]:
```
group1 = group1[['R', 'F', 'M']]
group1.plot.box(title = "Box Chart-Group1")
plt.grid(linestyle = "--", alpha = 0.3)
```



In [35]:
```
group2 = group2[['R', 'F', 'M']]
group2.plot.box(title = "Box Chart-Group2")
plt.grid(linestyle = "--", alpha = 0.3)
```

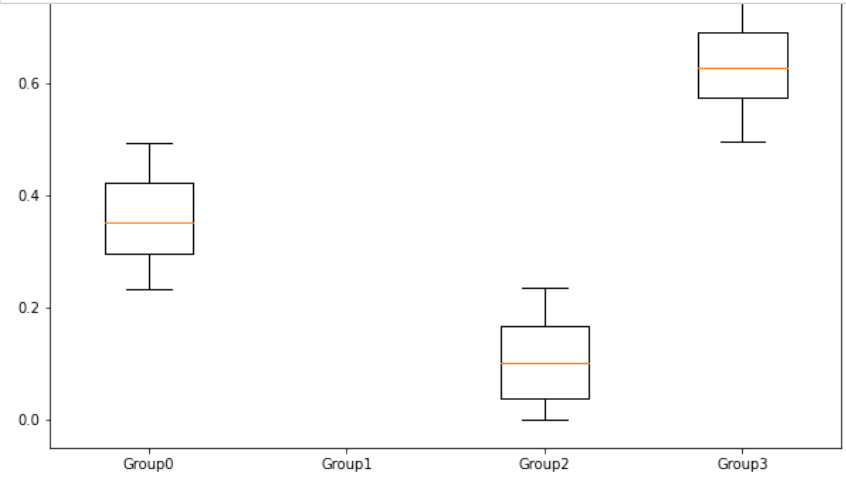


In [36]:
```
group3 = group3[['R', 'F', 'M']]
group3.plot.box(title = "Box Chart-Group3")
plt.grid(linestyle = "--", alpha = 0.3)
```

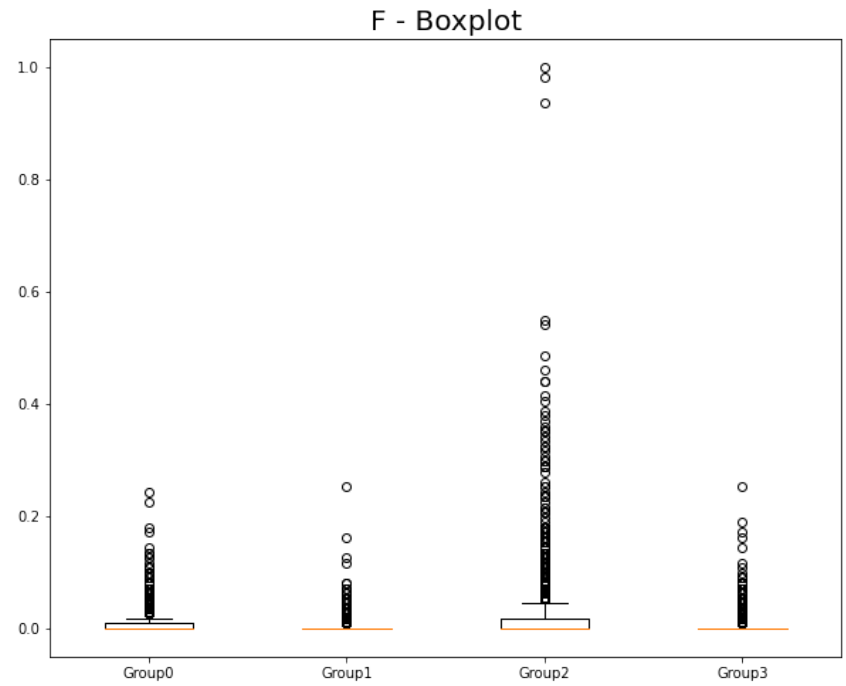


In [37]:
```
#重新編排資料, 改為一次看四個群的同一個欄位值
g0 = r[r['GroupID'] == 0]
g1 = r[r['GroupID'] == 1]
g2 = r[r['GroupID'] == 2]
g3 = r[r['GroupID'] == 3]
```

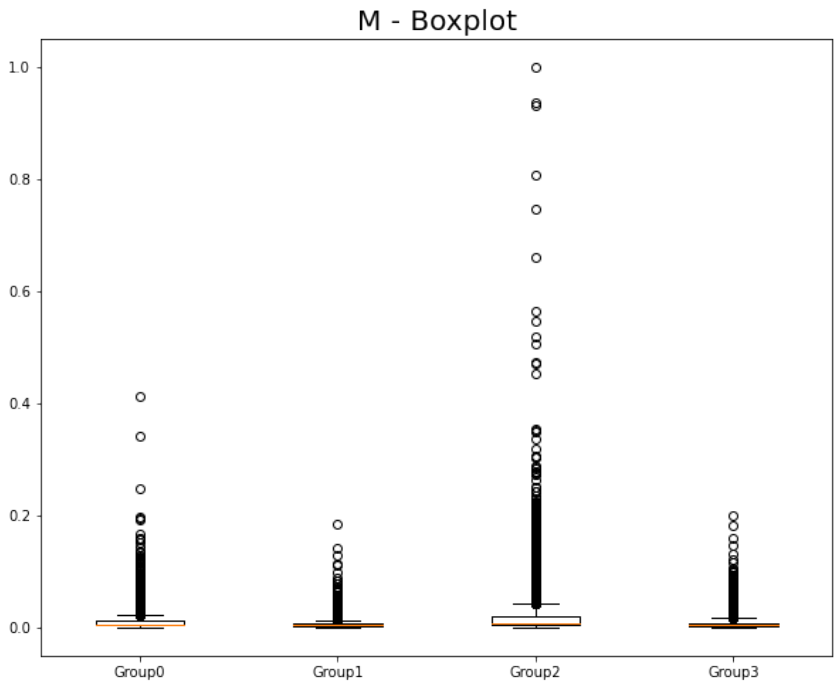
In [38]:
```
# R
plt.figure(figsize = (10, 8))
labels = 'Group0', 'Group1', 'Group2', 'Group3'
plt.boxplot([g0['R'], g1['R'], g2['R'], g3['R']], labels = labels)
plt.title('R - Boxplot', fontsize = 20)

plt.show()
```



In [39]:
```
# F
plt.figure(figsize = (10, 8))
labels = 'Group0', 'Group1', 'Group2', 'Group3'
plt.boxplot([g0['F'], g1['F'], g2['F'], g3['F']], labels = labels)
plt.title('F - Boxplot', fontsize = 20)

plt.show()
```



In [40]:
```
# M
plt.figure(figsize = (10, 8))
labels = 'Group0', 'Group1', 'Group2', 'Group3'
plt.boxplot([g0['M'], g1['M'], g2['M'], g3['M']], labels = labels)
plt.title('M - Boxplot', fontsize = 20)

plt.show()
```



In [ ]: