In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-py
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fi

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets prese
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
```

```
/kaggle/input/titanic/train.csv
/kaggle/input/titanic/test.csv
/kaggle/input/titanic/gender_submission.csv
```

七步流程

1. 定义问题
2. 获取训练数据和测试数据
3. 整理、准备、清洗数据
4. 分析、发现模式、探索数据
5. 建模、预测、求解问题
6. 可视化、报告、呈现问题求解步骤和最终结论
7. 提交

1. 定义问题

训练集包含一些乘客的样本并给出了是否生还的标记，训练一个模型判断测试集中的乘客是否生还

On **April 15, 1912**, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, **there weren’t enough lifeboats for everyone onboard**, resulting in **the death of 1502 out of 2224 passengers and crew**. While there was some element of luck involved in surviving, it seems **some groups of people were more likely to survive than others**.

其他背景信息：

- 1912年4月15日Titanic撞冰山后沉没，1502/2204乘客和船员死亡，大约为32%生还率
- 造成事故的其中一个原因是没有足够的救生艇
- 一些群体比其他人更容易生还，例如女人、孩子、上层阶级的人

七个常见的预处理

1. 分组：对样本进行分组(特征化)，探索群体之间的关系以及和问题目标的关系

2. 联系：探索哪些特征对问题的求解贡献最大，是否有统计意义上的相关性，特征之间是否有相关性
3. 转化：将特征转化为符合算法模型要求的类型
4. 填充：对缺失值进行填充
5. 修正：纠正某些特征不正确的值，或者排除具有这些不正确值的样本，甚至是丢弃哪些对任务没有贡献的特征。异常检测是一个方法。
6. 构造：构造新的特征
7. 作图

导包

In [2]:
```
# 数据整理和分析
import pandas as pd
import numpy as np
import random as rnd

# 可视化
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# 机器学习
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
```

2. 获取数据

In [3]:
```
train_df = pd.read_csv('../input/titanic/train.csv')
test_df = pd.read_csv('../input/titanic/test.csv')
combine = [train_df, test_df]
```

3. 探索性数据分析（EDA）

3.1 描述性数据分析

导入数据集后首要任务的是了解数据集，逐一回答如下问题：

1. 数据集包含哪些特征
2. 哪些特征是分类型(Categorical)特征，哪些是数值型(numeric)特征
3. 分类型特征里哪些属于标称型(枚举型)，哪些是序数型
4. 数值型特征里哪些属于区间标度，哪些属于比率标度
5. 数值型特征里哪些是连续的，哪些是离散的
6. 哪些特征的取值不规范
7. 哪些特征有缺失值
8. 哪些特征有离群值(异常值、噪声)

9. 存储这些特征的数据类型是否合适
10. 数据集中是否有重复的样本
11. 数据集中各个特征的分布如何，是否类别不平衡

数据集中包含哪些特征

In [4]:
```
print(train_df.columns.values)
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

字段名	定义	值
PassengerId	乘客ID号	
Survived	是否生还	0 = No, 1 = Yes
Pclass	船票类型	1 = 1st, 2 = 2nd, 3 = 3rd
Name	姓名	
Sex	性别	
Age	年龄	
SibSp	船上兄弟姐妹或配偶的数量	
Parch	船上父母或子女的数量	
Ticket	船票号	
Fare	票价	
Cabin	舱位号	
Embarked	登船港口	C = Cherbourg, Q = Queenstown, S = Southampton

哪些特征是分类型(定性)的

不仅应该识别出哪些是分类型特征，最好能细分为标称型、序数型

- 标称型特征有：PassengerId, Survived, Name, Sex, Ticket, Carbin, Embarked
- 序数型特征有：Pclass

哪些特征是数值型(定量)的

根据取值还应该区分连续的和离散的

- 连续型特征有：Age, Fare
- 离散型特征有：SibSp, Parch

哪些特征是混合类型的

- Ticket既有纯数值又有字符加数值

例如：A/5 21171

- Cabin字符加数值

例如：C123

这些特征可能后续需要进行纠正

In [5]:
```
```

```
train_df.head()
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

哪些特征的值不规范

- Name特征可能具有错误值

例如：Futrelle, Mrs. Jacques Heath (Lily May Peel)，有多余的括号，而且有称谓信息Mrs

输入错误、大小写错误、圆括号、引号等

后续可能需要进行修正

In [6]:
```
train_df.tail()
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Er
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Er
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	

哪些特征具有缺失值

这些缺失值后续需要处理

- 训练集：Cabin、Age、Embarked有缺失值
- 测试集：Cabin、Age、Fare有缺失值

In [7]:
```
train_df.isnull().sum()
```

Out[7]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [8]:
```
test_df.isnull().sum()
```

Out[8]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

哪些特征具有异常值(离群值)

练习1：采取适当的异常值处理方法

参考：https://www.kaggle.com/yassineghouzam/titanic-top-4-with-ensemble-modeling

各个特征的数据类型是什么

后续需要根据特征存储的数据类型和特征真正的类型进行类型转换

- 七个特征是整型或浮点型
- 五个特征是字符串

In [9]:
```
train_df.info()
print('_'*40)
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

数值型特征的经验分布如何

数据集的经验分布有助于我们对数据集进行初步观察，判断经验分布能否代表真实分布

In [10]:
```
train_df.describe(percentiles=[.61, .62, .68, .69, .75, .8, .99])
```

Out[10]:

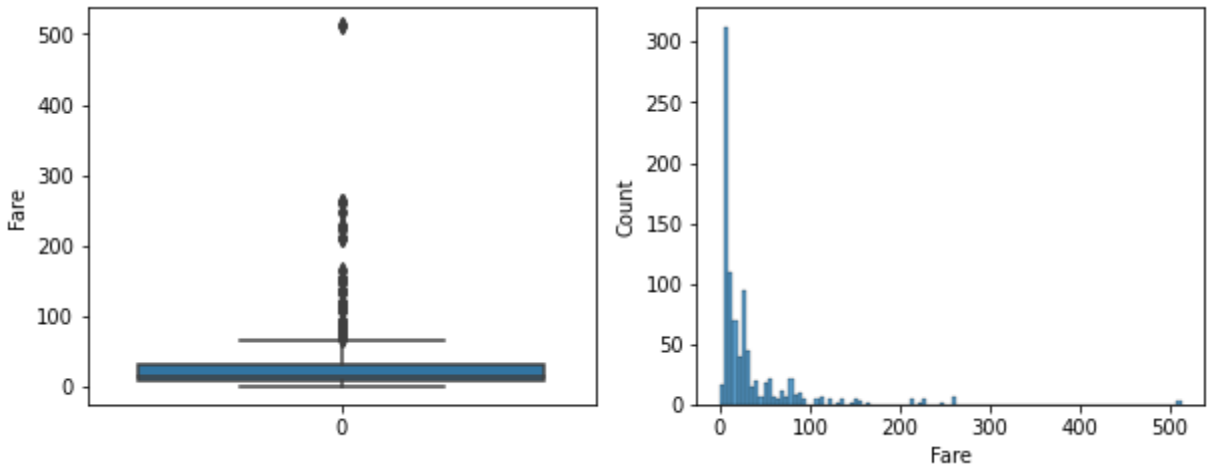
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
61%	543.900000	0.000000	3.000000	32.000000	0.000000	0.000000	23.225000
62%	552.800000	1.000000	3.000000	32.000000	0.000000	0.000000	24.150000
68%	606.200000	1.000000	3.000000	35.000000	0.000000	0.000000	26.307500
69%	615.100000	1.000000	3.000000	35.000000	1.000000	0.000000	26.550000
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
80%	713.000000	1.000000	3.000000	41.000000	1.000000	1.000000	39.687500

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
99%	882.100000	1.000000	3.000000	65.870000	5.000000	4.000000	249.006220
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

从百分位数来看， Fare 的分布非常不平衡，再把它的箱线图和柱形图画出来看看

In [11]:
```
plt.figure(figsize=[10, 8])
plt.subplot(221)
sns.boxplot(data=train_df['Fare'])
plt.ylabel('Fare')
plt.subplot(222)
sns.histplot(train_df['Fare'])
plt.xlabel('Fare')
```

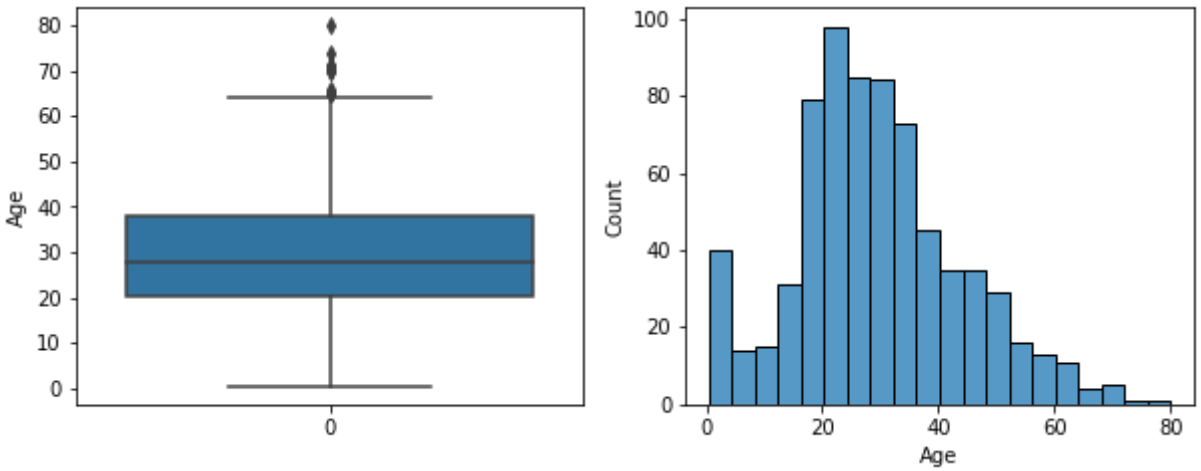
Out[11]: Text(0.5, 0, 'Fare')



顺便也将 Age 的分布图画出来看看

In [12]:
```
plt.figure(figsize=[10, 8])
plt.subplot(221)
sns.boxplot(data=train_df['Age'])
plt.ylabel('Age')
plt.subplot(222)
sns.histplot(train_df['Age'])
plt.xlabel('Age')
```

Out[12]: Text(0.5, 0, 'Age')



除了婴儿的数量较多之外，没有什么特别之处

关于数值型特征的一些结论

- 数据集中包含891个样本，问题背景里说Titanic有2224个人，数据集只包含了真实数据约40%个体
- 数据集里的样本的生还率是38%，问题背景里说真实的生还率为32%
- 超过75%的样本没有父母或孩子同行
- 接近30%的样本有兄弟姐妹或配偶同行
- 票价分布不均，<1%样本的票价高达512美元
- 老年人(65-80岁)的比例小于1%

分类型特征分布如何

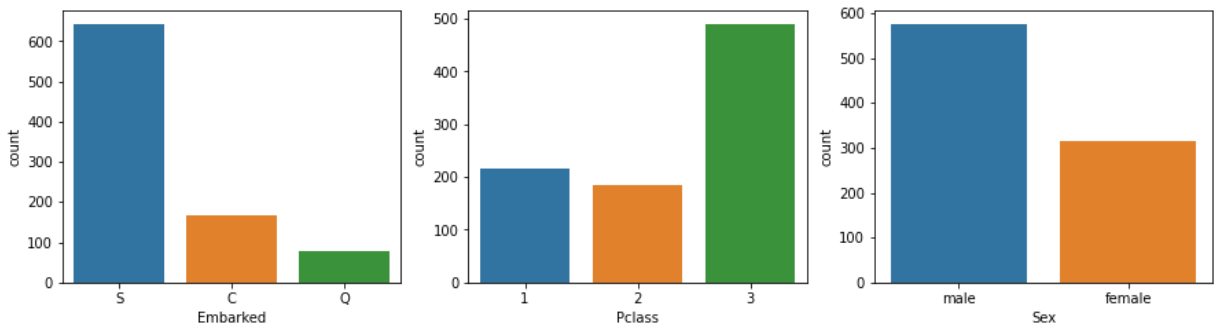
In [13]:
```
train_df.describe(include='O')
```

Out[13]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Palsson, Miss. Stina Viola	male	CA. 2343	B96 B98	S
freq	1	577	7	4	644

In [14]:
```
plt.figure(figsize=[15, 8])
plt.subplot(231)
sns.countplot(x='Embarked', data=train_df)
plt.subplot(232)
sns.countplot(x='Pclass', data=train_df)
plt.subplot(233)
sns.countplot(x='Sex', data=train_df)
```

Out[14]: <AxesSubplot:xlabel='Sex', ylabel='count'>



关于分类型特征的一些结论

- Names没有重复值，没有人重名
- 577/891(65%)是男性
- Cabin有不少重复值，有不少乘客共享舱位
- Embarked只有3个值，S港口最多人登船
- Ticket有不少重复值，有不少乘客共享船票?

3.2 初步想法与一些假设

对数据集的情况有了基本了解之后，根据发现的问题，我们会在这一阶段提出一些初步想法、一些先验的假设，以及给出处理方案

按照处理方案一一列举如下：

填充

年龄显然是与生还有很大关系的（尊老爱幼）

我们假设 Age 和 Embarked 都与生还有紧密联系，接下来对这两个特征进行填充

纠正

1. Ticket 有22%重复值，我们假设船票号与生还没有联系。删除这个特征。
2. Cabin 在训练集和测试集里都有太多缺失值。删除这个特征。
3. PassenerId 是乘客ID号，每个样本都不一样，对于预测生还没有帮助。删除这个特征。
4. Name 是乘客名称，名字本身与生还没有联系，但里面包含了称谓信息，称谓信息有可能对我们的预测有帮助。提取称谓信息构造新特征并删除这个特征。

构造

1. 构造特征 Family 表示乘客的家庭成员数量。根据 Parch 和 SibSp 可以计算出每个乘客家庭成员的数量。直觉上来看，有家室的乘客和单身的乘客这两个群体，在遇到突发事件时表现是不一样的，因此会影响是否生还。这个特征的构造是非常合理的。
2. 将年龄 Age 进行分箱。直觉上来看，两个年龄相差不大的乘客，可能在生还的表现上没有太大的差异；但是年龄相差较大的乘客，例如中年人和青少年，在生还的表现上肯定是有差异的。在这种情况下使用分箱这一技巧，可以减少计算量，提高模型的准确度。而且如果有年龄虚报的情况，分箱可以纠正这种异常（虚报年龄不会和真实年龄相差太远），提高模型的鲁棒性。
3. 将船票的价格 Fare 进行分箱。和 Age 同理，而且 Fare 后1%的值非常大，分箱可以提高鲁棒性。
4. Name 字段中有些包含了称谓，可以构造 Title 特征表示样本的称谓。

分组（特征化）

根据问题背景里的描述，女性群体(Sex=female)、年轻群体(Age<=?)、高收入群体(Pclass=1)更可能生还

将上述的初步想法、假设和设想的处理方案整理如下：

字段名	定义	处理方法
PassengerId	乘客ID号	删除
Survived	是否生还	
Pclass	船票类型	
Name	姓名	删除
Sex	性别	
Age	年龄	填充缺失值
SibSp	船上兄弟姐妹或配偶的数量	构造新特征
Parch	船上父母或子女的数量	构造新特征

字段名	定义	处理方法
Ticket	船票号	删除
Fare	票价	
Cabin	舱位号	删除
Embarked	登船港口	填充缺失值

在正式敲定预处理方案之前，还需要对上述的猜想进行验证。经过验证后我们才有充分的理由实施对应的预处理方案

接下来将会用其他的分析方法探索数据集，提出更多的猜想以及逐一验证我们的猜想。

3.3 分析特征与标签的关系

3.3.1 通过groupby分析分类型特征

先前关于分类型特征的猜想有如下几条：

- `Pclass` ： 船票类型代表了乘客的收入水平，我们认为高收入群体更可能生还
- `Sex` ： 我们认为女性群体更可能生还
- `Embarked` ： 我们认为登船港口可能与是否生还有关
- `family` ： 按照 `SibSp` 和 `Parch` 将数据集单独groupby

Pclass

In [15]:
```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_valu
```

Out[15]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

发现Pclass=1的群体的确拥有更高的生还率，验证了我们先前的假设

Sex

In [16]:
```
train_df[['Sex', 'Survived']].groupby(['Sex'], as_index=False).mean().sort_values(by=
```

Out[16]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

发现女性群体生还率高达74%，男性群体生还率只有不到19%，验证了先前的假设

Embarked

In [17]:
```
train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_
```

Out[17]:

	Embarked	Survived

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.336957

C港口登船的乘客有更高的生还率，验证了先前的假设

SibSp

In [18]:
```
train_df[['SibSp', 'Survived']].groupby(['SibSp'], as_index=False).mean().sort_values
```

Out[18]:

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

Parch

In [19]:
```
train_df[['Parch', 'Survived']].groupby(['Parch'], as_index=False).mean().sort_values
```

Out[19]:

	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

单独看这两个特征都和生还率没有直接关系

3.3.2 通过条形图探索数值型特征

先前对数值型特征的猜想有：

- Age ： 年龄与生还有密切联系

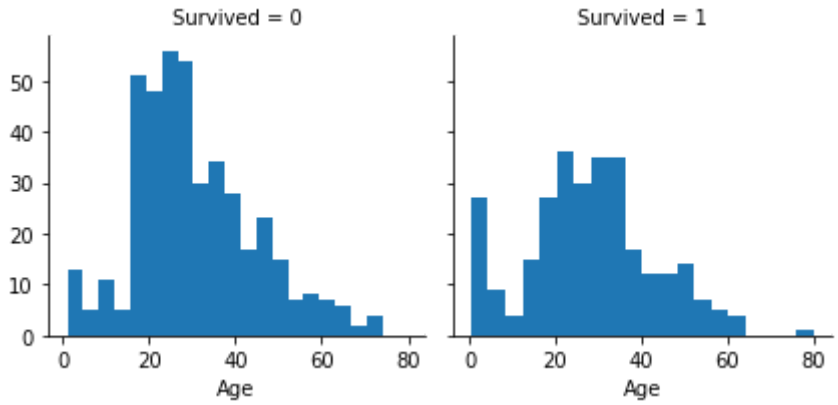
首先对年龄 Age 和生还 Survived 的关系进行探索

Age

In [20]:

```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x7fe1b09c68d0>



观察

1. 婴儿群体(Age <= 4)生还率较高
2. 最老的乘客(Age = 80)生还了
3. 大量青壮年乘客(15 <= Age <= 25)死亡了
4. 15-35岁的乘客占比最多

关于年龄的结论

基于上述这四条观察，我们证实了年龄与生还有密切联系这一猜想，先前关于年龄的几个想法是合理的

- 模型需要考虑 Age 特征
- 填充 Age 特征的缺失值
- 对年龄进行分箱

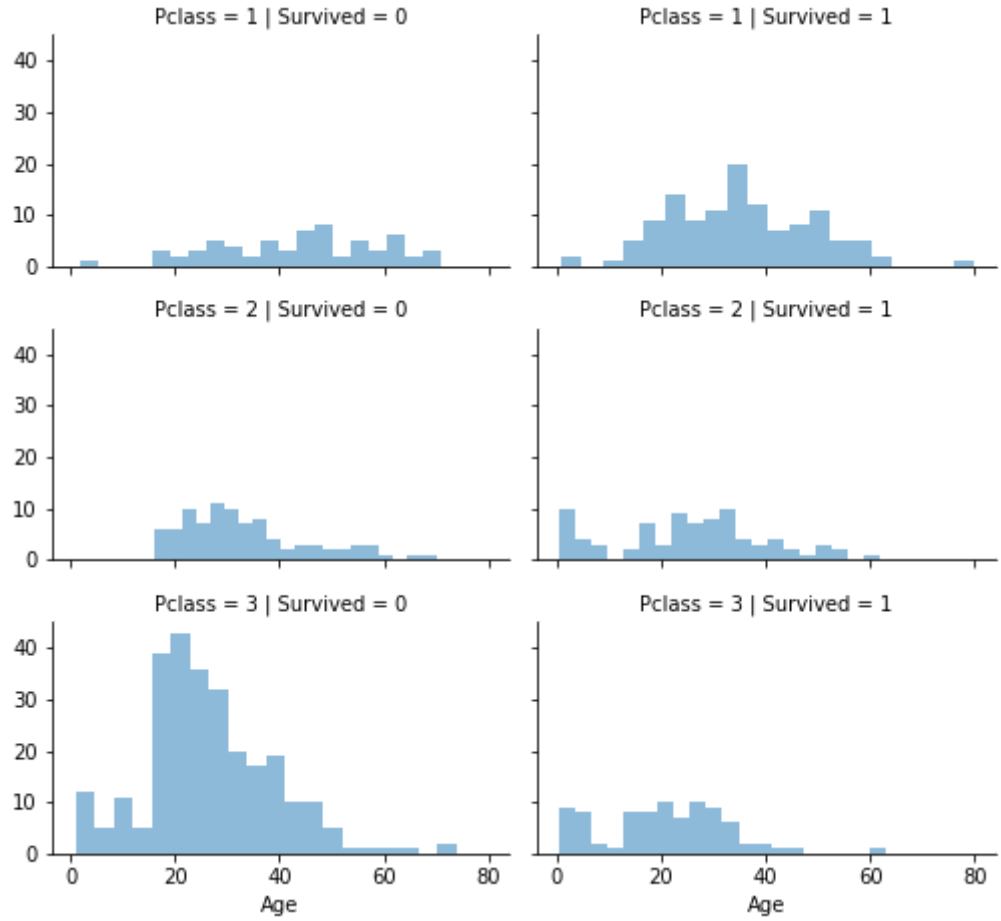
3.3.3 使用条形图结合多个特征分析

我们将 Pclass 和 Age 结合在一起考虑与生还的关系，同时也探索不同收入水平的人群的年龄分布情况

Pclass和Age

In [21]:
```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=0.5, bins=20)
grid.add_legend()
```

Out[21]: <seaborn.axisgrid.FacetGrid at 0x7fe1b0884cd0>



观察

- Pclass=3 的乘客死亡率是最高的， Pclass=1 的乘客死亡率是最低的，而且差异相当明显，再次验证了先前关于 Pclass 与生还率有关的猜想
- Pclass=2 or 3 的婴儿大多数都生还了（ Pclass=1 的婴儿数量太少，不考虑），对年龄进行分箱的确有助于分析
- Pclass 不同的乘客群体年龄分布不同，其中 Pclass=3 的乘客里青壮年特别多

关于乘客收入水平的结论

不同收入水平的乘客群体生还率差异较大，特别是低收入水平的乘客生还率最低，我们的模型需要考虑 Pclass 特征

3.3.4 通过折线图结合多个特征分析

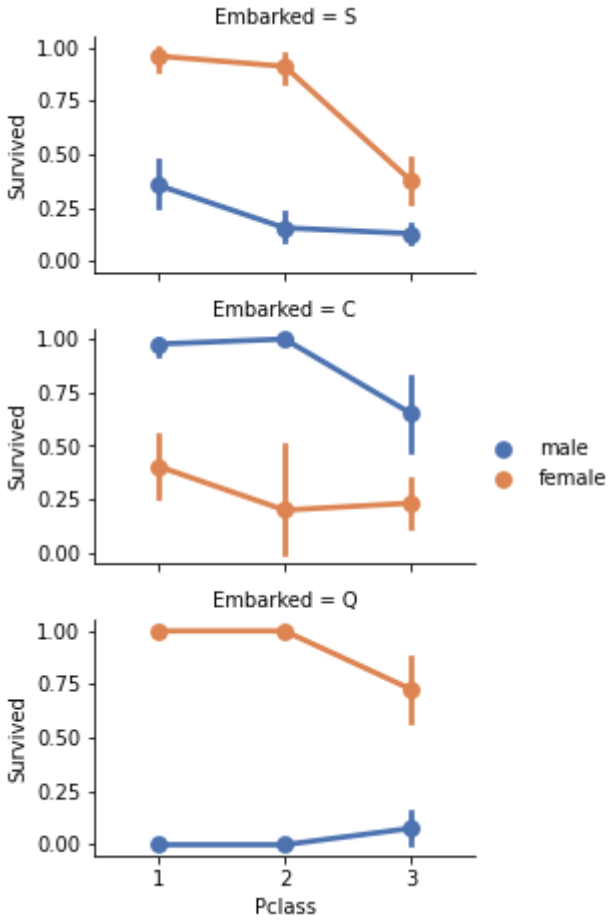
我们将登船港口 Embarked 、船票类型 Pclass 、性别 Age 综合在一起考察生还率

Embarked、Pclass和Age

In [22]:
```
grid = sns.FacetGrid(train_df, row='Embarked', height=2.2, aspect=1.6)
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/axisgrid.py:643: UserWarning: Using the
pointplot function without specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
/opt/conda/lib/python3.7/site-packages/seaborn/axisgrid.py:648: UserWarning: Using the
pointplot function without specifying `hue_order` is likely to produce an incorrect pl
ot.
  warnings.warn(warning)
```

Out[22]: <seaborn.axisgrid.FacetGrid at 0x7fe1b051eed0>



观察

- 在S港口和Q港口登船的女性普遍比男性生还率更高，但是在C港口登船的男性反而生还率更高
- 在Q港口登船的低收入男性反而比高收入男性生还率高

结论

先前认为的女性普遍比男性生还率高不完全正确，高收入群体普遍比低收入群体生还率高也不完全正确

生还率还要受到登船港口的影响，不同的登船港口对这两个趋势影响很大

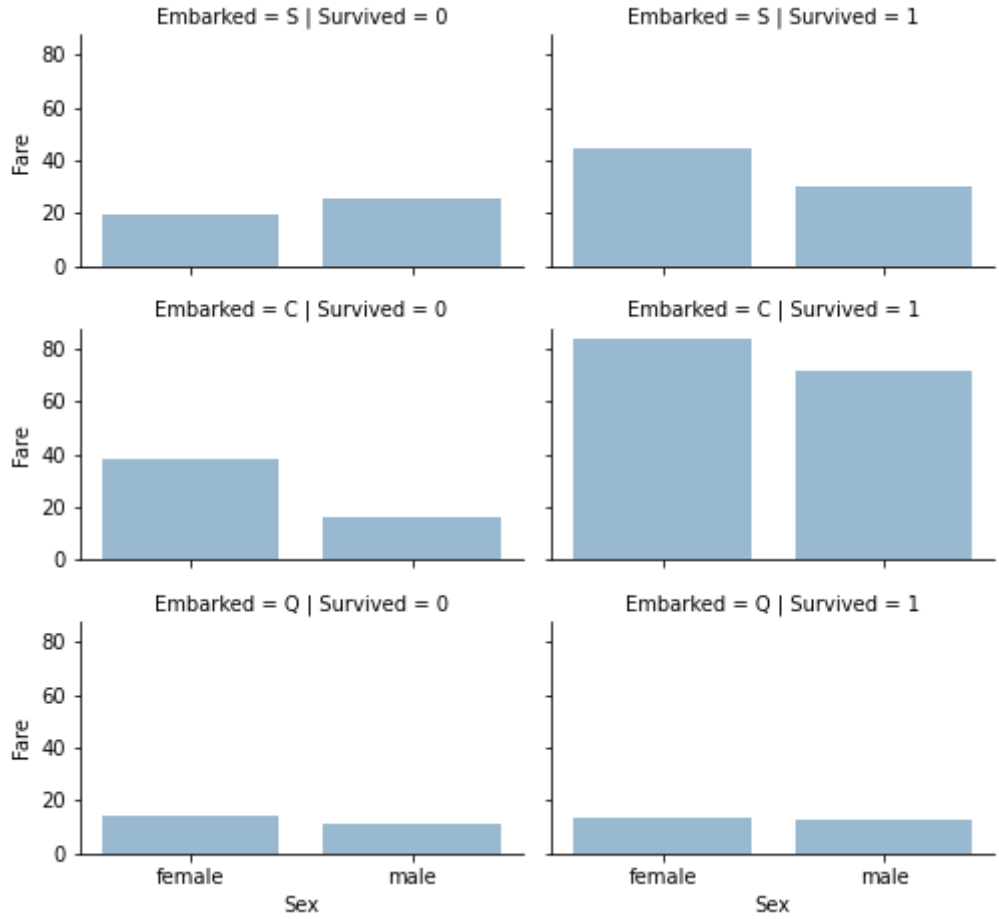
- 需要填充 Embarked 特征的缺失值，并且模型也需要考虑这个特征

我们将 Embarked 登船港口、 Sex 性别、 Fare 票价综合起来分析它们与生还率的联系

In [23]:
```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', height=2.2, aspect=1.6
grid.map(sns.barplot, 'Sex', 'Fare', alpha=0.5, ci=None)
grid.add_legend()
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/axisgrid.py:643: UserWarning: Using the
barplot function without specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x7fe1b03a5dd0>



观察

- 票价越高的乘客生还率越高
- Q港口登船的乘客票价普遍较低，而且生还率似乎与票价无关

关于票价的结论

- 模型需要考虑 `Fare` 特征
- 需要将票价进行分箱

3.4 分析特征与特征之间的关系

通常使用相关系数和热力图来探索特征之间的相关性，删除相关程度高的特征也许能提高模型的预测能力

练习2：使用相关系数、热力图探索特征之间的相关性

练习2 plus*：根据相关分析的结果进行特征筛选，看看是否能提高模型的预测能力

参考：https://www.kaggle.com/gunesevitan/titanic-advanced-feature-engineering-tutorial#1.-Exploratory-Data-Analysis

4. 数据预处理

在EDA阶段我们已经对整个数据集进行了细致的考察，探索了各个特征之间的联系、与预测目标（生还）的关系，提出了许多初步想法和猜想，并且对这些想法和猜想进行了简单的验证

在数据预处理阶段，我们将会遵循先前已经验证的想法和猜想，对数据集进行处理。在这一阶段我们还会不断提出其他猜想、验证并且处理

字段名	定义	处理方法
PassengerId	乘客ID号	在训练集中删除
Survived	是否生还	
Pclass	船票类型	
Name	姓名	构造Title特征，删除
Sex	性别	
Age	年龄	填充缺失值,分箱
SibSp	船上兄弟姐妹或配偶的数量	构造新特征FamilySize，删除
Parch	船上父母或子女的数量	构造新特征FamilySize，删除
Ticket	船票号	删除
Fare	票价	填充缺失值，分箱
Cabin	舱位号	删除
Embarked	登船港口	填充缺失值，转为One-Hot编码
Title	称谓	由Name构造，转为One-Hot编码
FamilySize	家庭大小	由SibSp和Parch构造
IsAlone	是否单身	由SibSp和Parch构造

4.1 删除特征

根据先前的想法，我们需要删除特征 `Cabin` 、 `Ticket` 、 `Name` 和 `PassengerId`

在删除之前，我们可以先验证这几个特征和乘客生还与否确实没有关联，然后再删

这里为了省略篇幅，只在删除 `Name` 之前验证它和生还确实没有关联， `Cabin` 和 `Ticket` 直接删除

乘客ID号 `PassengerId` 需要在训练集中删除，在测试集中保留（为了提交）

Tips

- 所有操作必须对训练集和测试集同时处理
- 数据预处理阶段建议从删除特征开始，这样可以减少数据量，减少后续处理的计算量

删除 `Cabin` 和 `Ticket`

In [24]:
```
train_df = train_df.drop(columns=['Ticket', 'Cabin'])
test_df = test_df.drop(['Ticket', 'Cabin'], axis=1)
```

我们没有分析 `Cabin` 和生还率的相关性，实际上 `Cabin` 字段里包含了舱位类型信息

练习3：分析 `Cabin` 和 `Survived` 的关系，提取舱位类型信息，构造新特征

参考：https://zhuanlan.zhihu.com/p/50194676

在训练集中删除 `PassengerId`

In [25]:
```
train_df = train_df.drop(columns=['PassengerId'])
combine = [train_df, test_df]
```

4.2 构造新特征

注意到 Name 特征中包含了样本的称谓，某种程度上反映了样本的身份地位

虽然姓名本身和生还率没有直接关系，但身份地位有可能与生还有关，因此我们在删除 Name 特征之前，可以先把样本的身份地位提取出来，构造新特征 Title 然后再删去 Name 特征

从 Name 中提取称谓信息 Title

称谓通常都有一个句点.和姓名分隔开，例如Mr. Mrs.等，可以根据这个特点构建正则表达式，提取样本的称谓

如果不懂正则表达式也没关系，可以用 apply() 方法配合 split() 方法将字段按句点.分隔开，然后提取句点之前的部分

In [26]:
```
for dataset in combine:
    dataset['Title'] = dataset['Name'].str.extract('([A-Za-z]+)\.', expand=False)
```

许多称谓和性别是直接相关的，我们可以通过 Title 和 Sex 的交叉表来观察

In [27]:
```
pd.crosstab(train_df['Title'], train_df['Sex'])
```

Out[27]:

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6

Sex	female	male
Title		
Sir	0	1

查阅资料后发现不少问题：

- Ms、Miss和Mlle是一个意思
- Mme和Mrs是一个意思
- 有许多称谓只有少数几个样本(Capt, Col, Coutess, Don, Jonkheer, Dr, Lady, Major, Mlle, Mme, Ms, Rev, Sir)

接下来我们把一样意思的称谓用同一个称谓代替，只有少量样本的称谓统一称为 Rare

In [28]:
```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess', 'Capt', 'Col', 'D
                                                 'Dr', 'Major', 'Rev', 'Sir', 'Jonkhee
                                                'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
```

通过groupby分析称谓和生存率之间的关系

In [29]:
```
train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

Out[29]:

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

发现的确称谓与生还存在某种联系

- 女性称谓存活率更高，男性称谓存活率更低
- 其他称谓的存活率较低，只有34%
- Master的存活率约57%

删除 Name 特征

In [30]:
```
train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(columns=['Name'])
combine = [train_df, test_df]
train_df.head()
```

Out[30]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	male	22.0	1	0	7.2500	S	Mr
1	1	1	female	38.0	1	0	71.2833	C	Mrs
2	1	3	female	26.0	0	0	7.9250	S	Miss

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
3	1	1	female	35.0	1	0	53.1000	S	Mrs
4	0	3	male	35.0	0	0	8.0500	S	Mr

构造 FamilySize 和 IsAlone 特征

将 Parch 和 SibSp 结合，构造 FamilySize 特征

按 FamilySize 区分是否单身

In [31]:
```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1 # 别忘了加上自己
```

用groupby看看新构造的特征是否和生存有关

In [32]:
```
train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().s
```

Out[32]:

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

观察

- 家庭大小太大的人(FamilySize >= 5)生还率都比较低
- 家庭大小适中的人(FamilySize = 2, 3, 4)生还率较高
- 单身的人(FamilySize = 1)生还率也很低

In [33]:
```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
train_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

Out[33]:

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

单身的乘客生还率比较低，非单身的乘客生还率达到50%

根据上面的观察，我们选择保留 IsAlone 和 FamilySize 特征，删去两个原始特征

In [34]:
```
train_df = train_df.drop(['Parch', 'SibSp'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

Out[34]:

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	FamilySize	IsAlone
0	0	3	male	22.0	7.2500	S	Mr	2	0
1	1	1	female	38.0	71.2833	C	Mrs	2	0
2	1	3	female	26.0	7.9250	S	Miss	1	1
3	1	1	female	35.0	53.1000	S	Mrs	2	0
4	0	3	male	35.0	8.0500	S	Mr	1	1

4.3 转换数据类型

为了后续输入到模型进行预测，需要将字符串类型的字段转换成数值类型

使用get_dummies()将 `Title` 改成one-hot编码

In [35]:
```
train_df = train_df.join(pd.get_dummies(train_df['Title']))
train_df = train_df.drop(columns=['Title'])
test_df = test_df.join(pd.get_dummies(test_df['Title']))
test_df = test_df.drop(columns=['Title'])
combine = [train_df, test_df]
```

In [36]:
```
train_df
```

Out[36]:

	Survived	Pclass	Sex	Age	Fare	Embarked	FamilySize	IsAlone	Master	Miss	Mr	M
0	0	3	male	22.0	7.2500	S	2	0	0	0	1	
1	1	1	female	38.0	71.2833	C	2	0	0	0	0	
2	1	3	female	26.0	7.9250	S	1	1	0	1	0	
3	1	1	female	35.0	53.1000	S	2	0	0	0	0	
4	0	3	male	35.0	8.0500	S	1	1	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	13.0000	S	1	1	0	0	0	
887	1	1	female	19.0	30.0000	S	1	1	0	1	0	
888	0	3	female	NaN	23.4500	S	4	0	0	1	0	
889	1	1	male	26.0	30.0000	C	1	1	0	0	1	
890	0	3	male	32.0	7.7500	Q	1	1	0	0	1	

891 rows × 13 columns

将 Sex 特征转换为数值类型，用0表示男性，1表示女性

In [37]:
```
for dataset in combine:
```

```
    dataset['Sex'] = dataset['Sex'].map({
        'female': 1,
        'male': 0,
    }).astype(int)
```

将 Embarked 特征转换为数值类型，由于这个字段有缺失值，需要先填充缺失值再转换数据类型

4.4 缺失值处理

填充分类型特征—— Embarked 字段

因为这个字段只有两个缺失值，采取简单的填充方法或者干脆删除缺失的两条数据都可以

这里我们采取众数填充的方法

In [38]:
```
freq_port = train_df['Embarked'].dropna().mode()[0]
print("填充的港口是:{}".format(freq_port))
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)
```

```
填充的港口是:S
```

填充完之后把 Embarked 转换one-hot编码

In [39]:
```
train_df = train_df.join(pd.get_dummies(train_df['Embarked']))
train_df = train_df.drop(columns=['Embarked'])
test_df = test_df.join(pd.get_dummies(test_df['Embarked']))
test_df = test_df.drop(columns=['Embarked'])
combine = [train_df, test_df]
```

In [40]:
```
train_df.head()
```

Out[40]:

	Survived	Pclass	Sex	Age	Fare	FamilySize	IsAlone	Master	Miss	Mr	Mrs	Rare	C	Q
0	0	3	0	22.0	7.2500	2	0	0	0	1	0	0	0	0
1	1	1	1	38.0	71.2833	2	0	0	0	0	1	0	1	0
2	1	3	1	26.0	7.9250	1	1	0	1	0	0	0	0	0
3	1	1	1	35.0	53.1000	2	0	0	0	0	1	0	0	0
4	0	3	0	35.0	8.0500	1	1	0	0	1	0	0	0	0

填充数值型特征—— Age 字段

有缺失值需要处理的字段包括 Age 和 Embarked ，首先对 Age 字段进行缺失值填充

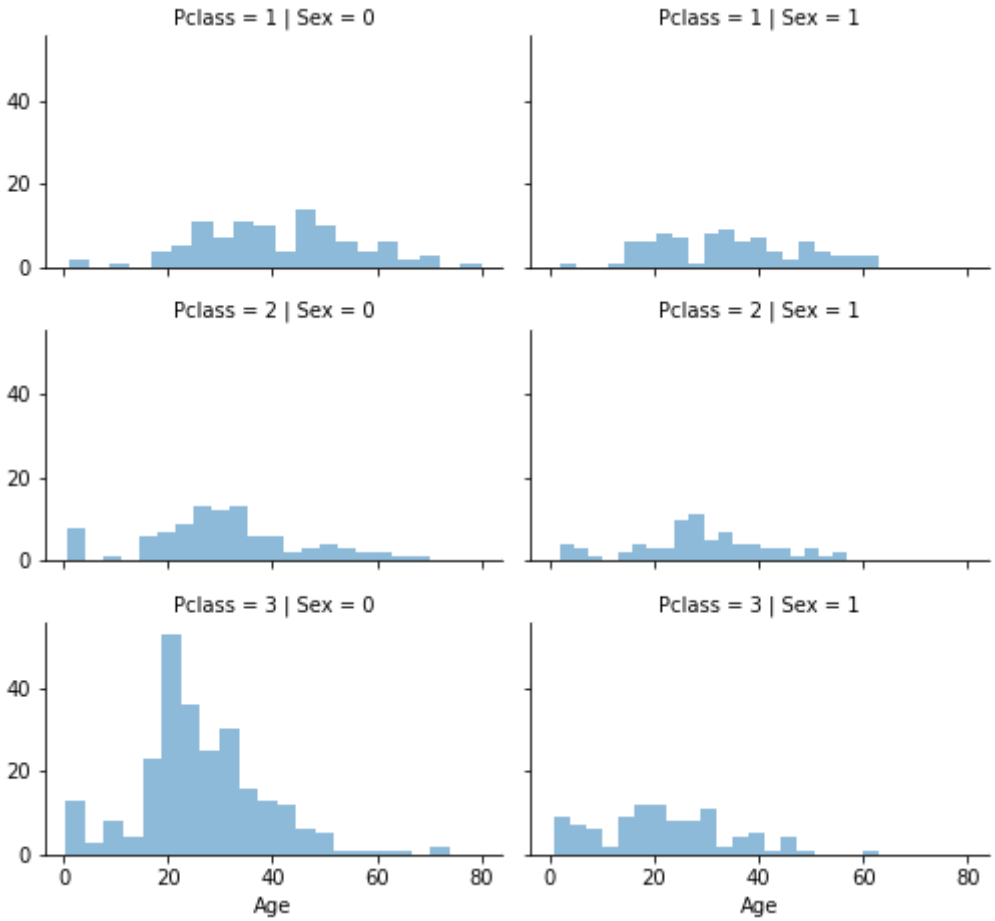
简单的填充方法可以填充中位数、平均值、随机数等，更精确的方法是基于其他相关的特征来填充

这里我们根据 Pclass 和 Gender 对 Age 进行缺失值填充（更高级的有基于其他机器学习算法的缺失值填充）

Pclass 有三种， Gender 有两种，它们的组合一共有六种情况，我们根据这六种群体的中位数对 Age 进行填充

In [41]:
```
grid = sns.FacetGrid(train_df, row='Pclass', col='Sex', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```

Out[41]: <seaborn.axisgrid.FacetGrid at 0x7fe1b0050490>



In [42]:
```
guess_ages = np.zeros((2, 3))
for dataset in combine:
    # 计算填充值
    for i in range(2):
        for j in range(3):
            age_guess = dataset[(dataset['Sex'] == i) & (dataset['Pclass'] == j+1)]['
            guess_ages[i, j] = round(age_guess) # 四舍五入

    # 填充
    for i in range(2):
        for j in range(3):
            dataset.loc[(dataset['Age'].isnull()) & (dataset['Sex'] == i) &
                        (dataset['Pclass'] == j+1), 'Age'] = guess_ages[i, j]

    dataset['Age'] = dataset['Age'].astype(int)
```

In [43]:
```
train_df.head()
```

Out[43]:

	Survived	Pclass	Sex	Age	Fare	FamilySize	IsAlone	Master	Miss	Mr	Mrs	Rare	C	Q
0	0	3	0	22	7.2500	2	0	0	0	1	0	0	0	0
1	1	1	1	38	71.2833	2	0	0	0	0	1	0	1	0
2	1	3	1	26	7.9250	1	1	0	1	0	0	0	0	0

	Survived	Pclass	Sex	Age	Fare	FamilySize	IsAlone	Master	Miss	Mr	Mrs	Rare	C	Q
3	1	1	1	35	53.1000	2	0	0	0	0	1	0	0	0
4	0	3	0	35	8.0500	1	1	0	0	1	0	0	0	0

填充测试集中的 Fare 字段

因为测试集中 Fare 字段只有1个缺失值，使用简单的中位数填充即可

In [44]:
```
test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
```

4.5 其他处理

根据前面的观察和猜想，我们认为对年龄进行分箱有助于提高模型的准确率

将 Age 字段按等间距分箱，分成5份，分别统计生还率

In [45]:
```
train_df['AgeBand'] = pd.cut(train_df['Age'], 5)
train_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_va
```

Out[45]:

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

将数值型属性 Age 按照分箱的结果转换成序数性属性，删除 AgeBand 字段

In [46]:
```
for dataset in combine:
    dataset.loc[dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] <= 32) & (dataset['Age'] > 16), 'Age'] = 1
    dataset.loc[(dataset['Age'] <= 48) & (dataset['Age'] > 32), 'Age'] = 2
    dataset.loc[(dataset['Age'] <= 64) & (dataset['Age'] > 48), 'Age'] = 3
    dataset.loc[dataset['Age'] > 64, 'Age'] = 4
train_df = train_df.drop(['AgeBand'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

Out[46]:

	Survived	Pclass	Sex	Age	Fare	FamilySize	IsAlone	Master	Miss	Mr	Mrs	Rare	C	Q
0	0	3	0	1	7.2500	2	0	0	0	1	0	0	0	0
1	1	1	1	2	71.2833	2	0	0	0	0	1	0	1	0
2	1	3	1	1	7.9250	1	1	0	1	0	0	0	0	0
3	1	1	1	2	53.1000	2	0	0	0	0	1	0	0	0
4	0	3	0	2	8.0500	1	1	0	0	1	0	0	0	0

对 Fare 特征进行分箱

In [47]:
```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort
```

Out[47]:

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

In [48]:
```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]
```

预处理之后的数据集

In [49]:
```
train_df.head()
```

Out[49]:

	Survived	Pclass	Sex	Age	Fare	FamilySize	IsAlone	Master	Miss	Mr	Mrs	Rare	C	Q	S
0	0	3	0	1	0	2	0	0	0	1	0	0	0	0	1
1	1	1	1	2	3	2	0	0	0	0	1	0	1	0	0
2	1	3	1	1	1	1	1	0	1	0	0	0	0	0	1
3	1	1	1	2	3	2	0	0	0	0	1	0	0	0	1
4	0	3	0	2	1	1	1	0	0	1	0	0	0	0	1

5. 建立模型，预测求解

重新明确一下数据挖掘任务：我们希望通过乘客的信息预测该乘客是否生还

这是一个分类/回归任务。可以使用有监督的机器学习算法训练模型进行预测

适用于这一任务的模型包括但不限于：

- Logistic Regression
- 支持向量机
- k-近邻
- 朴素贝叶斯分类器
- 决策树
- 随机森林
- 感知机
- 神经网络

首先不进行超参数调节，直接对各个算法使用默认的参数拟合整个训练集，计算训练集上的准确率。再通过10折交叉验证来估计模型的泛化能力，

先直观感受一下树模型和其他简单的模型对数据集拟合能力的差异

In [50]:
```
X_train = train_df.drop('Survived', axis=1)
Y_train = train_df['Survived']
X_test = test_df.drop('PassengerId', axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
```

Out[50]: ((891, 14), (891,), (418, 14))

5.1 Logistic Regression

In [51]:
```
LR = LogisticRegression()
LR.fit(X_train, Y_train)
Y_pred_LR = LR.predict(X_test)
acc_LR = round(LR.score(X_train, Y_train) * 100, 2)
print("LR在训练集上的准确率是：{}%".format(acc_LR))

acc_LR_cv = round(cross_val_score(LR, X_train, Y_train, cv=10, scoring='accuracy').me
print("LR在训练集上的cv准确率是：{}%".format(acc_LR_cv))
```

```
LR在训练集上的准确率是：83.28%
LR在训练集上的cv准确率是：82.94%
```

因为LR的可解释性很强，每个特征的权重代表了对预测目标的贡献情况

通常建议在构建baseline的时候使用Logistic Regression，可以通过观察各个特征的权重来判断特征的重要性

同时也可以验证我们先前的想法和猜想，还可以评估我们构建的新特征是否真的对预测目标有帮助

In [52]:
```
coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df['Correlation'] = pd.Series(LR.coef_[0])
coeff_df.sort_values(by='Correlation', ascending=False)
```

Out[52]:

	Feature	Correlation
6	Master	1.611252
1	Sex	1.431155
9	Mrs	0.664695
3	Fare	0.179510
11	C	0.162093
7	Miss	0.088824
12	Q	0.075472
13	S	-0.236502
5	IsAlone	-0.310208
2	Age	-0.464871

	Feature	Correlation
4	FamilySize	-0.519000
10	Rare	-0.987935
0	Pclass	-1.018277
8	Mr	-1.375772

5.2 Support Vector Machines

In [53]:
```
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred_svc = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
print("svc在训练集上的准确率是：{}%".format(acc_svc))

acc_svc_cv = round(cross_val_score(svc, X_train, Y_train, cv=10, scoring='accuracy').
print("svc在训练集上的cv准确率是：{}%".format(acc_svc_cv))
```

```
svc在训练集上的准确率是：83.5%
svc在训练集上的cv准确率是：83.28%
```

5.3 KNN

In [54]:
```
knn = KNeighborsClassifier(n_neighbors = 9)
knn.fit(X_train, Y_train)
Y_pred_knn = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
print("KNN在训练集上的准确率是：{}%".format(acc_knn))

acc_knn_cv = round(cross_val_score(knn, X_train, Y_train, cv=10, scoring='accuracy').
print("KNN在训练集上的cv准确率是：{}%".format(acc_knn_cv))
```

```
KNN在训练集上的准确率是：84.74%
KNN在训练集上的cv准确率是：82.72%
```

5.4 Gaussian Naive Bayes

In [55]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred_gaussian = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
print("Gaussian NB在训练集上的准确率是：{}%".format(acc_gaussian))

acc_gaussian_cv = round(cross_val_score(gaussian, X_train, Y_train, cv=10, scoring='a
print("Gaussian NB在训练集上的cv准确率是：{}%".format(acc_gaussian_cv))
```

```
Gaussian NB在训练集上的准确率是：79.91%
Gaussian NB在训练集上的cv准确率是：79.02%
```

5.5 Perceptron

In [56]:
```
perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred_perceptron = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
```

```
print("Perceptron在训练集上的准确率是：{}%".format(acc_perceptron))

acc_perceptron_cv = round(cross_val_score(perceptron, X_train, Y_train, cv=10, scorin
print("Perceptron在训练集上的cv准确率是：{}%".format(acc_perceptron_cv))
```

```
Perceptron在训练集上的准确率是：73.51%
Perceptron在训练集上的cv准确率是：69.02%
```

5.6 LinearSVC

In [57]:

```
linear_svc = LinearSVC(max_iter = 10000)
linear_svc.fit(X_train, Y_train)
Y_pred_linear_svc = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
print("linear_svc在训练集上的准确率是：{}%".format(acc_linear_svc))

acc_linear_svc_cv = round(cross_val_score(linear_svc, X_train, Y_train, cv=10, scorin
print("linear_svc在训练集上的cv准确率是：{}%".format(acc_linear_svc_cv))
```

```
linear_svc在训练集上的准确率是：83.16%
linear_svc在训练集上的cv准确率是：83.05%
```

5.7 Stochastic Gradient Descent

In [58]:

```
sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred_SGD = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
print("SGD在训练集上的准确率是：{}%".format(acc_sgd))

acc_sgd_cv = round(cross_val_score(sgd, X_train, Y_train, cv=10, scoring='accuracy').
print("SGD在训练集上的cv准确率是：{}%".format(acc_sgd_cv))
```

```
SGD在训练集上的准确率是：82.04%
SGD在训练集上的cv准确率是：79.69%
```

5.8 Decision Tree

In [59]:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred_decision_tree = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
print("Decision Tree在训练集上的准确率是：{}%".format(acc_decision_tree))

acc_decision_tree_cv = round(cross_val_score(decision_tree, X_train, Y_train, cv=10, 
print("Decision Tree在训练集上的cv准确率是：{}%".format(acc_decision_tree_cv))
```

```
Decision Tree在训练集上的准确率是：88.55%
Decision Tree在训练集上的cv准确率是：80.25%
```

5.9 Random Forest

In [60]:

```
random_forest = RandomForestClassifier()
random_forest.fit(X_train, Y_train)
Y_pred_random_forest = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
print("RF在训练集上的准确率是：{}%".format(acc_random_forest))
```

```
acc_random_forest_cv = round(cross_val_score(random_forest, X_train, Y_train, cv=10, s
print("RF在训练集上的cv准确率是：{}%".format(acc_random_forest_cv))
```

```
RF在训练集上的准确率是：88.55%
RF在训练集上的cv准确率是：80.7%
```

5.10 对比各个模型的准确率

In [61]:

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_LR,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree],
    'CV-Score': [acc_svc_cv, acc_knn_cv, acc_LR_cv,
              acc_random_forest_cv, acc_gaussian_cv, acc_perceptron_cv,
              acc_sgd_cv, acc_linear_svc_cv, acc_decision_tree_cv]
})
models.sort_values(by='Score', ascending=False)
```

Out[61]:

	Model	Score	CV-Score
3	Random Forest	88.55	80.70
8	Decision Tree	88.55	80.25
1	KNN	84.74	82.72
0	Support Vector Machines	83.50	83.28
2	Logistic Regression	83.28	82.94
7	Linear SVC	83.16	83.05
6	Stochastic Gradient Decent	82.04	79.69
4	Naive Bayes	79.91	79.02
5	Perceptron	73.51	69.02

对比各模型在训练集上的准确率和交叉验证的准确率可以发现，随机森林和决策树的拟合能力强于其他分类算法，但交叉验证的平均准确率反而降低了

这说明随机森林的拟合能力太强，很有可能产生了过拟合

5.11 超参数调节

在kaggle竞赛里，集成学习方法例如随机森林，XGBoost等通常都能取得很好的表现，也是众多选手们首选的模型。

为了节省训练模型的时间，也为了精简教学，突出核心思想，我们将随机森林作为预测用的最终模型，使用交叉验证法对模型进行评估，使用网格搜索法进行超参数调节。

其他更高级而复杂的模型融合方法、参数调节方法可以进一步提高模型的表现，留待同学们自行学习

In [62]:

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
RFC = RandomForestClassifier(n_jobs=-1)
```

```
## Search grid for optimal parameters
rf_param_grid = {
    'max_depth' : [4, 5, 6],
    "min_samples_split": [2, 3, 5],
    "min_samples_leaf": [2, 3, 5],
    "n_estimators" :[50, 70, 90],
    "criterion": ["gini"]
}

# Cross validate model with Kfold stratified cross val
kfold = StratifiedKFold(n_splits=10)

gsRFC = GridSearchCV(RFC, param_grid=rf_param_grid, cv=kfold, scoring="accuracy", ver

gsRFC.fit(X_train, Y_train)

RFC_best = gsRFC.best_estimator_

# Best score
print("10折CV准确率：{}".format(round(gsRFC.best_score_, 2)))
print("训练集上的准确率：{}".format(round(RFC_best.score(X_train, Y_train), 2)))
```

```
Fitting 10 folds for each of 81 candidates, totalling 810 fits
10折CV准确率：0.84
训练集上的准确率：0.84
```

In [63]:

```
feature_importance = pd.DataFrame({
    'feature': X_train.columns,
    'feature importances': RFC_best.feature_importances_
})
feature_importance.sort_values(by='feature importances', ascending=False)
```

Out[63]:

	feature	feature importances
1	Sex	0.229128
8	Mr	0.207978
0	Pclass	0.144741
9	Mrs	0.103605
4	FamilySize	0.077083
7	Miss	0.075736
3	Fare	0.053809
2	Age	0.025095
11	C	0.019227
6	Master	0.019196
13	S	0.015611
5	IsAlone	0.014889
10	Rare	0.008743
12	Q	0.005160

In [64]:

```
Y_pred_RFbest = RFC_best.predict(X_test)
```

练习4:

使用XGBoost构建分类器，使用交叉验证法对模型进行评估，使用网格搜索法进行超参数调节，看是否能提高模型的预测能力

6. 提交

In [65]:

```
submission = pd.DataFrame({
        "PassengerId": test_df["PassengerId"],
        "Survived": Y_pred_RFbest
    })

submission.to_csv('submission.csv', index=False)

submission_LR = pd.DataFrame({
        "PassengerId": test_df["PassengerId"],
        "Survived": Y_pred_LR
    })
submission_LR.to_csv('submission_LR.csv', index=False)

submission_random_forest = pd.DataFrame({
        "PassengerId": test_df["PassengerId"],
        "Survived": Y_pred_random_forest
    })
submission_random_forest.to_csv('submission_random_forest.csv', index=False)
```