In [1]:
```
import numpy as np
import pandas as pd
#-------------对数据集进行预处理-------------
#训练数据集
train = pd.read_csv('E:\\文档\\A-数据挖掘技术与应用\\实验五\\data\\train.csv')
#测试数据集
test = pd.read_csv('E:\\文档\\A-数据挖掘技术与应用\\实验五\\data\\test.csv')
# 将训练数据与测试数据连接起来，以便一起进行数据清洗
#这里的ignore_index=True是为了保证连接之后，数据仍是连续的
full=pd.concat([train,test],ignore_index=True)
#查看数据结构
print('训练数据集：',train.shape,'测试数据集：',test.shape)
```

```
训练数据集： (891, 12) 测试数据集： (418, 11)
```

In [2]:
```
print('合并之后的数据集：',full.shape)
```

```
合并之后的数据集： (1309, 12)
```

In [3]:
```
#查看合并后的数据，返回前五行
print(full.head())
```

```
   PassengerId  Survived  Pclass  \
0            1       0.0       3
1            2       1.0       1
2            3       1.0       3
3            4       1.0       1
4            5       0.0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [4]:
```
#查看数据类型列的描述统计信息
#count数据总数
#mean平均值
#std标准值
#min最小值
#25%下四份位数
#50%中位数
#75%上四分位数
#max最大值
print(full.describe())
```

```
       PassengerId     Survived       Pclass          Age        SibSp  \
count  1309.000000  891.000000  1309.000000  1046.000000  1309.000000
mean    655.000000    0.383838     2.294882    29.881138     0.498854
std     378.020061    0.486592     0.837836    14.413493     1.041658
min       1.000000    0.000000     1.000000     0.170000     0.000000
25%     328.000000    0.000000     2.000000    21.000000     0.000000
50%     655.000000    0.000000     3.000000    28.000000     0.000000
75%     982.000000    1.000000     3.000000    39.000000     1.000000
max    1309.000000    1.000000     3.000000    80.000000     8.000000

             Parch         Fare
count  1309.000000  1308.000000
mean      0.385027    33.295479
std       0.865560    51.758668
min       0.000000     0.000000
25%       0.000000     7.895800
50%       0.000000    14.454200
75%       0.000000    31.275000
max       9.000000   512.329200
```

In [5]:

```
#查看数据集的缺失情况，进行数据预处理
print(full.isnull().sum())
```

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

In [6]:

```
#对年龄Age和船票价格Fare采用平均值填充
#年龄（Age）
full['Age']=full['Age'].fillna(full['Age'].mean())
#船票价格（Fare）
full['Fare']=full['Fare'].fillna(full['Fare'].mean())
```

In [7]:

```
#输出填充之后的数据集缺失情况，还剩登船港口和客舱号需要处理
print(full.isnull().sum())
```

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age               0
SibSp             0
Parch             0
Ticket            0
Fare              0
Cabin          1014
Embarked          2
dtype: int64
```

In [8]:

```
#对登船港口（Embarked）是分类变量：用最常见的那个类别（众数）来填充
```

```
#用来统计各个类别的总数
print(full['Embarked'].value_counts())
#统计登船港口的众数并输出
print("众数是："+full.Embarked.mode())
```

```
S    914
C    270
Q    123
Name: Embarked, dtype: int64
0    众数是：S
dtype: object
```

In [9]:

```
#这里S是众数，所以采用众数填充
full['Embarked']=full['Embarked'].fillna('S')
```

In [10]:

```
#因为客舱号不能用特征值填充，所以使用Unknown标记，简记为“U”
full['Cabin']=full['Cabin'].fillna('U')
```

In [11]:

```
#输出填充之后的数据集缺失情况
print(full.isnull().sum())
```

```
PassengerId      0
Survived       418
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin            0
Embarked         0
dtype: int64
```

In [12]:

```
#根据结果显示，除了需要预测的survived还有空值外，其余数据所有的缺失值已经被处理完毕
#接下来进行特征提取，首先展示所有的数据信息
print(full.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     891 non-null    float64
 2   Pclass       1309 non-null   int64
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   object
 5   Age          1309 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   int64
 8   Ticket       1309 non-null   object
 9   Fare         1309 non-null   float64
 10  Cabin        1309 non-null   object
 11  Embarked     1309 non-null   object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
None
```

In [13]:
```
#-------------下面进行特征提取------------

#对于有直接类别可以分类的数据，如sex（性别）、Embarked（登船港口）、Pclass（客舱等级
#对性别进行特征提取
#male对应数值1，female对应数值2，存储在sex_mapDict中
sex_mapDict={'male':1,'female':2}
#map函数：对Series每个数据应用自定义的函数计算，将上面的规律存储到数据集中
full['Sex']=full['Sex'].map(sex_mapDict)
#打印输出前五条数据，查看sex的处理情况
print(full.head())
```

```
   PassengerId  Survived  Pclass  \
0            1       0.0       3
1            2       1.0       1
2            3       1.0       3
3            4       1.0       1
4            5       0.0       3

                                                Name  Sex   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris    1  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th...    2  38.0      1      0
2                             Heikkinen, Miss. Laina    2  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    2  35.0      1      0
4                           Allen, Mr. William Henry    1  35.0      0      0

             Ticket     Fare Cabin Embarked
0         A/5 21171   7.2500     U        S
1          PC 17599  71.2833   C85        C
2  STON/O2. 3101282   7.9250     U        S
3            113803  53.1000  C123        S
4            373450   8.0500     U        S
```

In [14]:
```
#对登船港口进行特征提取

#首先查看该类数据内容
print(full['Embarked'].head())
```

```
0    S
1    C
2    S
3    S
4    S
Name: Embarked, dtype: object
```

In [15]:
```
#存放提取后的特征
embarkedDf= pd.DataFrame()
#使用get_dummies进行one-hot编码，产生虚拟变量（dummy variables），列名前缀是Embarke
#get_dummies函数的作用是将该列出现值的类型再重新划分为子列，通过子列里的bool类型标识
embarkedDf = pd.get_dummies( full['Embarked'] , prefix='Embarked' )
#打印输出前五条数据，查看Embarked的处理情况
print(embarkedDf.head())
```

```
   Embarked_C  Embarked_Q  Embarked_S
0           0           0           1
1           1           0           0
2           0           0           1
3           0           0           1
4           0           0           1
```

In [16]:
```
#将Embarked替换成embarkedDF
#删除Embarked
full.drop('Embarked',axis=1,inplace=True)
```

```
#通过contact将新增的列加到原数据表中，作为原来数据的替换
full = pd.concat([full,embarkedDf],axis=1)
#展示处理后的结果，显示前五条数据
print(full.head())
```

```
   PassengerId  Survived  Pclass  \
0            1       0.0       3
1            2       1.0       1
2            3       1.0       3
3            4       1.0       1
4            5       0.0       3

                                                Name  Sex   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris    1  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th...    2  38.0      1      0
2                             Heikkinen, Miss. Laina    2  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    2  35.0      1      0
4                           Allen, Mr. William Henry    1  35.0      0      0

             Ticket     Fare Cabin  Embarked_C  Embarked_Q  Embarked_S
0         A/5 21171   7.2500     U           0           0           1
1          PC 17599  71.2833   C85           1           0           0
2  STON/O2. 3101282   7.9250     U           0           0           1
3            113803  53.1000  C123           0           0           1
4            373450   8.0500     U           0           0           1
```

In [17]:

```
#对客舱号进行特征提取
#一等舱=1；二等舱=2；三等舱=3
#和登船港口处理方法相同，将Pclass分为Pclass_1，Pclass_2，Pclass_3，采用One-hot编码

#存放提取后的特征
pclassDf= pd.DataFrame()
#使用get_dummies进行one-hot编码，产生虚拟变量（dummy variables），列名前缀是Embarke
#get_dummies函数的作用是将该列出现值的类型再重新划分为子列，通过子列里的bool类型标识
pclassDf = pd.get_dummies( full['Pclass'] , prefix='Pclass' )
#打印输出前五条数据，查看Pclass的处理情况
print(pclassDf.head())
```

```
   Pclass_1  Pclass_2  Pclass_3
0         0         0         1
1         1         0         0
2         0         0         1
3         1         0         0
4         0         0         1
```

In [18]:

```
#用处理后的数据来替换原先数据里的“客舱等级”，先删掉客舱等级这一列
full.drop('Pclass',axis=1,inplace=True)
#添加one-hot编码产生的虚拟变量（dummy variables）到泰坦尼克号数据集full
full = pd.concat([full,pclassDf],axis=1)
#显示数据前五条
print(full.head())
```

```
   PassengerId  Survived                                               Name  \
0            1       0.0                            Braund, Mr. Owen Harris
1            2       1.0  Cumings, Mrs. John Bradley (Florence Briggs Th...
2            3       1.0                             Heikkinen, Miss. Laina
3            4       1.0       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4            5       0.0                           Allen, Mr. William Henry

   Sex   Age  SibSp  Parch            Ticket     Fare Cabin  Embarked_C  \
0    1  22.0      1      0         A/5 21171   7.2500     U           0
1    2  38.0      1      0          PC 17599  71.2833   C85           1
2    2  26.0      0      0  STON/O2. 3101282   7.9250     U           0
3    2  35.0      1      0            113803  53.1000  C123           0
4    1  35.0      0      0            373450   8.0500     U           0

   Embarked_Q  Embarked_S  Pclass_1  Pclass_2  Pclass_3
0           0           1         0         0         1
1           0           0         1         0         0
2           0           1         0         0         1
3           0           1         1         0         0
4           0           1         0         0         1
```

In [19]:

```
#对乘前五条客姓名进行特征提取
#首先展示姓名列的数据,展示前五条数据
print(full['Name'].head())
```

```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                             Allen, Mr. William Henry
Name: Name, dtype: object
```

In [20]:

```
#从上面的结果可以看出，大部分乘客都会有一个头衔，比如'Mrs','Mr'等
#因此我们将这种头衔拆分出来作为一个新的可用变量，帮助预测生存率

#定义函数：从姓名中获取头衔
def getTitle(Name):
    #split用于字符串拆分，返回一个列表。
    #姓名中’Braund, Mr. Owen Harris’，逗号前面的是“名”，逗号后面是‘头衔. 姓’
    str1=Name.split( ',' )[1]#返回Mr. Owen Harris
    str2=str1.split( '.' )[0]#返回Mr
    #strip() 方法用于移除字符串头尾指定的字符（默认为空格）
    str3=str2.strip()
    return str3

#存放提取后的特征
titleDf = pd.DataFrame()
#map函数：对Series每个数据应用自定义的函数计算
titleDf['Title'] = full['Name'].map(getTitle)

#查看titleDF的种类
print(titleDf.value_counts())
```

```
Title
Mr              757
Miss            260
Mrs             197
Master           61
Rev               8
Dr                8
Col               4
Ms                2
Major             2
Mlle              2
Sir               1
Capt              1
Mme               1
Lady              1
Jonkheer          1
Dona              1
Don               1
the Countess      1
dtype: int64
```

In [21]:

```
#结果显示一共有18种头衔，过于繁杂，因此根据常用的头衔及含义对其进行简化和One-hot编码
#将上面的姓名结果中头衔字符串与定义头衔一一映射，形成类别的对应关系
title_mapDict = {


                    "Mr" :          "Mr",
                    "Mrs" :         "Mrs",
                    "Master" :      "Master",
                    "Miss" :        "Miss",
                    "Rev":          "Officer",
                    "Dr":           "Officer",
                    "Col":          "Officer",
                    "Ms":           "Mrs",
                    "Major":        "Officer",
                    "Mlle":         "Miss",
                    "Sir" :         "Royalty",
                    "Capt":         "Officer",
                    "Mme":          "Mrs",
                    "the Countess":"Royalty",
                    "Lady" :        "Royalty",
                    "Jonkheer":     "Royalty",
                    "Dona":         "Royalty",
                    "Don":          "Royalty",

                    }

#map函数：对Series每个数据应用自定义的函数计算
titleDf['Title'] = titleDf['Title'].map(title_mapDict)

#使用get_dummies进行one-hot编码
titleDf = pd.get_dummies(titleDf['Title'])
#查看处理结果
print(titleDf.head())
```

```
   Master  Miss  Mr  Mrs  Officer  Royalty
0       0     0   1    0        0        0
1       0     0   0    1        0        0
2       0     1   0    0        0        0
3       0     0   0    1        0        0
4       0     0   1    0        0        0
```

In [22]:

```
#删掉原先数据中姓名这一列
```

```
full.drop('Name',axis=1,inplace=True)
#添加one-hot编码产生的虚拟变量（dummy variables）到泰坦尼克号数据集full中
full = pd.concat([full,titleDf],axis=1)
#查看替换后的数据，显示前五条
print(full.head())
```

```
   PassengerId  Survived  Sex   Age  SibSp  Parch            Ticket     Fare  \
0            1       0.0    1  22.0      1      0         A/5 21171   7.2500
1            2       1.0    2  38.0      1      0          PC 17599  71.2833
2            3       1.0    2  26.0      0      0  STON/O2. 3101282   7.9250
3            4       1.0    2  35.0      1      0            113803  53.1000
4            5       0.0    1  35.0      0      0            373450   8.0500

  Cabin  Embarked_C  ...  Embarked_S  Pclass_1  Pclass_2  Pclass_3  Master  \
0     U           0  ...           1         0         0         1       0
1   C85           1  ...           0         1         0         0       0
2     U           0  ...           1         0         0         1       0
3  C123           0  ...           1         1         0         0       0
4     U           0  ...           1         0         0         1       0

   Miss  Mr  Mrs  Officer  Royalty
0     0   1    0        0        0
1     0   0    1        0        0
2     1   0    0        0        0
3     0   0    1        0        0
4     0   1    0        0        0

[5 rows x 21 columns]
```

In [23]:

```
#对客舱号进行特征提取
#显示客舱号的前五条数据
print(full['Cabin'].head())
```

```
0       U
1     C85
2       U
3    C123
4       U
Name: Cabin, dtype: object
```

In [24]:

```
#存放客舱号信息
cabinDf = pd.DataFrame()
#用客舱号的首字母代表客舱的类别，取首字母
full[ 'Cabin' ] = full[ 'Cabin' ].map( lambda c : c[0] )
##使用get_dummies进行one-hot编码，列名前缀是Cabin
cabinDf = pd.get_dummies( full['Cabin'] , prefix = 'Cabin' )
#显示处理后的客舱号信息，输出前五条数据
print(cabinDf.head())
```

```
   Cabin_A  Cabin_B  Cabin_C  Cabin_D  Cabin_E  Cabin_F  Cabin_G  Cabin_T  \
0        0        0        0        0        0        0        0        0
1        0        0        1        0        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        0        1        0        0        0        0        0
4        0        0        0        0        0        0        0        0

   Cabin_U
0        1
1        0
2        1
3        0
4        1
```

In [25]:
```
#用处理后的数据来替换原先数据里的“客舱号”，先删掉客舱号这一列
full.drop('Cabin',axis=1,inplace=True)
#添加one-hot编码产生的虚拟变量（dummy variables）到泰坦尼克号数据集full
full = pd.concat([full,cabinDf],axis=1)
#显示数据前五条
print(full.head())
```

```
   PassengerId  Survived  Sex   Age  SibSp  Parch            Ticket     Fare  \
0            1       0.0    1  22.0      1      0         A/5 21171   7.2500
1            2       1.0    2  38.0      1      0          PC 17599  71.2833
2            3       1.0    2  26.0      0      0  STON/O2. 3101282   7.9250
3            4       1.0    2  35.0      1      0            113803  53.1000
4            5       0.0    1  35.0      0      0            373450   8.0500

   Embarked_C  Embarked_Q  ...  Royalty  Cabin_A  Cabin_B  Cabin_C  Cabin_D  \
0           0           0  ...        0        0        0        0        0
1           1           0  ...        0        0        0        1        0
2           0           0  ...        0        0        0        0        0
3           0           0  ...        0        0        0        1        0
4           0           0  ...        0        0        0        0        0

   Cabin_E  Cabin_F  Cabin_G  Cabin_T  Cabin_U
0        0        0        0        0        1
1        0        0        0        0        0
2        0        0        0        0        1
3        0        0        0        0        0
4        0        0        0        0        1

[5 rows x 29 columns]
```

In [26]:
```
#建立家庭人数和家庭类别
#家庭人数=同代直系亲属数（Parch）+不同代直系亲属数（SibSp）+乘客自己
#小家庭Family_Single：家庭人数=1
#中等家庭Family_Small：2<=家庭人数<=4
#大家庭Family_Large：家庭人数>=5

#存放家庭信息
familyDf = pd.DataFrame()
#计算家庭人数
familyDf[ 'FamilySize' ] = full[ 'Parch' ] + full[ 'SibSp' ] + 1
#判断家庭类别
familyDf[ 'Family_Single' ] = familyDf[ 'FamilySize' ].map( lambda s : 1 if s == 1
familyDf[ 'Family_Small' ]  = familyDf[ 'FamilySize' ].map( lambda s : 1 if 2 <= s
familyDf[ 'Family_Large' ]  = familyDf[ 'FamilySize' ].map( lambda s : 1 if 5 <= s
#查看处理后的结果，显示前五条数据
print(familyDf.head())
```

```
   FamilySize  Family_Single  Family_Small  Family_Large
0           2              0             1             0
1           2              0             1             0
2           1              1             0             0
3           2              0             1             0
4           1              1             0             0
```

In [27]:
```
#添加one-hot编码产生的虚拟变量（dummy variables）到泰坦尼克号数据集full
full = pd.concat([full,familyDf],axis=1)
#查看添加“家庭类别”后的数据结果
print(full.head())
```

```
   PassengerId  Survived  Sex   Age  SibSp  Parch            Ticket     Fare  \
0            1       0.0    1  22.0      1      0         A/5 21171   7.2500
1            2       1.0    2  38.0      1      0          PC 17599  71.2833
2            3       1.0    2  26.0      0      0  STON/O2. 3101282   7.9250
3            4       1.0    2  35.0      1      0            113803  53.1000
4            5       0.0    1  35.0      0      0            373450   8.0500

   Embarked_C  Embarked_Q  ...  Cabin_D  Cabin_E  Cabin_F  Cabin_G  Cabin_T  \
0           0           0  ...        0        0        0        0        0
1           1           0  ...        0        0        0        0        0
2           0           0  ...        0        0        0        0        0
3           0           0  ...        0        0        0        0        0
4           0           0  ...        0        0        0        0        0

   Cabin_U  FamilySize  Family_Single  Family_Small  Family_Large
0        1           2              0             1             0
1        0           2              0             1             0
2        1           1              1             0             0
3        0           2              0             1             0
4        1           1              1             0             0

[5 rows x 33 columns]
```

In [28]:

```
#至此所有的特性已经提取完毕，查看数据最终的结构
print(full.shape)
```

```
(1309, 33)
```

In [29]:

```
#----------------下面进行特征选择---------------------

#选择出部分特征用于预测生存率，这里运用相关系数来筛选特征

"""
DataFrame.corr(method='pearson', min_periods=1)
参数说明：
method：可选值为{‘pearson’, ‘kendall’, ‘spearman’}
pearson：Pearson相关系数来衡量两个数据集合是否在一条线上面，即针对线性数据的相关系数
kendall：用于反映分类变量相关性的指标，即针对无序序列的相关系数，非正太分布的数据
spearman：非线性的，非正太分析的数据的相关系数
min_periods：样本最少的数据量
返回值：各类型之间的相关系数DataFrame表格。

"""


correlation_Df=full.corr()
print(correlation_Df)
```

	PassengerId	Survived	Sex	Age	SibSp	Parch
PassengerId	1.000000	-0.005007	-0.013406	0.025731	-0.055224	0.008942
Survived	-0.005007	1.000000	0.543351	-0.070323	-0.035322	0.081629
Sex	-0.013406	0.543351	1.000000	-0.057397	0.109609	0.213125
Age	0.025731	-0.070323	-0.057397	1.000000	-0.190747	-0.130872
SibSp	-0.055224	-0.035322	0.109609	-0.190747	1.000000	0.373587
Parch	0.008942	0.081629	0.213125	-0.130872	0.373587	1.000000
Fare	0.031416	0.257307	0.185484	0.171521	0.160224	0.221522
Embarked_C	0.048101	0.168240	0.066564	0.076179	-0.048396	-0.008635
Embarked_Q	0.011585	0.003650	0.088651	-0.012718	-0.048678	-0.100943
Embarked_S	-0.049836	-0.149683	-0.115193	-0.059153	0.073709	0.071881
Pclass_1	0.026495	0.285904	0.107371	0.362587	-0.034256	-0.013033
Pclass_2	0.022714	0.093349	0.028862	-0.014193	-0.052419	-0.010057
Pclass_3	-0.041544	-0.322308	-0.116562	-0.302093	0.072610	0.019521
Master	0.002254	0.085221	-0.164375	-0.363923	0.329171	0.253482
Miss	-0.050027	0.332795	0.672819	-0.254146	0.077564	0.066473
Mr	0.014116	-0.549199	-0.870678	0.165476	-0.243104	-0.304780
Mrs	0.033299	0.344935	0.571176	0.198091	0.061643	0.213491
Officer	0.002231	-0.031316	-0.087288	0.162818	-0.013813	-0.032631
Royalty	0.004400	0.033391	0.020408	0.059466	-0.010787	-0.030197
Cabin_A	-0.002831	0.022287	-0.047561	0.125177	-0.039808	-0.030707
Cabin_B	0.015895	0.175095	0.094453	0.113458	-0.011569	0.073051
Cabin_C	0.006092	0.114652	0.077473	0.167993	0.048616	0.009601
Cabin_D	0.000549	0.150716	0.057396	0.132886	-0.015727	-0.027385
Cabin_E	-0.008136	0.145321	0.040340	0.106600	-0.027180	0.001084
Cabin_F	0.000306	0.057935	0.006655	-0.072644	-0.008619	0.020481
Cabin_G	-0.045949	0.016040	0.083285	-0.085977	0.006015	0.058325
Cabin_T	-0.023049	-0.026456	-0.020558	0.032461	-0.013247	-0.012304
Cabin_U	0.000208	-0.316912	-0.137396	-0.271918	0.009064	-0.036806
FamilySize	-0.031437	0.016639	0.188583	-0.196996	0.861952	0.792296
Family_Single	0.028546	-0.203367	-0.284537	0.116675	-0.591077	-0.549022
Family_Small	0.002975	0.279855	0.255196	-0.038189	0.253590	0.248532
Family_Large	-0.063415	-0.125147	0.077748	-0.161210	0.699681	0.624627

	Fare	Embarked_C	Embarked_Q	Embarked_S	...	Cabin_D
PassengerId	0.031416	0.048101	0.011585	-0.049836	...	0.000549
Survived	0.257307	0.168240	0.003650	-0.149683	...	0.150716
Sex	0.185484	0.066564	0.088651	-0.115193	...	0.057396
Age	0.171521	0.076179	-0.012718	-0.059153	...	0.132886
SibSp	0.160224	-0.048396	-0.048678	0.073709	...	-0.015727
Parch	0.221522	-0.008635	-0.100943	0.071881	...	-0.027385
Fare	1.000000	0.286241	-0.130054	-0.169894	...	0.072737
Embarked_C	0.286241	1.000000	-0.164166	-0.778262	...	0.107782
Embarked_Q	-0.130054	-0.164166	1.000000	-0.491656	...	-0.061459
Embarked_S	-0.169894	-0.778262	-0.491656	1.000000	...	-0.056023
Pclass_1	0.599956	0.325722	-0.166101	-0.181800	...	0.275698
Pclass_2	-0.121372	-0.134675	-0.121973	0.196532	...	-0.037929
Pclass_3	-0.419616	-0.171430	0.243706	-0.003805	...	-0.207455
Master	0.011596	-0.014172	-0.009091	0.018297	...	-0.042192
Miss	0.092051	-0.014351	0.198804	-0.113886	...	-0.012516
Mr	-0.192192	-0.065538	-0.080224	0.108924	...	-0.030261
Mrs	0.139235	0.098379	-0.100374	-0.022950	...	0.080393
Officer	0.028696	0.003678	-0.003212	-0.001202	...	0.006055
Royalty	0.026214	0.077213	-0.021853	-0.054250	...	-0.012950
Cabin_A	0.020094	0.094914	-0.042105	-0.056984	...	-0.024952
Cabin_B	0.393743	0.161595	-0.073613	-0.095790	...	-0.043624
Cabin_C	0.401370	0.158043	-0.059151	-0.101861	...	-0.053083
Cabin_D	0.072737	0.107782	-0.061459	-0.056023	...	1.000000
Cabin_E	0.073949	0.027566	-0.042877	0.002960	...	-0.034317
Cabin_F	-0.037567	-0.020010	-0.020282	0.030575	...	-0.024369
Cabin_G	-0.022857	-0.031566	-0.019941	0.040560	...	-0.011817
Cabin_T	0.001179	-0.014095	-0.008904	0.018111	...	-0.005277
Cabin_U	-0.507197	-0.258257	0.142369	0.137351	...	-0.353822
FamilySize	0.226465	-0.036553	-0.087190	0.087771	...	-0.025313

Family_Single	-0.274826	-0.107874	0.127214	0.014246	...	-0.074310
Family_Small	0.197281	0.159594	-0.122491	-0.062909	...	0.102432
Family_Large	0.170853	-0.092825	-0.018423	0.093671	...	-0.049336

	Cabin_E	Cabin_F	Cabin_G	Cabin_T	Cabin_U	FamilySize	\
PassengerId	-0.008136	0.000306	-0.045949	-0.023049	0.000208	-0.031437	
Survived	0.145321	0.057935	0.016040	-0.026456	-0.316912	0.016639	
Sex	0.040340	0.006655	0.083285	-0.020558	-0.137396	0.188583	
Age	0.106600	-0.072644	-0.085977	0.032461	-0.271918	-0.196996	
SibSp	-0.027180	-0.008619	0.006015	-0.013247	0.009064	0.861952	
Parch	0.001084	0.020481	0.058325	-0.012304	-0.036806	0.792296	
Fare	0.073949	-0.037567	-0.022857	0.001179	-0.507197	0.226465	
Embarked_C	0.027566	-0.020010	-0.031566	-0.014095	-0.258257	-0.036553	
Embarked_Q	-0.042877	-0.020282	-0.019941	-0.008904	0.142369	-0.087190	
Embarked_S	0.002960	0.030575	0.040560	0.018111	0.137351	0.087771	
Pclass_1	0.242963	-0.073083	-0.035441	0.048310	-0.776987	-0.029656	
Pclass_2	-0.050210	0.127371	-0.032081	-0.014325	0.176485	-0.039976	
Pclass_3	-0.169063	-0.041178	0.056964	-0.030057	0.527614	0.058430	
Master	0.001860	0.058311	-0.013690	-0.006113	0.041178	0.355061	
Miss	0.008700	-0.003088	0.061881	-0.013832	-0.004364	0.087350	
Mr	-0.032953	-0.026403	-0.072514	0.023611	0.131807	-0.326487	
Mrs	0.045538	0.013376	0.042547	-0.011742	-0.162253	0.157233	
Officer	-0.024048	-0.017076	-0.008281	-0.003698	-0.067030	-0.026921	
Royalty	-0.012202	-0.008665	-0.004202	-0.001876	-0.071672	-0.023600	
Cabin_A	-0.023510	-0.016695	-0.008096	-0.003615	-0.242399	-0.042967	
Cabin_B	-0.041103	-0.029188	-0.014154	-0.006320	-0.423794	0.032318	
Cabin_C	-0.050016	-0.035516	-0.017224	-0.007691	-0.515684	0.037226	
Cabin_D	-0.034317	-0.024369	-0.011817	-0.005277	-0.353822	-0.025313	
Cabin_E	1.000000	-0.022961	-0.011135	-0.004972	-0.333381	-0.017285	
Cabin_F	-0.022961	1.000000	-0.007907	-0.003531	-0.236733	0.005525	
Cabin_G	-0.011135	-0.007907	1.000000	-0.001712	-0.114803	0.035835	
Cabin_T	-0.004972	-0.003531	-0.001712	1.000000	-0.051263	-0.015438	
Cabin_U	-0.333381	-0.236733	-0.114803	-0.051263	1.000000	-0.014155	
FamilySize	-0.017285	0.005525	0.035835	-0.015438	-0.014155	1.000000	
Family_Single	-0.042535	0.004055	-0.076397	0.022411	0.175812	-0.688864	
Family_Small	0.068007	0.012756	0.087471	-0.019574	-0.211367	0.302640	
Family_Large	-0.046485	-0.033009	-0.016008	-0.007148	0.056438	0.801623	

	Family_Single	Family_Small	Family_Large
PassengerId	0.028546	0.002975	-0.063415
Survived	-0.203367	0.279855	-0.125147
Sex	-0.284537	0.255196	0.077748
Age	0.116675	-0.038189	-0.161210
SibSp	-0.591077	0.253590	0.699681
Parch	-0.549022	0.248532	0.624627
Fare	-0.274826	0.197281	0.170853
Embarked_C	-0.107874	0.159594	-0.092825
Embarked_Q	0.127214	-0.122491	-0.018423
Embarked_S	0.014246	-0.062909	0.093671
Pclass_1	-0.126551	0.165965	-0.067523
Pclass_2	-0.035075	0.097270	-0.118495
Pclass_3	0.138250	-0.223338	0.155560
Master	-0.265355	0.120166	0.301809
Miss	-0.023890	-0.018085	0.083422
Mr	0.386262	-0.300872	-0.194207
Mrs	-0.354649	0.361247	0.012893
Officer	0.013303	0.003966	-0.034572
Royalty	0.008761	-0.000073	-0.017542
Cabin_A	0.045227	-0.029546	-0.033799
Cabin_B	-0.087912	0.084268	0.013470
Cabin_C	-0.137498	0.141925	0.001362
Cabin_D	-0.074310	0.102432	-0.049336
Cabin_E	-0.042535	0.068007	-0.046485
Cabin_F	0.004055	0.012756	-0.033009

```
Cabin_G          -0.076397      0.087471     -0.016008
Cabin_T           0.022411     -0.019574     -0.007148
Cabin_U           0.175812     -0.211367      0.056438
FamilySize       -0.688864      0.302640      0.801623
Family_Single     1.000000     -0.873398     -0.318944
Family_Small     -0.873398      1.000000     -0.183007
Family_Large     -0.318944     -0.183007      1.000000

[32 rows x 32 columns]
```

In [30]:

```
#查看各个特征与生存情况（Survived）的相关系数，降序排列
print(correlation_Df['Survived'].sort_values(ascending=False))
```

```
Survived         1.000000
Sex              0.543351
Mrs              0.344935
Miss             0.332795
Pclass_1         0.285904
Family_Small     0.279855
Fare             0.257307
Cabin_B          0.175095
Embarked_C       0.168240
Cabin_D          0.150716
Cabin_E          0.145321
Cabin_C          0.114652
Pclass_2         0.093349
Master           0.085221
Parch            0.081629
Cabin_F          0.057935
Royalty          0.033391
Cabin_A          0.022287
FamilySize       0.016639
Cabin_G          0.016040
Embarked_Q       0.003650
PassengerId     -0.005007
Cabin_T         -0.026456
Officer         -0.031316
SibSp           -0.035322
Age             -0.070323
Family_Large    -0.125147
Embarked_S      -0.149683
Family_Single   -0.203367
Cabin_U         -0.316912
Pclass_3        -0.322308
Mr              -0.549199
Name: Survived, dtype: float64
```

In [31]:

```
#特征选择,选择头衔、客舱等级、家庭类别、船票价格、船舱号、登船港口、性别
full_X=pd.concat([titleDf,pclassDf,familyDf,full['Fare'],cabinDf,embarkedDf,full['S
print(full_X.head())
```

```
   Master  Miss  Mr  Mrs  Officer  Royalty  Pclass_1  Pclass_2  Pclass_3  \
0       0     0   1    0        0        0         0         0         1
1       0     0   0    1        0        0         1         0         0
2       0     1   0    0        0        0         0         0         1
3       0     0   0    1        0        0         1         0         0
4       0     0   1    0        0        0         0         0         1

   FamilySize  ...  Cabin_D  Cabin_E  Cabin_F  Cabin_G  Cabin_T  Cabin_U  \
0           2  ...        0        0        0        0        0        1
1           2  ...        0        0        0        0        0        0
2           1  ...        0        0        0        0        0        1
3           2  ...        0        0        0        0        0        0
4           1  ...        0        0        0        0        0        1

   Embarked_C  Embarked_Q  Embarked_S  Sex
0           0           0           1    1
1           1           0           0    2
2           0           0           1    2
3           0           0           1    2
4           0           0           1    1

[5 rows x 27 columns]
```

In [32]:

```
#-------下面进行数据可视化部分-----------
#导入可视化包
import matplotlib.pyplot as plt
#选取前891条数据进行可视化分析
sourceRow=891
viDf=full.loc[0:sourceRow-1,:]
#查看前五条数据
print(viDf.head())
```

```
   PassengerId  Survived  Sex   Age  SibSp  Parch            Ticket     Fare  \
0            1       0.0    1  22.0      1      0         A/5 21171   7.2500
1            2       1.0    2  38.0      1      0          PC 17599  71.2833
2            3       1.0    2  26.0      0      0  STON/O2. 3101282   7.9250
3            4       1.0    2  35.0      1      0            113803  53.1000
4            5       0.0    1  35.0      0      0            373450   8.0500

   Embarked_C  Embarked_Q  ...  Cabin_D  Cabin_E  Cabin_F  Cabin_G  Cabin_T  \
0           0           0  ...        0        0        0        0        0
1           1           0  ...        0        0        0        0        0
2           0           0  ...        0        0        0        0        0
3           0           0  ...        0        0        0        0        0
4           0           0  ...        0        0        0        0        0

   Cabin_U  FamilySize  Family_Single  Family_Small  Family_Large
0        1           2              0             1             0
1        0           2              0             1             0
2        1           1              1             0             0
3        0           2              0             1             0
4        1           1              1             0             0

[5 rows x 33 columns]
```

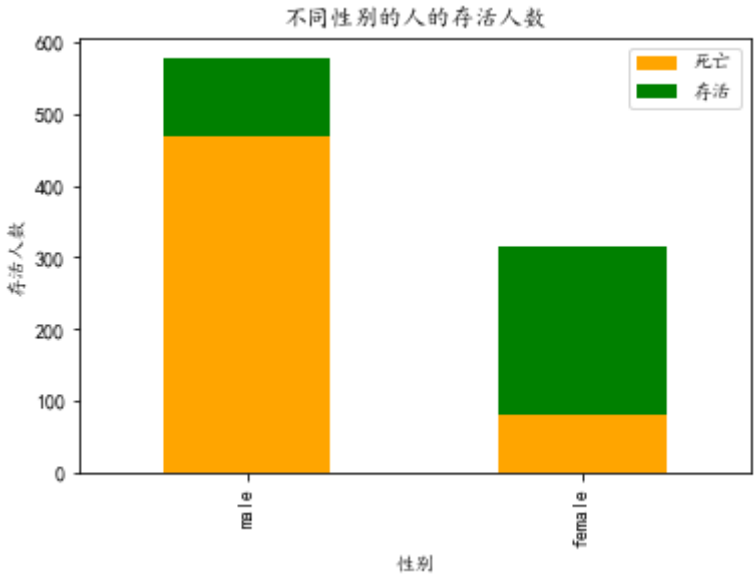
In [33]:

```
#性别和生存率的关系

#解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi'] # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题


#男性存活的人数和死亡的人数
```

```
#sex=full.groupby("Sex").Survived.mean()
#sexDF=pd.DataFrame(sex)
survived_m=viDf.loc[viDf['Sex']==1,'Survived'].value_counts()
survived_f=viDf.loc[viDf['Sex']==2,'Survived'].value_counts()
#用sexDf保存男女存活人数和死亡人数
sexDf=pd.DataFrame({'male':survived_m,'female':survived_f})
print(sexDf)
#sexDf.plot(kind="bar")# 绘制直方图

#绘制堆积图让结果更加直观
sexDf.T.plot(kind="bar",stacked=True,color=['orange','green'])

#添加文本
plt.xlabel("性别")
plt.ylabel("存活人数")
plt.legend(labels=['死亡','存活'])
#添加标题
plt.title("不同性别的人的存活人数")
```

Out[33]:

```
     male  female
0.0   468      81
1.0   109     233
Text(0.5, 1.0, '不同性别的人的存活人数')
```

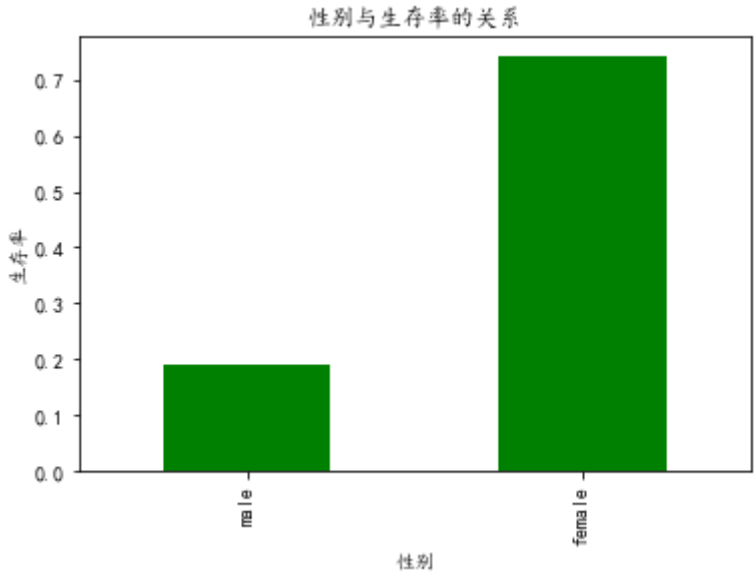


In [34]:

```
#性别与生存率的关系
print(sexDf)
for i in sexDf.columns:
    sexDf.loc['生存率',i]=sexDf.loc[1,i]/sexDf[i].sum()
#画出条形图，直观明了
sexDf.loc['生存率'].plot(kind='bar',color='green')
plt.xlabel('性别')
plt.ylabel('生存率')
plt.title('性别与生存率的关系')
```

Out[34]:

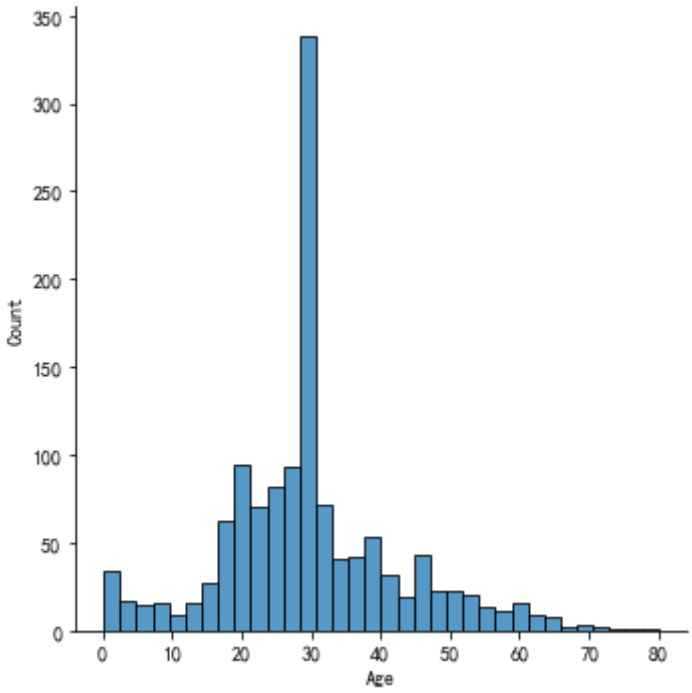
```
     male  female
0.0   468      81
1.0   109     233
Text(0.5, 1.0, '性别与生存率的关系')
```



In [35]:
```
#从上图可以看出，男性的死亡人数>女性死亡人数,女性的生存率远远大于男性
#合理推测在泰坦尼克号沉没时，男性做出了“女性优先”的决定
#让出了救生筏（和我看的电影结局吻合了，可怜的杰克）
```

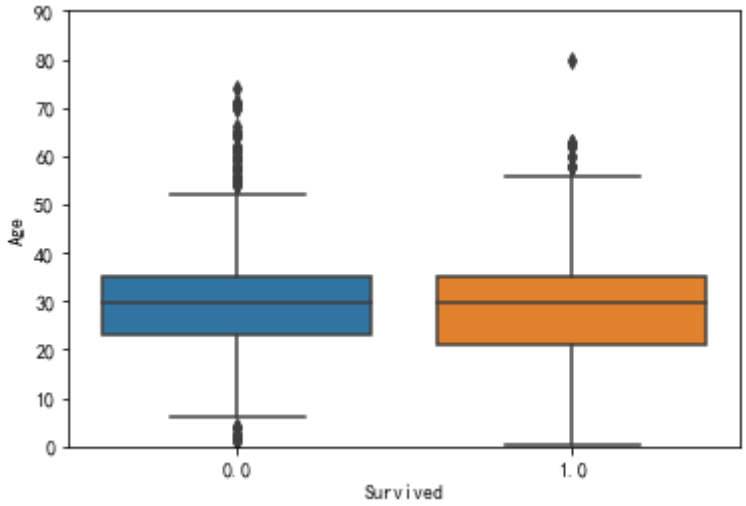
In [36]:
```
import seaborn as sns
#由下图可以看出，年龄是大致呈正态分布的，下面看看年龄和存活率有什么关系
sns.displot(full.Age)
```

Out[36]:
```
<seaborn.axisgrid.FacetGrid at 0x2548cfb7cd0>
```



In [37]:
```
#用箱线图可以看出，其实存活下来的人和死亡的人年龄中位数差不多
#平均年龄约为 30 岁,而这个信息基本没什么用，那下面用条形图试试
sns.boxplot(x="Survived",y='Age',data=full)
plt.ylim(0,90)
```

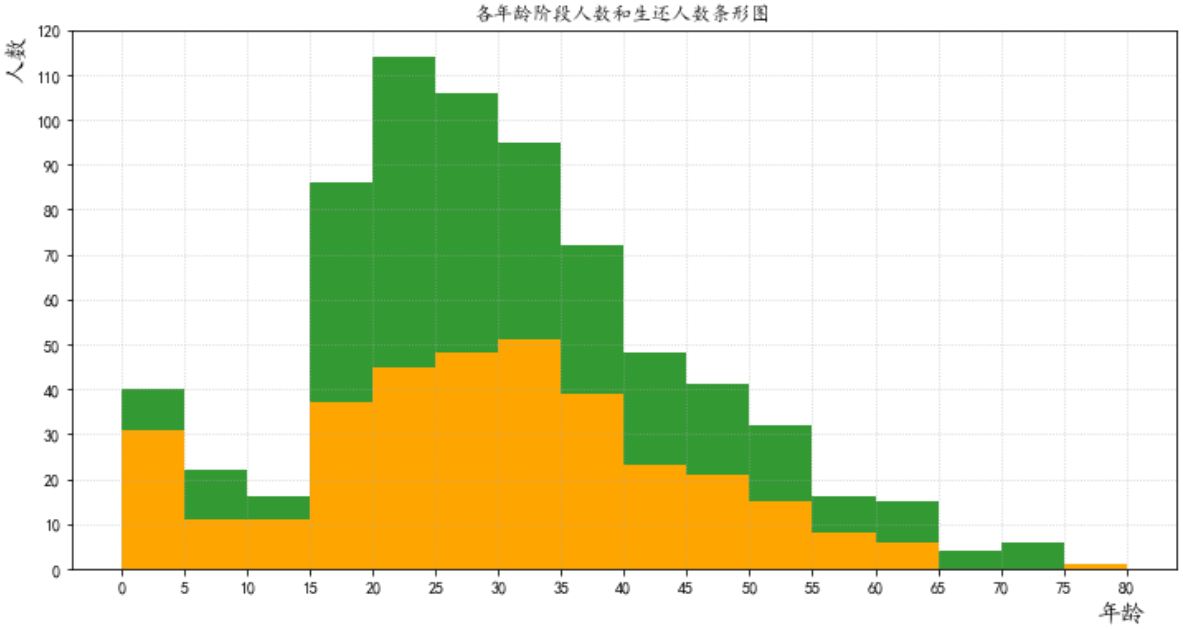
Out[37]:
```
(0.0, 90.0)
```



In [38]:

```
#年龄和生存率的关系
#因为viDf里面没有挑选出Age（年龄）这一项，所以就不用viDf来做了

#船上乘客的年龄范围
age_range = train['Age']
print("年龄范围：",age_range.min(),age_range.max())
# 各年龄阶段人数,生成直方图，共16个区间，年龄范围为0-80岁
age_num,_ = np.histogram(age_range,range=[0,80],bins=16 )
print("各年龄阶段人数：",age_num)

# 各年龄阶段生还人数
age_survived = []
for age in range(5,81,5):
    survived_num = train.loc[(age_range>=age-5) & (age_range<=age)]['Survived'].su
    age_survived.append(survived_num)
print("各年龄阶段生还人数：",age_survived)

# 各年龄阶段生存率，保留三位小数，为了下面画年龄和存活率的图做准备
age_sur_rate=[]
for i in range(16):
    rate=round((age_survived[i])/(age_num[i]),3)
    age_sur_rate.append(rate)
print("各年龄阶段生存率：",age_sur_rate)

# 绘制条形图
plt.figure(figsize=(12,6))
plt.bar(np.arange(2,78,5)+0.5,age_num,width=5,color=['green'],label='总人数',alpha=
plt.bar(np.arange(2,78,5)+0.5,age_survived,width=5,color=['orange'],label='生还人数
plt.xticks(range(0,81,5))
plt.yticks(range(0,121,10))
plt.xlabel('年龄',position=(0.95,0),fontsize=15)
plt.ylabel('人数',position=(0,0.95),fontsize=15)
plt.title('各年龄阶段人数和生还人数条形图')
plt.grid(True,linestyle=':',alpha=0.6)
```

```
年龄范围： 0.42 80.0
各年龄阶段人数： [ 40  22  16  86 114 106  95  72  48  41  32  16  15   4   6   1]
各年龄阶段生还人数： [31, 11, 11, 37, 45, 48, 51, 39, 23, 21, 15, 8, 6, 0, 0, 1]
各年龄阶段生存率： [0.775, 0.5, 0.688, 0.43, 0.395, 0.453, 0.537, 0.542, 0.479, 0.51
2, 0.469, 0.5, 0.4, 0.0, 0.0, 1.0]
```

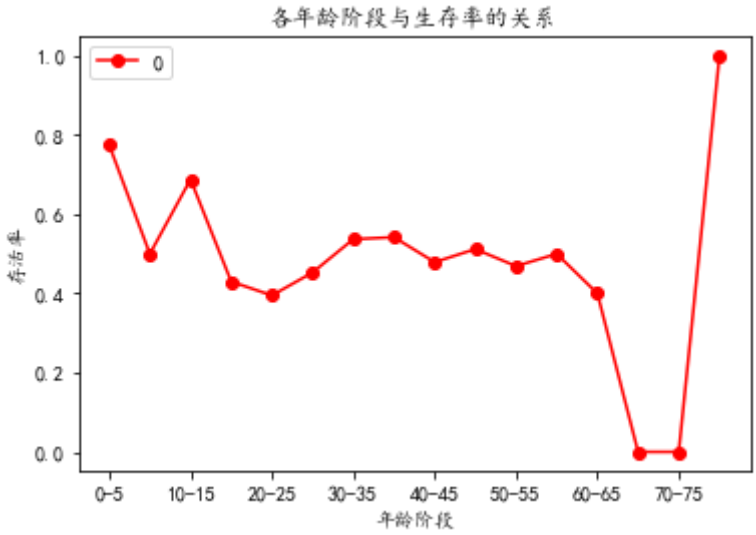


In [39]:
```
#从上图可以看出各年龄阶段生还人数条形图可以看出
#在0-15岁年龄段的绝大部分孩子都得以生还，而在15-80岁的年龄段都有近一半或超过一半的人
```

In [40]:
```
#将各年龄阶段的生存率生成dataframe，画出折线图
ageDf=pd.DataFrame([0.775, 0.5, 0.688, 0.43, 0.395, 0.453, 0.537, 0.542, 0.479, 0.5
          index=['0-5','5-10','10-15','15-20','20-25','25-30','30-35','35-40','40-45
print(ageDf)
ageDf.plot(kind='line',marker='o',color='r',title='各年龄阶段与生存率的关系')
plt.xlabel('年龄阶段')
plt.ylabel('存活率')
```

```
           0
0-5    0.775
5-10   0.500
10-15  0.688
15-20  0.430
20-25  0.395
25-30  0.453
30-35  0.537
35-40  0.542
40-45  0.479
45-50  0.512
50-55  0.469
55-60  0.500
60-65  0.400
65-70  0.000
70-75  0.000
75-80  1.000
```

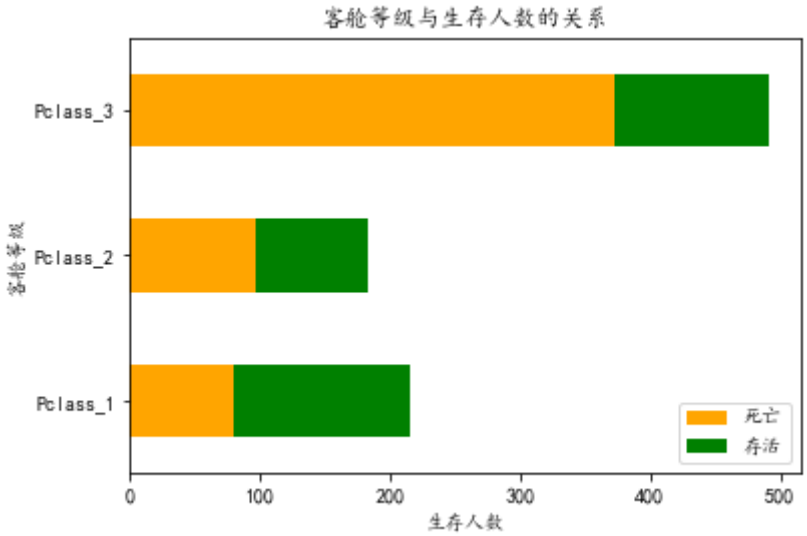
Out[40]:
```
Text(0, 0.5, '存活率')
```



In [41]:
```
#上图中，年龄阶段为75-80的数据出现异常，是因为这个年龄段的老人本身就少，所以使得生存
#但结合上面的条形图可以看出，虽然生存率高，但是没有参考性
```

In [42]:
```
#客舱等级与生存人数的关系
sur_pc1=viDf.loc[viDf['Pclass_1']==1,'Survived'].value_counts()
sur_pc2=viDf.loc[viDf['Pclass_2']==1,'Survived'].value_counts()
sur_pc3=viDf.loc[viDf['Pclass_3']==1,'Survived'].value_counts()
#将客舱等级分为三类，生成dataframe
pcDf=pd.DataFrame({'Pclass_1':sur_pc1,'Pclass_2':sur_pc2,'Pclass_3':sur_pc3})
#绘制水平柱状图
pcDf.T.plot(kind='barh',stacked=True,color=['orange','green'])
plt.xlabel('生存人数')
plt.ylabel('客舱等级')
plt.title('客舱等级与生存人数的关系')
plt.legend(labels=['死亡','存活'],loc='lower right')
```

Out[42]:
```
<matplotlib.legend.Legend at 0x2549d1e2970>
```



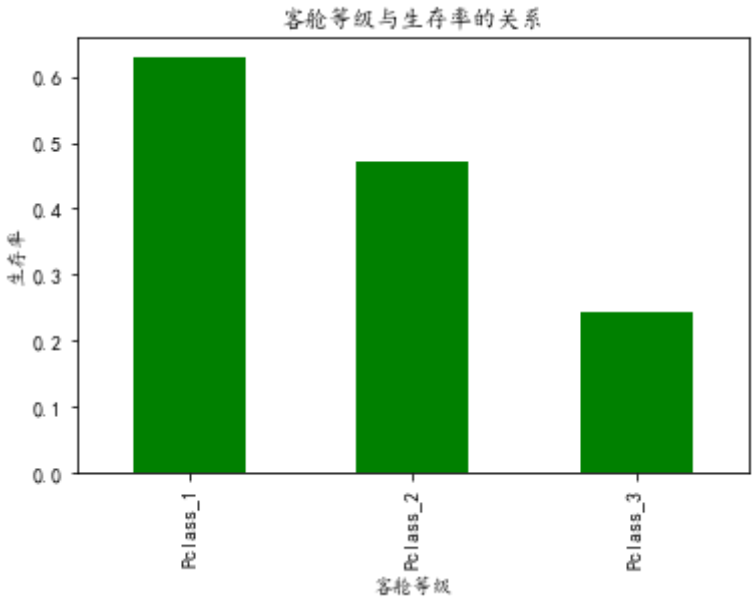
In [43]:
```
#由上图可以看出，人数上：三等舱>一等舱>二等舱，三等舱的死亡人数最多，一等舱的死亡人
```

In [44]:
```
#客舱等级与生存率的关系
print(pcDf)
for i in pcDf.columns:
    pcDf.loc['生存率',i]=pcDf.loc[1,i]/pcDf[i].sum()
```

```
#画出条形图，直观明了
pcDf.loc['生存率'].plot(kind='bar',color='green')
plt.xlabel('客舱等级')
plt.ylabel('生存率')
plt.title('客舱等级与生存率的关系')
```

Out[44]:

```
     Pclass_1  Pclass_2  Pclass_3
0.0        80        97       372
1.0       136        87       119
Text(0.5, 1.0, '客舱等级与生存率的关系')
```



In [45]:

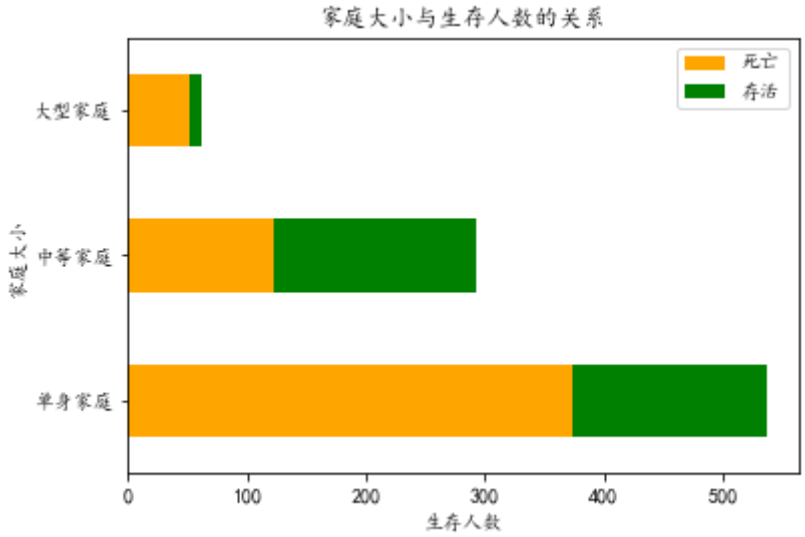
```
#由上图可以看出，一等舱的乘客生存率最高，且生存率随着舱位等级下降而递减。
#因此可以粗略判断事故发生时，上层阶级的人优先获得逃生机会。
```

In [46]:

```
#家庭大小和生存人数的关系
sur_s=viDf.loc[viDf['Family_Single']==1,'Survived'].value_counts()
sur_m=viDf.loc[viDf['Family_Small']==1,'Survived'].value_counts()
sur_l=viDf.loc[viDf['Family_Large']==1,'Survived'].value_counts()
#生成家庭大小和生存人数的dataframe
faDf=pd.DataFrame({'单身家庭':sur_s,'中等家庭':sur_m,'大型家庭':sur_l})
print(faDf)
#绘制水平柱状图
faDf.T.plot(kind='barh',stacked=True,color=['orange','green'])
plt.xlabel('生存人数')
plt.ylabel('家庭大小')
plt.title('家庭大小与生存人数的关系')
plt.legend(labels=['死亡','存活'],loc='upper right')
```

Out[46]:

```
     单身家庭  中等家庭  大型家庭
0.0   374   123    52
1.0   163   169    10
<matplotlib.legend.Legend at 0x2549d2df820>
```

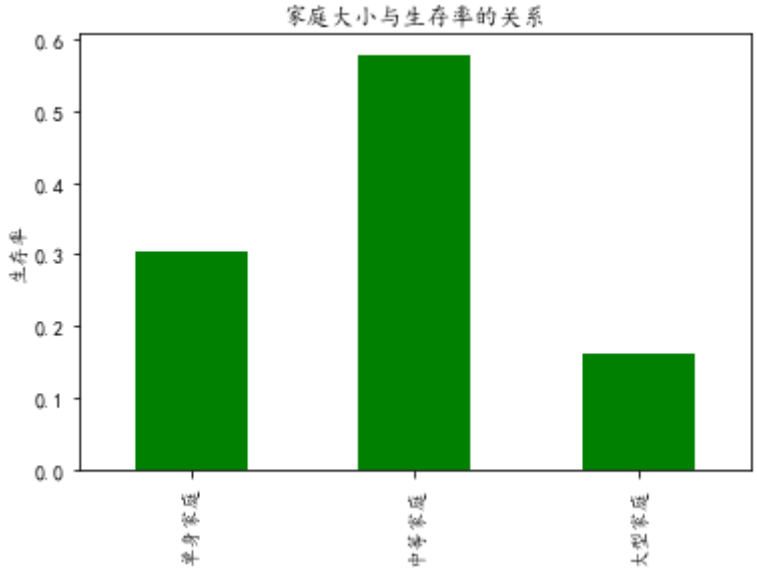


In [47]:
```
#从上图可以看出，船上单身家庭最多，中等家庭其次，大型家庭最少。死亡人数也是一样的趋
```

In [48]:
```
#家庭大小与生存率的关系
for i in faDf.columns:
    faDf.loc['生存率',i]=faDf.loc[1,i]/faDf[i].sum()
print(faDf)
#画出条形图，直观明了
faDf.loc['生存率'].plot(kind='bar',color='green')

plt.ylabel('生存率')
plt.title('家庭大小与生存率的关系')
```

```
          单身家庭        中等家庭       大型家庭
0.0   374.000000  123.000000  52.00000
1.0   163.000000  169.000000  10.00000
生存率    0.303538    0.578767   0.16129
```

Out[48]:
```
Text(0.5, 1.0, '家庭大小与生存率的关系')
```

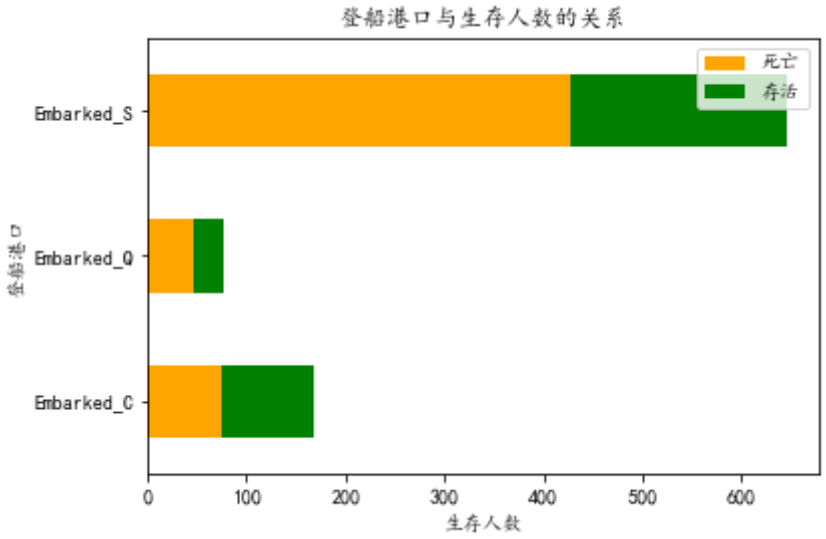


In [49]:
```
#由上图可知，中等家庭Family_Small: 2<=家庭人数<=4，即人数在2-4之间的家庭生存率最高
#大家庭Family_Large: 家庭人数>=5，即人数大于5的家庭存活率最低。
#合理推测，在事故发生的时候，有家人帮助逃生会比单身好的多，而家庭人数过多可能导致救
```

In [50]:
```
#登船港口与生存人数的关系
```

```
sur_C=viDf.loc[viDf['Embarked_C']==1,'Survived'].value_counts()
sur_Q=viDf.loc[viDf['Embarked_Q']==1,'Survived'].value_counts()
sur_S=viDf.loc[viDf['Embarked_S']==1,'Survived'].value_counts()
#生成登船港口和生存人数的dataframe
emDf=pd.DataFrame({'Embarked_C':sur_C,'Embarked_Q':sur_Q,'Embarked_S':sur_S})
#绘制水平柱状图
emDf.T.plot(kind='barh',stacked=True,color=['orange','green'])
plt.xlabel('生存人数')
plt.ylabel('登船港口')
plt.title('登船港口与生存人数的关系')
plt.legend(labels=['死亡','存活'],loc='upper right')
```

Out[50]:
```
<matplotlib.legend.Legend at 0x2549d4da0a0>
```

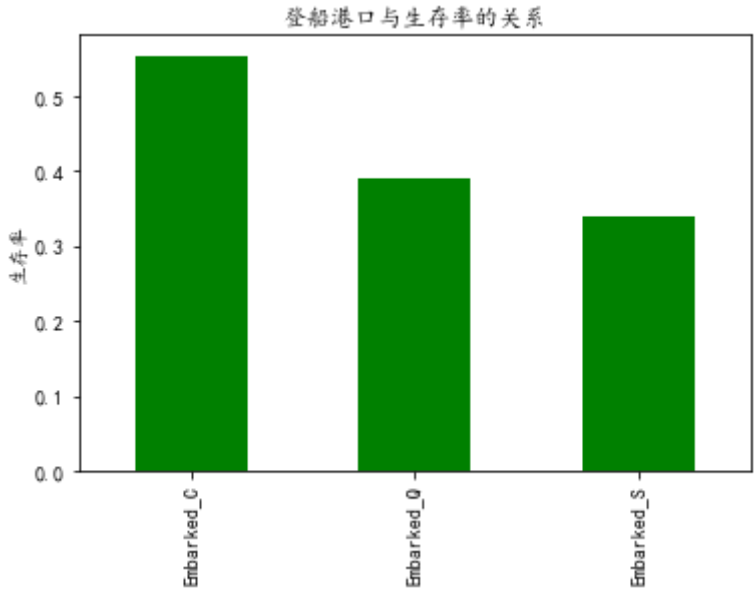


In [51]:
```
#从上图可以看出，从南安普顿登船的乘客存活的最多，但是死亡的也最多
#下面看看登船港口和生存率的关系
```

In [52]:
```
#登船港口与生存率的关系
for i in emDf.columns:
    emDf.loc['生存率',i]=emDf.loc[1,i]/emDf[i].sum()
print(emDf)
#画出条形图，直观明了
emDf.loc['生存率'].plot(kind='bar',color='green')

plt.ylabel('生存率')
plt.title('登船港口与生存率的关系')
```

```
     Embarked_C  Embarked_Q  Embarked_S
0.0   75.000000    47.00000  427.000000
1.0   93.000000    30.00000  219.000000
生存率    0.553571     0.38961    0.339009
```

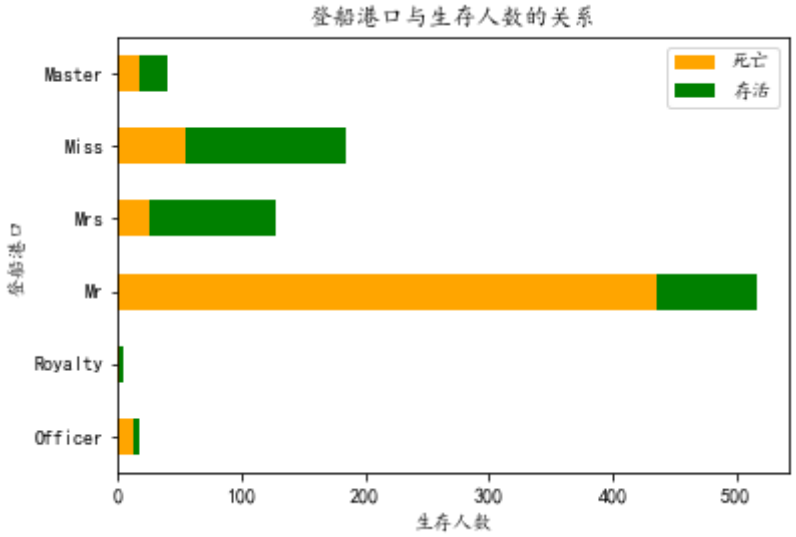
Out[52]:
```
Text(0.5, 1.0, '登船港口与生存率的关系')
```



In [53]:
```
#从上图可以看出，南安普顿登船的乘客存活率最高
#可能是因为泰坦尼克号的始发地是南安普顿，所以登船乘客基数大，所以即使死亡人数多，生
```

In [54]:
```
#乘客头衔和生存率的关系
sur_offi=viDf.loc[viDf['Officer']==1,'Survived'].value_counts()
sur_royal=viDf.loc[viDf['Royalty']==1,'Survived'].value_counts()
sur_mr=viDf.loc[viDf['Mr']==1,'Survived'].value_counts()
sur_mrs=viDf.loc[viDf['Mrs']==1,'Survived'].value_counts()
sur_miss=viDf.loc[viDf['Miss']==1,'Survived'].value_counts()
sur_master=viDf.loc[viDf['Master']==1,'Survived'].value_counts()
#生成乘客头衔和生存人数的dataframe
status_Df=pd.DataFrame({'Officer':sur_offi,'Royalty':sur_royal,'Mr':sur_mr,
                        'Mrs':sur_mrs,'Miss':sur_miss,'Master':sur_master})
#绘制水平柱状图
status_Df.T.plot(kind='barh',stacked=True,color=['orange','green'])
plt.xlabel('生存人数')
plt.ylabel('登船港口')
plt.title('登船港口与生存人数的关系')
plt.legend(labels=['死亡','存活'],loc='upper right')
```

Out[54]:
```
<matplotlib.legend.Legend at 0x2549d58ce50>
```



In [55]:
```
#乘客人数中已婚男士最多，其次是已婚女士和未婚女士，
```

```
#然后是有技能的人\教师，最后是政府官员和皇室成员。
```

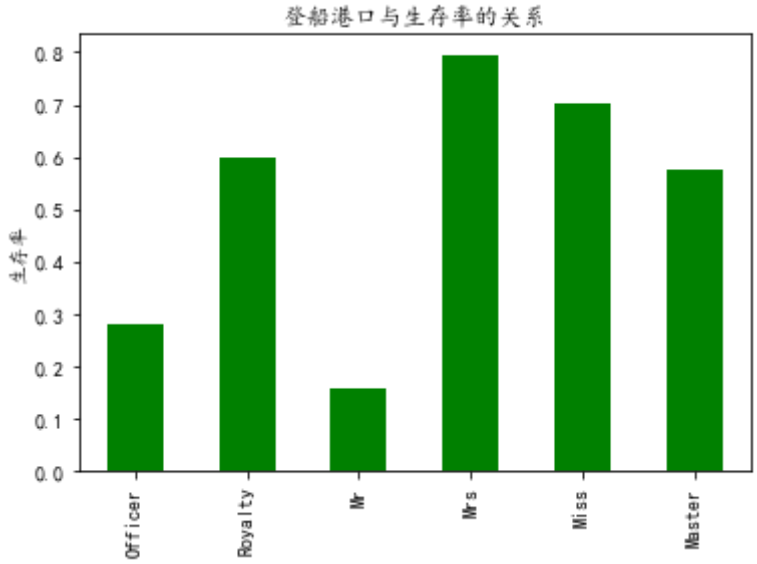
In [56]:

```
#乘客头衔与生存率的关系
for i in status_Df.columns:
    status_Df.loc['生存率',i]=status_Df.loc[1,i]/status_Df[i].sum()
print(status_Df)
#画出条形图，直观明了
status_Df.loc['生存率'].plot(kind='bar',color='green')

plt.ylabel('生存率')
plt.title('登船港口与生存率的关系')
```

```
       Officer  Royalty          Mr         Mrs        Miss  Master
0.0  13.000000      2.0  436.000000   26.000000   55.000000  17.000
1.0   5.000000      3.0   81.000000  101.000000  129.000000  23.000
生存率   0.277778      0.6    0.156673    0.795276    0.701087   0.575
```

Out[56]:

```
Text(0.5, 1.0, '登船港口与生存率的关系')
```



In [57]:

```
#从存活率看，已婚女士>未婚女士>皇室成员>有技能的人>政府官员>已婚男士，
#这说明当时船上可能秉承着女士优先，身份尊贵的乘客优先的原则。
#不得不说这艘船上绅士很多，有个王室身份或者地位比较高也更容易保命，interesting
```

In [58]:

```
#-----------------划分训练集和测试集-----------------
#构建模型
#原始数据集有891行
sourceRow=891
#原始数据集（从0行到891行）：特征
source_data = full_X.loc[0:sourceRow-1]
#原始数据集：标签
source_label = full.loc[0:sourceRow-1,'Survived']
#预测数据集(从891行到最后一行)：特征
predict_data = full_X.loc[sourceRow:,:]

#确认数据集和测试集的行数
print("用于训练模型的数据集行数:",source_data.shape[0])
print("用来预测的数据集行数:",predict_data.shape[0])
```

```
用于训练模型的数据集行数: 891
用来预测的数据集行数: 418
```

In [59]:

```
#将用于训练模型的数据集拆分为训练集和测试集
```

```
#train_test_split是交叉验证中常用的函数，功能是从样本中随机的按比例选取train data和
from sklearn.model_selection import train_test_split

#建立模型用的训练数据集和测试数据集
#训练集：测试集=4：1
train_data, test_data, train_label, test_label = train_test_split(source_data ,
                                                  source_label,
                                                  train_size=0.8)
#输出数据集大小
print ('原始数据集特征：',source_data.shape,
       '训练数据集特征：',train_data.shape ,
       '测试数据集特征：',test_data.shape)

print ('原始数据集标签：',source_label.shape,
       '训练数据集标签：',train_label.shape ,
       '测试数据集标签：',test_label.shape)
```

```
原始数据集特征： (891, 27) 训练数据集特征： (712, 27) 测试数据集特征： (179, 27)
原始数据集标签： (891,) 训练数据集标签： (712,) 测试数据集标签： (179,)
```

In [60]:

```
#查看原始数据的标签，输出前五行
print(source_label.head())
```

```
0    0.0
1    1.0
2    1.0
3    1.0
4    0.0
Name: Survived, dtype: float64
```

In [61]:

```
#----------------下面进行生存预测,用三种算法训练模型，选出最优精度的进行预测--------
#选择逻辑回归进行生存预测，计算精度
from sklearn.linear_model import LogisticRegression
#用逻辑回归算法创建模型,设置最大迭代次数max_iter=10000
model_logi=LogisticRegression(max_iter=10000)
#训练模型
model_logi.fit(train_data,train_label)
#对模型进行评估
predict_logi = model_logi.predict(test_data)
score_logi=model_logi.score(test_data,test_label)
print("逻辑回归算法训练模型的精度：",score_logi)
```

```
逻辑回归算法训练模型的精度： 0.8100558659217877
```

In [62]:

```
#选择KNN算法训练模型，计算精度
from sklearn.neighbors import KNeighborsClassifier
# k设置为6
model_knn= KNeighborsClassifier(n_neighbors=6)
model_knn.fit(train_data,train_label)
#模型评估
predict_knn = model_knn.predict(test_data)
score_knn=model_knn.score(test_data,test_label)
print("KNN算法训练模型的精度：",score_knn)
```

```
KNN算法训练模型的精度： 0.7541899441340782
```

In [63]:

```
#选择决策树训练模型，计算精度
from sklearn.tree import DecisionTreeClassifier
model_deci = DecisionTreeClassifier(criterion='entropy', max_depth=7, min_impurity_
model_deci.fit(train_data, train_label)
#模型评估
```

```
predict_deci = model_deci.predict(test_data)
score_deci=model_deci.score(test_data,test_label)
print("决策树算法训练模型的精度：",score_deci)
```

```
决策树算法训练模型的精度： 0.8100558659217877
```

In [64]:

```
#由上面三种预测的精度比较可以看出，逻辑回归算法训练的模型得到的精度最高。
#因此接下来的生存预测，就用刚刚训练出来的逻辑回归模型
```

In [65]:

```
#利用刚刚训练出来的模型进行生存预测
pre_label=model_logi.predict(predict_data)
#将结果转换为整型
predict_label=pre_label.astype(int)
#乘客ID
passenger_id=full.loc[sourceRow:,'PassengerId']
#生成预测生存情况和乘客ID的dataframe
preDf=pd.DataFrame({'PassengerId':passenger_id,'Survived':predict_label})
#输出预测值的信息
print(preDf.shape)
print(preDf.head())
#将结果保存至指定路径
preDf.to_csv('E:\\文档\\A-数据挖掘技术与应用\\实验五\\titanic_pre_sur.csv',index=Fa
print("预测完成！")
```

```
(418, 2)
     PassengerId  Survived
891          892         0
892          893         1
893          894         0
894          895         0
895          896         1
预测完成！
```