RFM模型

R：Recency,新进度,代表最近一次消费以来的天数 F：Frequency，消费频率，代表用户是否频繁使用服务，用户黏性的风向标 M：Monetory,消费金额，代表用户一段时间内的消费总金额

In [1]:
```
import pandas as pd # 引入Pandas
import matplotlib.pyplot as plt # 引入matplotlib.pyplot用于画图
import seaborn as sn # 引入seaborn箱线图
from sklearn.model_selection import train_test_split # 导入train_test_split用于分割数
```

In [2]:
```
df_sales = pd.read_csv('./dataset/易速鲜花订单记录.csv')
df_sales.head()
```

Out[2]:

	订单号	产品码	消费日期	产品说明	数量	单价	用户码	城市
0	536374	21258	6/1/2020 9:09	五彩玫瑰五支装	32	10.95	15100	北京
1	536376	22114	6/1/2020 9:32	茉莉花白色25枝	48	3.45	15291	上海
2	536376	21733	6/1/2020 9:32	教师节向日葵3枝尤加利5枝	64	2.55	15291	上海
3	536378	22386	6/1/2020 9:37	百合粉色10花苞	10	1.95	14688	北京
4	536378	85099C	6/1/2020 9:37	橙黄香槟色康乃馨	10	1.95	14688	北京

In [3]:
```
df_sales.describe()
```

Out[3]:

	数量	单价	用户码
count	87180.000000	87180.000000	87180.000000
mean	10.006435	3.584552	15338.503774
std	48.769498	133.436042	392.001142
min	-9360.000000	0.000000	14681.000000
25%	2.000000	1.250000	15022.000000
50%	4.000000	1.950000	15335.000000
75%	12.000000	3.750000	15674.000000
max	3114.000000	38970.000000	16019.000000

In [4]:
```
# 构建月度的订单数的DataFrame
df_sales['消费日期'] = pd.to_datetime(df_sales['消费日期']) # 转换日期格式
df_order_monthly = df_sales.set_index('消费日期')['订单号'].resample('M').nunique() #
# 设定绘图的画布
ax = pd.DataFrame(df_order_monthly.values).plot(grid=True,figsize=(12,6),legend=Fals
ax.set_xlabel('月份') # X轴Label
ax.set_ylabel('订单数') # Y轴Label
ax.set_title('月度订单数') # 图标题
# 设定X轴月份显示格式
plt.xticks(range(len(df_order_monthly.index)),[x.strftime('%m.%Y') for x in df_order
plt.show() #绘图
```

In [46]:
```
df_order_monthly.index
```

Out[46]:
```
DatetimeIndex(['2020-06-30', '2020-07-31', '2020-08-31', '2020-09-30', '2020-10-31',
'2020-11-30', '2020-12-31', '2021-01-31', '2021-02-28', '2021-03-31', '2021-04-30',
               '2021-05-31', '2021-06-30'],
              dtype='datetime64[ns]', name='消费日期', freq='M')
```

In [5]:
```
df_sales.drop_duplicates() # 删除重复的数据行
df_sales.describe()
```

Out[5]:

	数量	单价	用户码
count	87180.000000	87180.000000	87180.000000
mean	10.006435	3.584552	15338.503774
std	48.769498	133.436042	392.001142
min	-9360.000000	0.000000	14681.000000
25%	2.000000	1.250000	15022.000000
50%	4.000000	1.950000	15335.000000
75%	12.000000	3.750000	15674.000000
max	3114.000000	38970.000000	16019.000000

In [6]:
```
df_sales.isna().sum()
```

Out[6]:
```
订单号      0
产品码      0
消费日期     0
产品说明     0
数量       0
单价       0
用户码      0
城市       0
dtype: int64
```

In [7]:
```
df_sales.describe() # df_sales的统计数据
```

Out[7]:

	数量	单价	用户码
count	87180.000000	87180.000000	87180.000000
mean	10.006435	3.584552	15338.503774
std	48.769498	133.436042	392.001142
min	-9360.000000	0.000000	14681.000000
25%	2.000000	1.250000	15022.000000
50%	4.000000	1.950000	15335.000000
75%	12.000000	3.750000	15674.000000
max	3114.000000	38970.000000	16019.000000

In [8]:
```
df_sales = df_sales.loc[df_sales['数量']>0] # 清洗掉数量小于等于0的数据
df_sales.describe()
```

Out[8]:

	数量	单价	用户码
count	85354.000000	85354.000000	85354.000000
mean	10.641833	2.997970	15338.156712
std	33.788964	15.255909	392.798853
min	1.000000	0.000000	14681.000000
25%	2.000000	1.250000	15021.000000
50%	4.000000	1.950000	15334.000000
75%	12.000000	3.750000	15674.000000
max	3114.000000	3155.950000	16019.000000

In [9]:
```
df_sales['总价'] = df_sales['单价'] * df_sales['数量'] #计算每一单的总价
df_sales.head() # 显示头几行数据
```

Out[9]:

	订单号	产品码	消费日期	产品说明	数量	单价	用户码	城市	总价
0	536374	21258	2020-06-01 09:09:00	五彩玫瑰五支装	32	10.95	15100	北京	350.4
1	536376	22114	2020-06-01 09:32:00	茉莉花白色25枝	48	3.45	15291	上海	165.6
2	536376	21733	2020-06-01 09:32:00	教师节向日葵3枝尤加利5枝	64	2.55	15291	上海	163.2
3	536378	22386	2020-06-01 09:37:00	百合粉色10花苞	10	1.95	14688	北京	19.5
4	536378	85099C	2020-06-01 09:37:00	橙黄香槟色康乃馨	10	1.95	14688	北京	19.5

In [10]:
```
df_user = pd.DataFrame(df_sales['用户码'].unique()) # 生成以用户码为主键的结构df_user
df_user.columns = ['用户码'] #设定字段名
df_user = df_user.sort_values(by='用户码',ascending=True).reset_index(drop=True) #按
df_user # 显示df_user
```

Out[10]:

	用户码
0	14681
1	14682
2	14684
3	14687
4	14688
...	...
975	16015
976	16016
977	16017
978	16018
979	16019

980 rows × 1 columns

In [11]:
```
df_sales['消费日期'] = pd.to_datetime(df_sales['消费日期']) # 转化日期格式
df_recent_buy = df_sales.groupby('用户码').消费日期.max().reset_index() #构建消费日期信
df_recent_buy.columns = ['用户码','最近日期'] # 设定字段名
df_recent_buy['R值'] = (df_recent_buy['最近日期'].max()-df_recent_buy['最近日期']).dt.
df_user = pd.merge(df_user,df_recent_buy[['用户码','R值']],on='用户码') #把上次消费距最
df_user.head()
```

Out[11]:

	用户码	R值
0	14681	70
1	14682	187
2	14684	25
3	14687	106
4	14688	7

In [12]:
```
df_frequency = df_sales.groupby('用户码').消费日期.count().reset_index() # 计算每个用户
df_frequency.columns = ['用户码','F值'] # 设置字段名
df_user = pd.merge(df_user,df_frequency[['用户码','F值']],on='用户码') # 将消费频率整合
df_user.head()
```

Out[12]:

	用户码	R值	F值
0	14681	70	7
1	14682	187	2
2	14684	25	421
3	14687	106	15
4	14688	7	327

In [13]:
```
df_revenue = df_sales.groupby('用户码').总价.sum().reset_index() #根据消费总额，构建df_
df_revenue.columns = ['用户码','M值'] # 设定字段名
df_user = pd.merge(df_user,df_revenue[['用户码','M值']],on='用户码') # 将消费总额合并到
df_user.head()
```

Out[13]:

	用户码	R值	F值	M值
0	14681	70	7	498.95
1	14682	187	2	52.00
2	14684	25	421	1236.28
3	14687	106	15	628.38
4	14688	7	327	5630.87

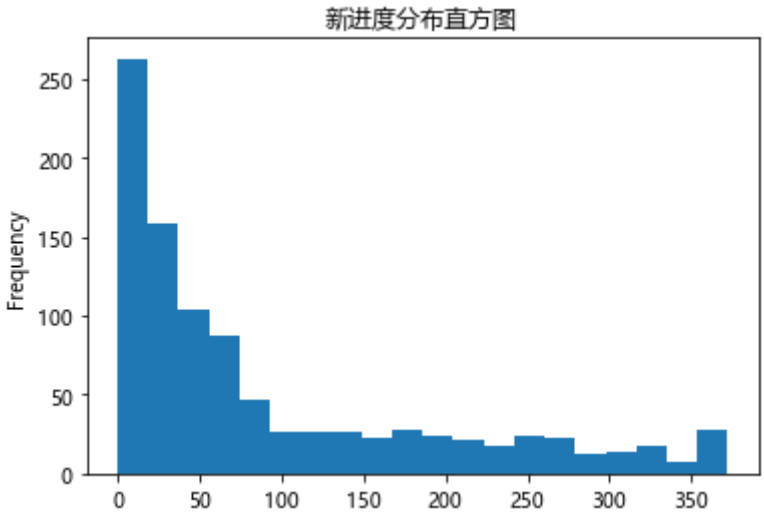
聚类分析：使用RFM给电商用户做价值组画像

R值、F值和M值直方图

注意：直方图中的bins参数表示直方图分为多少个离散组

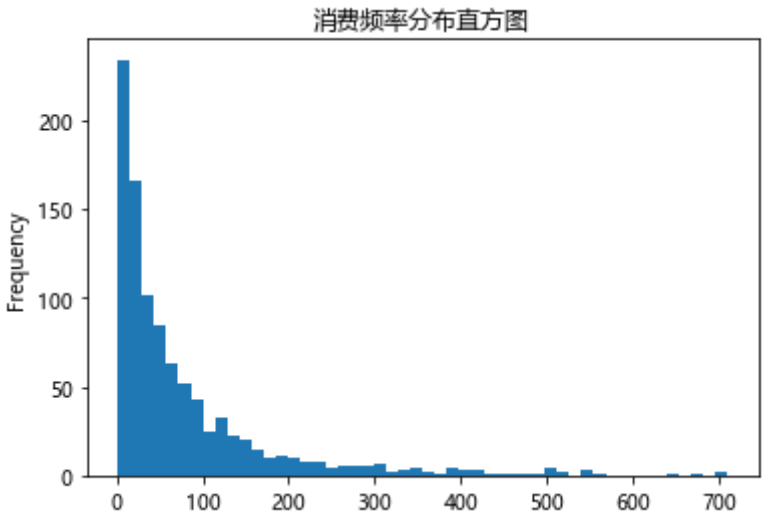
In [14]:
```
df_user['R值'].plot(kind='hist',bins=20,title='新进度分布直方图') #R值直方图
```

Out[14]:
```
<AxesSubplot:title={'center':'新进度分布直方图'}, ylabel='Frequency'>
```



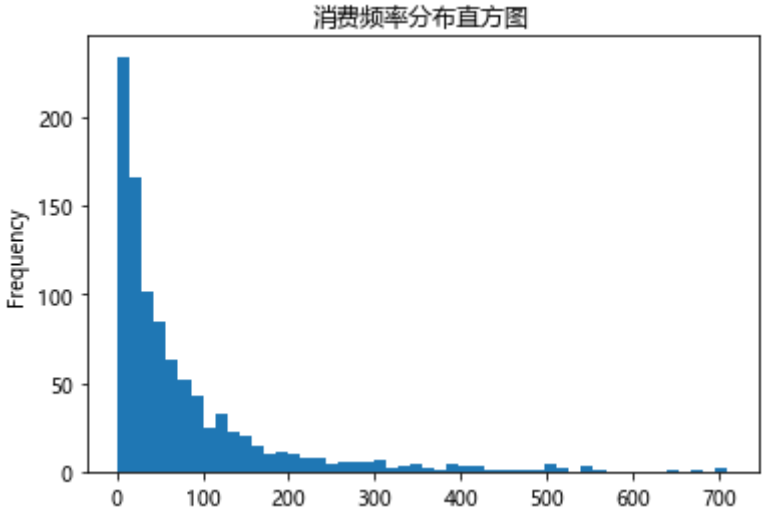
In [15]:
```
df_user[df_user['F值']<800]['F值'].plot(kind='hist',bins=50,title='消费频率分布直方图'
```

Out[15]:
```
<AxesSubplot:title={'center':'消费频率分布直方图'}, ylabel='Frequency'>
```



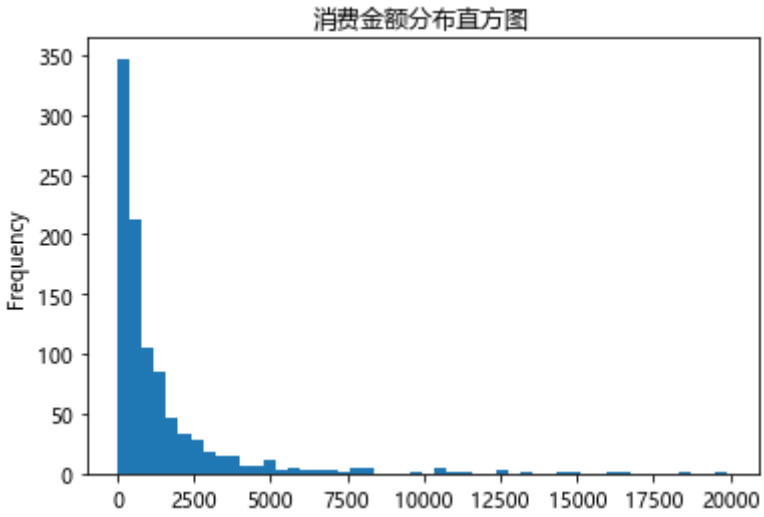
In [16]:
```
df_user.query('F值 < 800')['F值'].plot(kind='hist',bins=50,title='消费频率分布直方图')
```

Out[16]:
```
<AxesSubplot:title={'center':'消费频率分布直方图'}, ylabel='Frequency'>
```



In [17]:
```
df_user.query('M值 < 20000')['M值'].plot(kind='hist',bins=50,title='消费金额分布直方图
```

Out[17]:
```
<AxesSubplot:title={'center':'消费金额分布直方图'}, ylabel='Frequency'>
```



手肘法确定K值

In [18]:
```
from sklearn.cluster import KMeans # 导入KMeans模块
def show_elbow(df): # 定义手肘函数
    distance_list = [] #各个数据点到质心的距离（损失）
    K = range(1,10) #左闭右开
    for k in K:
        kmeans = KMeans(n_clusters=k,max_iter=20) # n_clusters表示簇的个数，max_iter每
        kmeans = kmeans.fit(df) #拟合模型
        distance_list.append(kmeans.inertia_) # intertia_表示每个数据点到质心的距离的平
    plt.plot(K,distance_list,'bx-') #绘图
    plt.xlabel('K值')
    plt.ylabel('距离均方误差')
    plt.title('K值手肘图') #标题
```

In [19]:
```
df_user.head()
```

Out[19]:

	用户码	R值	F值	M值
0	14681	70	7	498.95

	用户编码	R值	F值	M值
1	14683	1	1	5...
2	14684	25	421	1236.28
3	14687	106	15	628.38
4	14688	7	327	5630.87

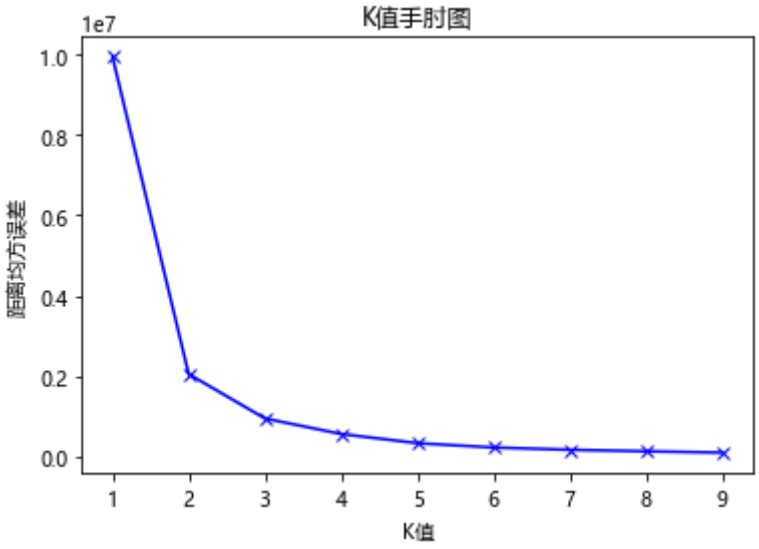
In [20]:
```
df_user.loc[:,['R值']]
```

Out[20]:

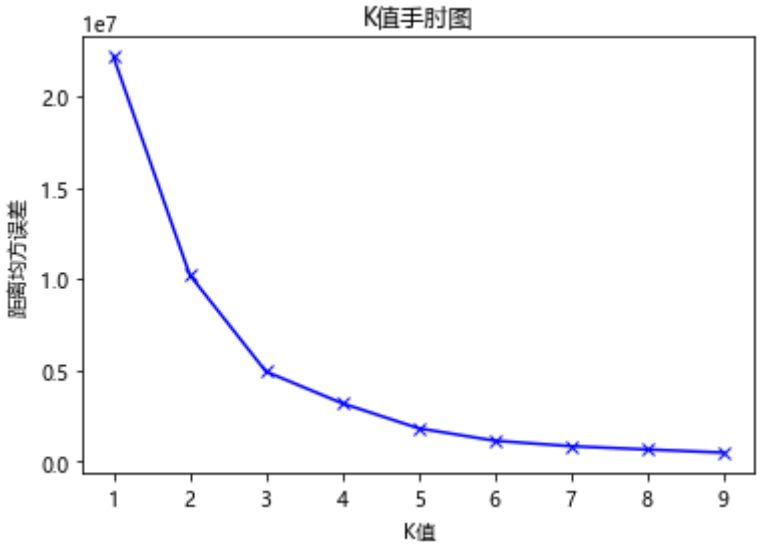
	R值
0	70
1	187
2	25
3	106
4	7
...	...
975	3
976	2
977	46
978	38
979	46

980 rows × 1 columns

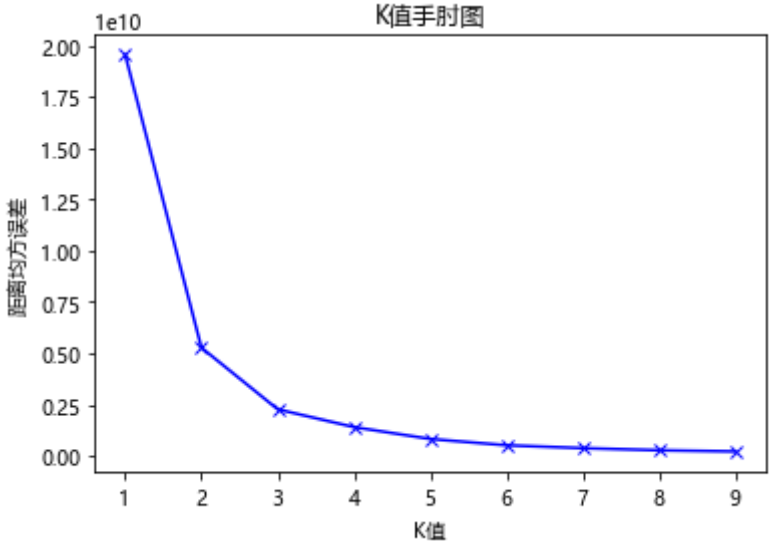
In [21]:
```
show_elbow(df_user.loc[:,['R值']]) #画R值手肘图
```



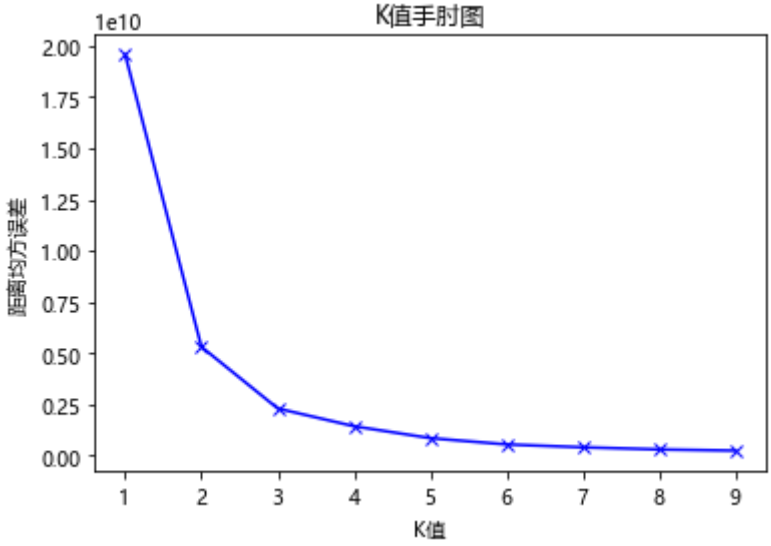
In [22]:
```
show_elbow(df_user.loc[:,['F值']]) #F值手肘图
```



In [23]:
```
show_elbow(df_user.loc[:,['M值']])# M值手肘图
```



In [24]:
```
show_elbow(df_user.loc[:,['R值','F值','M值']]) #为三个特征同时做聚类
```



R、F、M值分别的拐点大概都在3到5之间，即我们把用户分为3、4、5个组都可以； 这里我们选择：

- R值的簇的个数：3

- F值的簇的个数：4
- R值的簇的个数：5

R、F、M值同时做聚类，拐点在3到5之间，我们选择簇的个数为：4

创建和训练KMeans模型

In [25]:
```
from sklearn.cluster import KMeans # 导入KMeans模块
kmeans_R = KMeans(n_clusters=3)
kmeans_F = KMeans(n_clusters=4)
kmeans_M = KMeans(n_clusters=5)
kmeans_RFM = KMeans(n_clusters=4)
```

In [26]:
```
kmeans_R.fit(df_user.loc[:,['R值']]) #拟合模型
kmeans_F.fit(df_user.loc[:,['F值']]) #拟合模型
kmeans_M.fit(df_user.loc[:,['M值']]) #拟合模型
kmeans_RFM.fit(df_user.loc[:,['R值','F值','M值']]) #拟合模型
```

Out[26]:
```
KMeans
KMeans(n_clusters=4)
```

使用模型进行聚类，并给用户分组

In [27]:
```
df_user['RFM值层级'] = kmeans_RFM.predict(df_user.loc[:,['R值','F值','M值']]) #通过聚
df_user.head()
```

Out[27]:

	用户码	R值	F值	M值	RFM值层级
0	14681	70	7	498.95	0
1	14682	187	2	52.00	0
2	14684	25	421	1236.28	0
3	14687	106	15	628.38	0
4	14688	7	327	5630.87	2

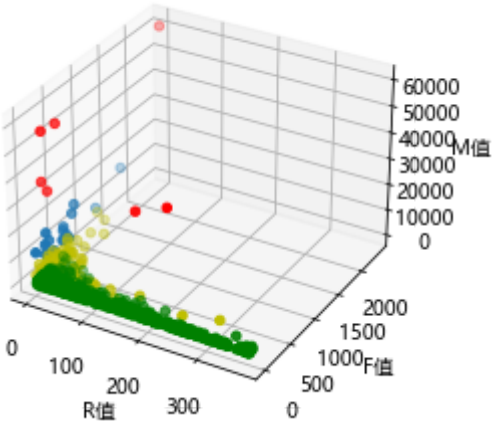
In [28]:
```
df_user.groupby('RFM值层级')[['R值','F值','M值']].describe() # RFM值层级分组之后RFM值的
```

Out[28]:

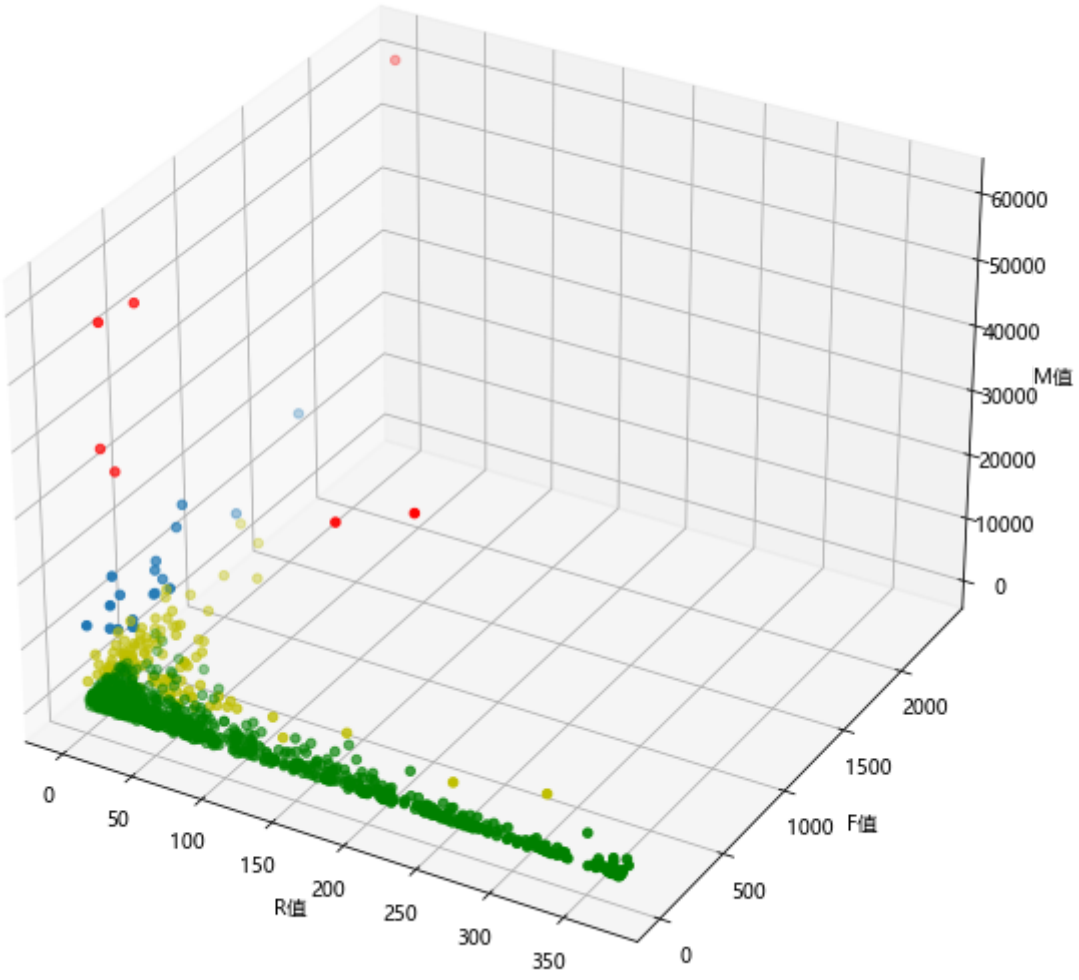
	R值												
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	
RFM值层级													
0	834.0	103.143885	103.808895	0.0	23.0	59.0	168.75	372.0	834.0	52.616307	...	69.0	
1	7.0	62.428571	100.324236	0.0	3.0	6.0	95.50	234.0	7.0	461.571429	...	285.0	2
2	120.0	31.658333	44.140246	0.0	7.0	17.0	42.00	305.0	120.0	252.458333	...	324.0	1
3	19.0	8.473684	8.565878	0.0	2.0	7.0	12.00	32.0	19.0	418.210526	...	505.0	1

4 rows × 24 columns

In [29]:
```
ax = plt.subplot(111,projection='3d') # 3D图  111表示第一个‘1’是子图的行数为，第二个‘1’
ax.scatter(df_user.query('RFM值层级 == 0')['R值'],df_user.query('RFM值层级 == 0')['F值
ax.scatter(df_user.query('RFM值层级 == 1')['R值'],df_user.query('RFM值层级 == 1')['F值
ax.scatter(df_user.query('RFM值层级 == 2')['R值'],df_user.query('RFM值层级 == 2')['F值
ax.scatter(df_user.query('RFM值层级 == 3')['R值'],df_user.query('RFM值层级 == 3')['F值
ax.set_xlabel('R值')
ax.set_ylabel('F值')
ax.set_zlabel('M值')
plt.show()
```



In [30]:
```
fig=plt.figure(figsize=(10,10)) #设置图表空间大小，单位英寸
ax = fig.add_subplot(111,projection='3d')
ax.scatter(df_user.query('RFM值层级 == 0')['R值'],df_user.query('RFM值层级 == 0')['F值
ax.scatter(df_user.query('RFM值层级 == 1')['R值'],df_user.query('RFM值层级 == 1')['F值
ax.scatter(df_user.query('RFM值层级 == 2')['R值'],df_user.query('RFM值层级 == 2')['F值
ax.scatter(df_user.query('RFM值层级 == 3')['R值'],df_user.query('RFM值层级 == 3')['F值
ax.set_xlabel('R值')
ax.set_ylabel('F值')
ax.set_zlabel('M值')
plt.show()
```



In [31]:
```
df_user['R值层级'] = kmeans_R.predict(df_user.loc[:,['R值']]) #通过聚类模型求出R值的层
df_user.head()
```

Out[31]:

	用户码	R值	F值	M值	RFM值层级	R值层级
0	14681	70	7	498.95	0	0
1	14682	187	2	52.00	0	2
2	14684	25	421	1236.28	0	0
3	14687	106	15	628.38	0	2
4	14688	7	327	5630.87	2	0

In [32]:
```
df_user.groupby('R值层级')['R值'].describe() # R值层级分组之后R值的统计信息
```

Out[32]:

R值层级	count	mean	std	min	25%	50%	75%	max
0	664.0	32.088855	25.141763	0.0	10.00	25.0	50.00	94.0
1	138.0	298.094203	45.436550	231.0	255.25	292.5	334.50	372.0
2	178.0	157.162921	37.340870	95.0	126.00	156.5	188.75	225.0

In [33]:
```
df_user['F值层级'] = kmeans_F.predict(df_user.loc[:,['F值']]) # 通过聚类模型求F值的层级
df_user.head()
```

Out[33]:

	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级
0	14681	70	7	498.95	0	0	0
1	14682	187	2	52.00	0	2	0
2	14684	25	421	1236.28	0	0	1
3	14687	106	15	628.38	0	2	0
4	14688	7	327	5630.87	2	0	1

In [34]:
```
df_user.groupby('F值层级')['F值'].describe() # F值层级分组之后F值的统计信息
```

Out[34]:

F值层级	count	mean	std	min	25%	50%	75%	max
0	751.0	35.288948	27.355251	1.0	12.0	28.0	53.0	103.0
1	43.0	444.279070	106.556787	310.0	356.0	415.0	508.0	710.0
2	7.0	1295.285714	516.333456	899.0	1013.5	1119.0	1321.5	2379.0
3	179.0	171.402235	55.350418	104.0	127.0	155.0	207.0	308.0

In [35]:
```
df_user['M值层级'] = kmeans_M.predict(df_user.loc[:,['M值']]) #通过聚类模型求M值的层级
df_user.head()
```

Out[35]:

	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级	M值层级
0	14681	70	7	498.95	0	0	0	0
1	14682	187	2	52.00	0	2	0	0
2	14684	25	421	1236.28	0	0	1	0
3	14687	106	15	628.38	0	2	0	0
4	14688	7	327	5630.87	2	0	1	2

In [36]:
```
df_user.groupby('M值层级')['M值'].describe() # M值层级分组之后M值的统计信息
```

Out[36]:

M值层级	count	mean	std	min	25%	50%	75%	max
0	834.0	706.773934	580.280535	6.20	263.8175	499.135	1044.0925	2446.60
1	3.0	57184.920000	3219.734238	54534.14	55393.4300	56252.720	58510.3100	60767.90
2	120.0	4192.362333	1576.253234	2465.48	2878.3250	3676.415	4971.8950	8347.20
3	4.0	38806.120000	4601.062891	33643.08	36258.7200	38523.550	41070.9500	44534.30
4	19.0	12945.388421	3023.220531	9585.91	10579.8700	12393.700	14574.0600	19914.44

为聚类的层级做排序*

这里为聚类的层级排序思路： 就是将聚类之后没有顺序概念的排个顺序，比如之前的索引0、1、2分组的值之间是没有顺序大小之分的，现在我们可以把它们按照分组的平均值大小进行排序，然后得到新的分组排序，新的分组里面新的索引比如0、1、2是有顺序的，要么从大到小，要么从小到大，用新分组的索引作为层级的排序； 说白了，就是比如原来分组1最大、分组2次之、分组0最小，现在我们要把排序，即（如从大到小）分组0最大、分组1次之、分组2最小；

In [37]:
```
def order_cluster(cluster_name, target_name, df, ascending=False):
    pd.set_option('display.unicode.ambiguous_as_wide', True)
    pd.set_option('display.unicode.east_asian_width', True)
    pd.set_option('display.width', 180)
    new_cluster_name = 'new_' + cluster_name # 新的聚类名称
    df_new = df.groupby(cluster_name)[target_name].mean().reset_index() #按聚类结果分
    print('df_new_1:')
    print(df_new.head())
    df_new = df_new.sort_values(by=target_name,ascending=ascending).reset_index(drop
    print('df_new_2:')
    print(df_new.head())
    df_new['index'] = df_new.index # 创建索引字段
    df_new = pd.merge(df,df_new.loc[:,[cluster_name,'index']],on=cluster_name) #基于
    print('df_new_3:')
    print(df_new.head())
    df_new = df_new.drop([cluster_name],axis=1) # 删除聚类名称，axis=1表示横坐标
    print('df_new_4:')
    print(df_new.head())
    df_new = df_new.rename(columns={"index":cluster_name}) #将索引字段重命名为聚类名称
    print('df_new_5:')
    print(df_new.head())
    return df_new # 返回排序后的df_new对象
```

df.reset():用于重置索引或其level 注意:df.reset()和df.reset(drop=True)的区别

- df.reset()：表示会把旧的索引列添加到列，并用新的顺序索引（整数索引）
- df.reset(drop=True)：表示不会把旧索引列添加为列
  - drop的作用：为True时不要把索引添加到DataFrame的列中，默认为False。这个会重置索引为默认的整数索引。

参考： df.reset()的用法

In [38]:
```
df_user = order_cluster('R值层级','R值',df_user,False) # 调用簇排序函数（降序，R值越大，
df_user = df_user.sort_values(by='用户码',ascending=True).reset_index(drop=True)
df_user.head()
```

```
df_new_1:
   R值层级         R值
0        0   32.088855
1        1  298.094203
2        2  157.162921
df_new_2:
   R值层级         R值
0        1  298.094203
1        2  157.162921
2        0   32.088855
df_new_3:
   用户码  R值  F值      M值  RFM值层级  R值层级  F值层级  M值层级  index
0  14681   70    7   498.95          0        0        0        0      2
1  14684   25  421  1236.28          0        0        1        0      2
2  14688    7  327  5630.87          2        0        1        2      2
3  14690   43   50   371.70          0        0        0        0      2
4  14691   30   71  2114.33          0        0        0        0      2
df_new_4:
   用户码  R值  F值      M值  RFM值层级  F值层级  M值层级  index
```

```
0   14681   70    7   498.95          0        0        0       2
1   14684   25  421  1236.28          0        1        0       2
2   14688    7  327  5630.87          2        1        2       2
3   14690   43   50   371.70          0        0        0       2
4   14691   30   71  2114.33          0        0        0       2
df_new_5:
   用户码  R值  F值       M值  RFM值层级  F值层级  M值层级  R值层级
0   14681   70    7   498.95          0        0        0        2
1   14684   25  421  1236.28          0        1        0        2
2   14688    7  327  5630.87          2        1        2        2
3   14690   43   50   371.70          0        0        0        2
4   14691   30   71  2114.33          0        0        0        2
```

Out[38]:

	用户码	R值	F值	M值	RFM值层级	F值层级	M值层级	R值层级
0	14681	70	7	498.95	0	0	0	2
1	14682	187	2	52.00	0	0	0	1
2	14684	25	421	1236.28	0	1	0	2
3	14687	106	15	628.38	0	0	0	1
4	14688	7	327	5630.87	2	1	2	2

In [39]:

```
df_user = order_cluster('F值层级','F值',df_user,True) # 为F值层级排序（升序，频率越高，
df_user.sort_values(by='用户码',ascending=True).reset_index()
df_user.head()
```

```
df_new_1:
   F值层级          F值
0        0    35.288948
1        1   444.279070
2        2  1295.285714
3        3   171.402235
df_new_2:
   F值层级          F值
0        0    35.288948
1        3   171.402235
2        1   444.279070
3        2  1295.285714
df_new_3:
   用户码  R值  F值      M值  RFM值层级  F值层级  M值层级  R值层级  index
0   14681   70    7  498.95          0        0        0        2      0
1   14682  187    2   52.00          0        0        0        1      0
2   14687  106   15  628.38          0        0        0        1      0
3   14689  208   13  112.80          0        0        0        1      0
4   14690   43   50  371.70          0        0        0        2      0
df_new_4:
   用户码  R值  F值      M值  RFM值层级  M值层级  R值层级  index
0   14681   70    7  498.95          0        0        2      0
1   14682  187    2   52.00          0        0        1      0
2   14687  106   15  628.38          0        0        1      0
3   14689  208   13  112.80          0        0        1      0
4   14690   43   50  371.70          0        0        2      0
df_new_5:
   用户码  R值  F值      M值  RFM值层级  M值层级  R值层级  F值层级
0   14681   70    7  498.95          0        0        2        0
1   14682  187    2   52.00          0        0        1        0
2   14687  106   15  628.38          0        0        1        0
3   14689  208   13  112.80          0        0        1        0
4   14690   43   50  371.70          0        0        2        0
```

Out[39]:

	用户码	R值	F值	M值	RFM值层级	M值层级	R值层级	F值层级
0	14681	70	7	498.95	0	0	2	0
1	14682	187	2	52.00	0	0	1	0

	用户码	R值	F值	M值	RFM值层级	M值层级	R值层级	F值层级
2	14687	106	15	628.38	0	0	1	0
3	14689	208	13	112.80	0	0	1	0
4	14690	43	50	371.70	0	0	2	0

In [40]:
```
df_user = order_cluster('M值层级','M值',df_user,True) #调用聚类排序函数(升序，M值越高，
df_user.sort_values(by='用户码',ascending=True).reset_index() #按用户码排序
df_user.head()
```

```
df_new_1:
   M值层级           M值
0        0    706.773934
1        1  57184.920000
2        2   4192.362333
3        3  38806.120000
4        4  12945.388421
df_new_2:
   M值层级           M值
0        0    706.773934
1        2   4192.362333
2        4  12945.388421
3        3  38806.120000
4        1  57184.920000
df_new_3:
   用户码  R值  F值     M值  RFM值层级  M值层级  R值层级  F值层级  index
0   14681   70    7  498.95          0        0        2        0      0
1   14682  187    2   52.00          0        0        1        0      0
2   14687  106   15  628.38          0        0        1        0      0
3   14689  208   13  112.80          0        0        1        0      0
4   14690   43   50  371.70          0        0        2        0      0
df_new_4:
   用户码  R值  F值     M值  RFM值层级  R值层级  F值层级  index
0   14681   70    7  498.95          0        2        0      0
1   14682  187    2   52.00          0        1        0      0
2   14687  106   15  628.38          0        1        0      0
3   14689  208   13  112.80          0        1        0      0
4   14690   43   50  371.70          0        2        0      0
df_new_5:
   用户码  R值  F值     M值  RFM值层级  R值层级  F值层级  M值层级
0   14681   70    7  498.95          0        2        0        0
1   14682  187    2   52.00          0        1        0        0
2   14687  106   15  628.38          0        1        0        0
3   14689  208   13  112.80          0        1        0        0
4   14690   43   50  371.70          0        2        0        0
```

Out[40]:

	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级	M值层级
0	14681	70	7	498.95	0	2	0	0
1	14682	187	2	52.00	0	1	0	0
2	14687	106	15	628.38	0	1	0	0
3	14689	208	13	112.80	0	1	0	0
4	14690	43	50	371.70	0	2	0	0

为用户整体分组画像

In [41]:
```
df_user['总分'] = df_user['R值层级'] +df_user['F值层级'] + df_user['M值层级']
df_user.head()
```

Out[41]:

	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级	M值层级	总分
0	14681	70	7	498.95	0	2	0	0	2
1	14682	187	2	52.00	0	1	0	0	1
2	14687	106	15	628.38	0	1	0	0	1
3	14689	208	13	112.80	0	1	0	0	1
4	14690	43	50	371.70	0	2	0	0	2

In [42]:
```
df_user['总分'].describe()
```

Out[42]:
```
count    980.000000
mean       2.014286
std        1.352862
min        0.000000
25%        1.000000
50%        2.000000
75%        2.000000
max        9.000000
Name: 总分, dtype: float64
```

In [43]:
```
df_user.loc[(df_user['总分'] <= 2) & (df_user['总分'] >= 0),'总体价值'] = '低价值'
df_user.loc[(df_user['总分'] <= 5) & (df_user['总分'] >= 3),'总体价值'] = '中价值'
df_user.loc[(df_user['总分'] <= 9) & (df_user['总分'] >= 6),'总体价值'] = '高价值'
df_user
```

Out[43]:

	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级	M值层级	总分	总体价值
0	14681	70	7	498.95	0	2	0	0	2	低价值
1	14682	187	2	52.00	0	1	0	0	1	低价值
2	14687	106	15	628.38	0	1	0	0	1	低价值
3	14689	208	13	112.80	0	1	0	0	1	低价值
4	14690	43	50	371.70	0	2	0	0	2	低价值
...	...	...	...	...	...	...	...	...	...	...
975	15838	10	167	33643.08	1	2	1	3	6	高价值
976	16013	3	139	37130.60	1	2	1	3	6	高价值
977	15061	3	403	54534.14	1	2	2	4	8	高价值
978	15769	6	130	56252.72	1	2	1	4	7	高价值
979	15311	0	2379	60767.90	1	2	3	4	9	高价值

980 rows × 10 columns

In [44]:
```
df_user.groupby('总体价值').count()
```

Out[44]:

总体价值	用户码	R值	F值	M值	RFM值层级	R值层级	F值层级	M值层级	总分
中价值	222	222	222	222	222	222	222	222	222
低价值	740	740	740	740	740	740	740	740	740
高价值	18	18	18	18	18	18	18	18	18

In [45]:
```
plt.scatter(df_user.query('总体价值=="高价值"')['F值'],df_user.query('总体价值=="高价值
plt.scatter(df_user.query('总体价值=="中价值"')['F值'],df_user.query('总体价值=="中价值
plt.scatter(df_user.query('总体价值=="低价值"')['F值'],df_user.query('总体价值=="低价值
plt.xlabel('F值')
plt.ylabel('M值')
```

Out[45]:
```
Text(0, 0.5, 'M值')
```