Python 标准 Titanic 数据集存活率预测数据项目操作手册

一、 数据准备

Titanic 生存模型预测，其中包含了两组数据 :train.csv 和 test.csv，分别为训练集合和测试集合。

新建工程，命名为 TitanicPredicate（用 Spyder 或 PyCharm 均可）

1、 加载并浏览数据，以及查看数据的基本信息 （TitanicDataDescribe1.py）

```
import re #加载正则表达式库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

#matplotlib inline


#用程序观察前几行的源数据：

train_data = pd.read_csv('data/train.csv')   #训练数据集

test_data = pd.read_csv('data/test.csv') #  验证数据集

sns.set_style('whitegrid')

train_data.head()

#train data
```

train_data - DataFrame

Index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	nan	S
1	2	1	1	Cumings, Mrs. John Bradley …	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	nan	S
3	4	1	1	Futrelle, Mrs. Jacques Heath…	female	35	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	nan	S
5	6	0	3	Moran, Mr. James	male	nan	0	0	330877	8.4583	nan	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta…	male	2	3	1	349909	21.075	nan	S
8	9	1	3	Johnson, Mrs. Oscar W (Elis…	female	27	0	2	347742	11.1333	nan	S
9	10	1	2	Nasser, Mrs. Nicholas (Ade…	female	14	1	0	237736	30.0708	nan	C
10	11	1	3	Sandstrom, Miss. Marguer…	female	4	1	1	PP 9549	16.7	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
12	13	0	3	Saundercock,	male	20	0	0	A/5. 2151	8.05	nan	S

#test data

test_data - DataFrame

Index	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	nan	Q
1	893	3	Wilkes, Mrs. James (Ellen …	female	47	1	0	363272	7	nan	S
2	894	2	Myles, Mr. Thomas Francis	male	62	0	0	240276	9.6875	nan	Q
3	895	3	Wirz, Mr. Albert	male	27	0	0	315154	8.6625	nan	S
4	896	3	Hirvonen, Mrs. Alexander (He…	female	22	1	1	3101298	12.2875	nan	S
5	897	3	Svensson, Mr. Johan Cervin	male	14	0	0	7538	9.225	nan	S
6	898	3	Connolly, Miss. Kate	female	30	0	0	330972	7.6292	nan	Q
7	899	2	Caldwell, Mr. Albert Francis	male	26	1	1	248738	29	nan	S
8	900	3	Abrahim, Mrs. Joseph (Sophi…	female	18	0	0	2657	7.2292	nan	C
9	901	3	Davies, Mr. John Samuel	male	21	2	0	A/4 48871	24.15	nan	S
10	902	3	Ilieff, Mr. Ylio	male	nan	0	0	349220	7.8958	nan	S
11	903	1	Jones, Mr. Charles Cress…	male	46	0	0	694	26	nan	S
12	904	1	Snyder, Mrs. John Pillsbur…	female	23	1	0	21228	82.2667	B45	S

数据信息总览：

train_data.info()

print("-" * 40)

test_data.info()

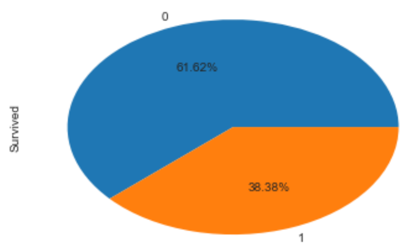
#

```
In [3]: runfile('C:/Users/zhongyunqin/TitanicKaggle/
TitanicDataDescribe.py', wdir='C:/Users/zhongyunqin/Titan:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
----------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
```

#从上面我们可以看出，Age、Cabin、Embarked、Fare 几个特征存在缺失值。

绘制存活的比例

train_data['Survived'].value_counts().plot.pie(autopct = '%1.2f%%')



2、 对 Titanic 数据集的缺失值进行处理 （TitanicProcessMiss2.py）

对数据进行分析的时候要注意其中是否有缺失值。

一些机器学习算法能够处理缺失值，比如神经网络，一些则不能。对于缺失值，一般有以下几种处理方法：

（1）如果数据集很多，但有很少的缺失值，可以删掉带缺失值的行；

（2）如果该属性相对学习来说不是很重要，可以对缺失值赋均值或者众数

#比如在哪儿上船 Embarked 这一属性（共有三个上船地点），缺失俩值，可以用众数赋值

train_data.Embarked[train_data.Embarked.isnull()] = train_data.Embarked.dropna().mode().values

#（3）对于标称属性，可以赋一个代表缺失的值，比如'U0'。因为缺失本身也可能代表着一些隐含信息。比如船舱号 Cabin 这一属性，缺失可能代表并没有船舱。

#replace missing value with U0

train_data['Cabin'] = train_data.Cabin.fillna('U0') # train_data.Cabin[train_data.Cabin.isnull()]='U0'

使用回归随机森林等模型来预测缺失属性的值。因为 Age 在该数据集里是一个相当重要的特征（先对 Age 进行分析即可得知），所以保证一定的缺失值填充准确率是非常重要的，对结果也会产生较大影响。一般情况下，会使用数据完整的条目作为模型的训练集，以此来预测缺失值。对于当前的这个数据，可以使用随机森林来预测也可以使用线性回归预测。这里使用随机森林预测模型，选取数据集中的数值属性作为特征（因为 sklearn 的模型只能处理数值属性，所以这里先仅选取数值特征，但在实际的应用中需要将非数值特征转换为数值特征）

```
from sklearn.ensemble import RandomForestRegressor


#choose training data to predict age

age_df = train_data[['Age','Survived','Fare', 'Parch', 'SibSp', 'Pclass']]

age_df_notnull = age_df.loc[(train_data['Age'].notnull())]

age_df_isnull = age_df.loc[(train_data['Age'].isnull())]

X = age_df_notnull.values[:,1:]

Y = age_df_notnull.values[:,0]

# use RandomForestRegression to train data
```

RFR = RandomForestRegressor(n_estimators=1000, n_jobs=-1)

RFR.fit(X,Y)

predictAges = RFR.predict(age_df_isnull.values[:,1:])

train_data.loc[train_data['Age'].isnull(), ['Age']]= predictAges

输出缺失数据处理后的 DataFrame，可以看到年龄已经被填充：

train_data.info()

```
In [2]: runfile('C:/Users/zhongyunqin/
TitanicKaggle/TitanicProcessMiss.py', wdir='C:/
Users/zhongyunqin/TitanicKaggle')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          891 non-null object
Embarked       891 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	U0	S
2	1	1	Cumings, Mrs. John Bradley …	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	U0	S
4	1	1	Futrelle, Mrs. Jacques Heath…	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	U0	S
6	0	3	Moran, Mr. James	male	23.7013	0	0	330877	8.4583	U0	Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta…	male	2	3	1	349909	21.075	U0	S
9	1	3	Johnson, Mrs. Oscar W (Elis…	female	27	0	2	347742	11.1333	U0	S
10	1	2	Nasser, Mrs. Nicholas (Ade…	female	14	1	0	237736	30.0708	U0	C
11	1	3	Sandstrom, Miss. Marguer…	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
13	0	3	Saundercock, Mr. William H…	male	20	0	0	A/5. 2151	8.05	U0	S

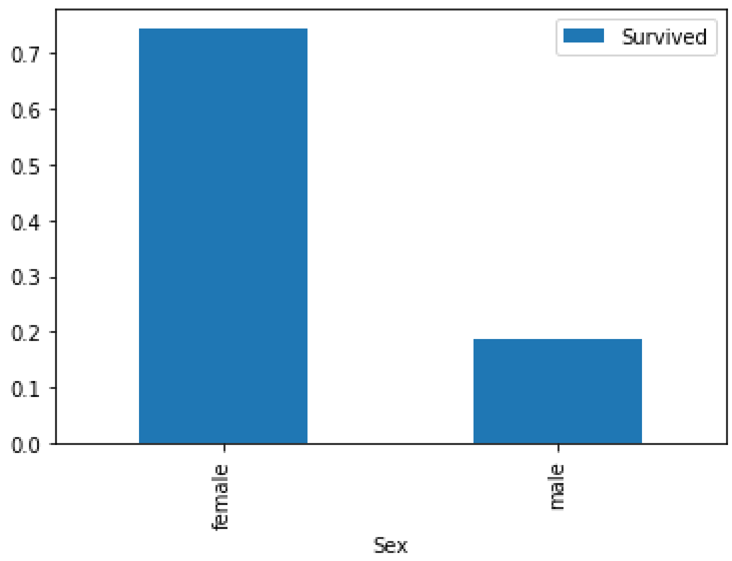
3、 分析数据属性（变量、特征）之间关系 （TitanicFeatureRelation3.py）

性别与是否生存的关系 Sex

train_data.groupby(['Sex','Survived'])['Survived'].count()

train_data[['Sex','Survived']].groupby(['Sex']).mean().plot.bar()

以下图为不同性别的生存率，可见在泰坦尼克号事故中，还是体现了 Lady First。

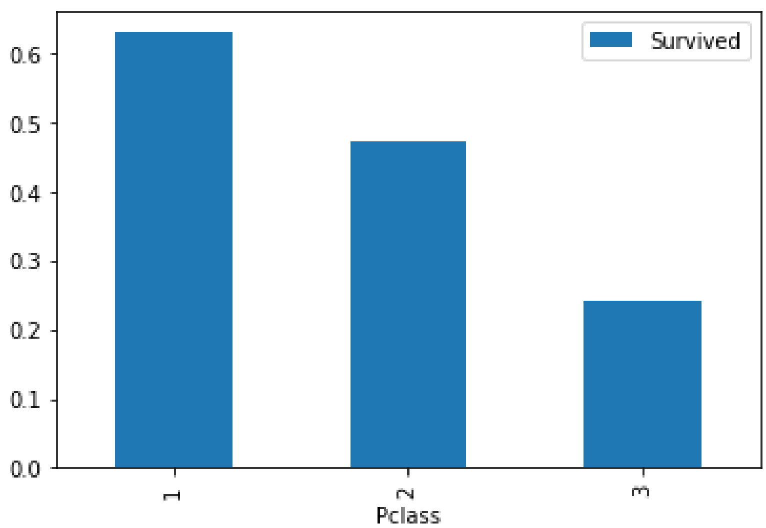


船舱等级和生存与否的关系 Pclass

train_data.groupby(['Pclass','Survived'])['Pclass'].count()

train_data[['Pclass','Survived']].groupby(['Pclass']).mean().plot.bar()

船舱等级和生存的关系如下输出结果

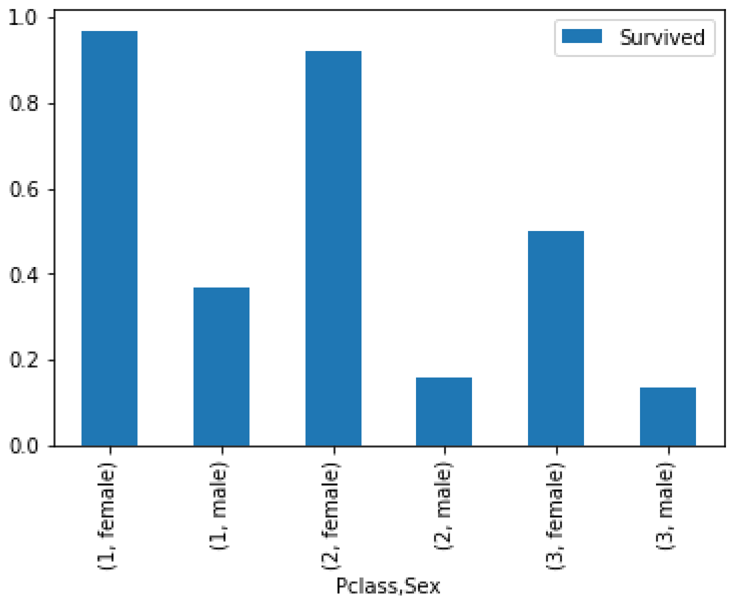


#不同等级船舱的男女生存率

train_data.groupby(['Sex', 'Pclass', 'Survived'])['Survived'].count()

train_data[['Sex','Pclass','Survived']].groupby(['Pclass','Sex']).mean().plot.bar()

#从图中可以看出，总体上泰坦尼克号逃生是妇女优先，但是对于不同等级的船舱还是有一定的区别。



#年龄与存活与否的关系 Age

#分别分析不同等级船舱和不同性别下的年龄分布和生存的关系：

fig, ax = plt.subplots(1, 2, figsize = (18, 8))

sns.violinplot("Pclass", "Age", hue="Survived", data=train_data, split=True, ax=ax[0])

ax[0].set_title('Pclass and Age vs Survived')

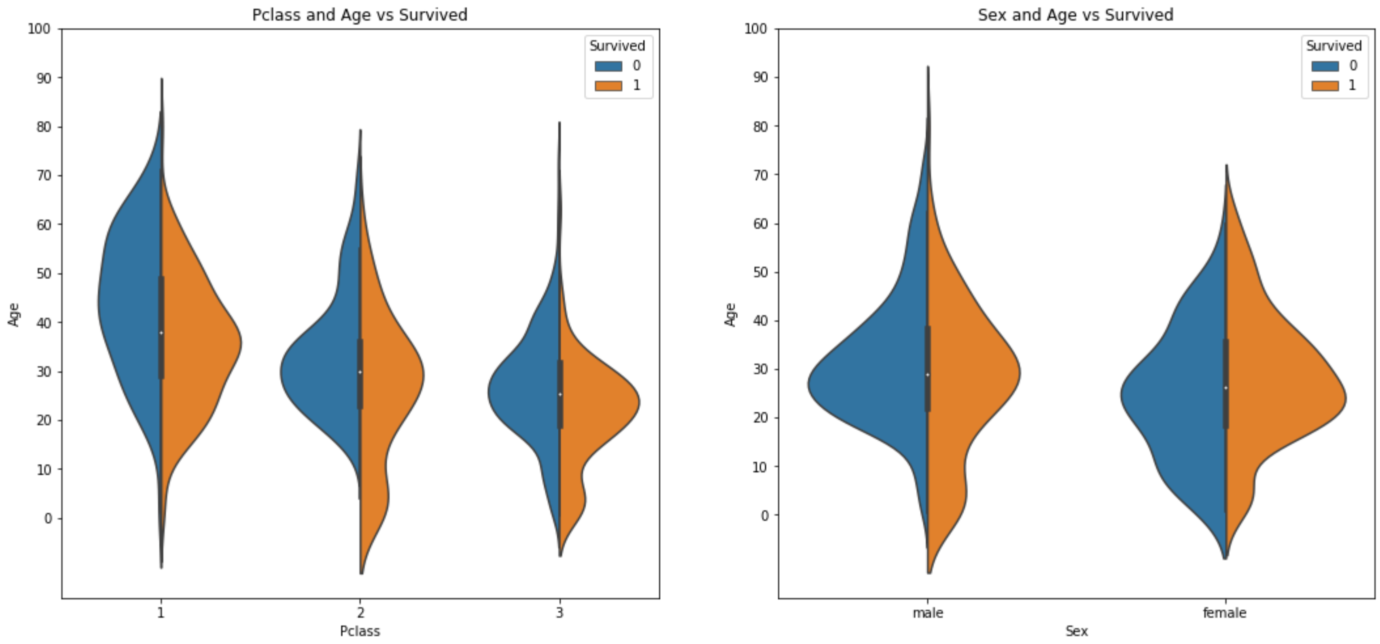
ax[0].set_yticks(range(0, 110, 10))

sns.violinplot("Sex", "Age", hue="Survived", data=train_data, split=True, ax=ax[1])

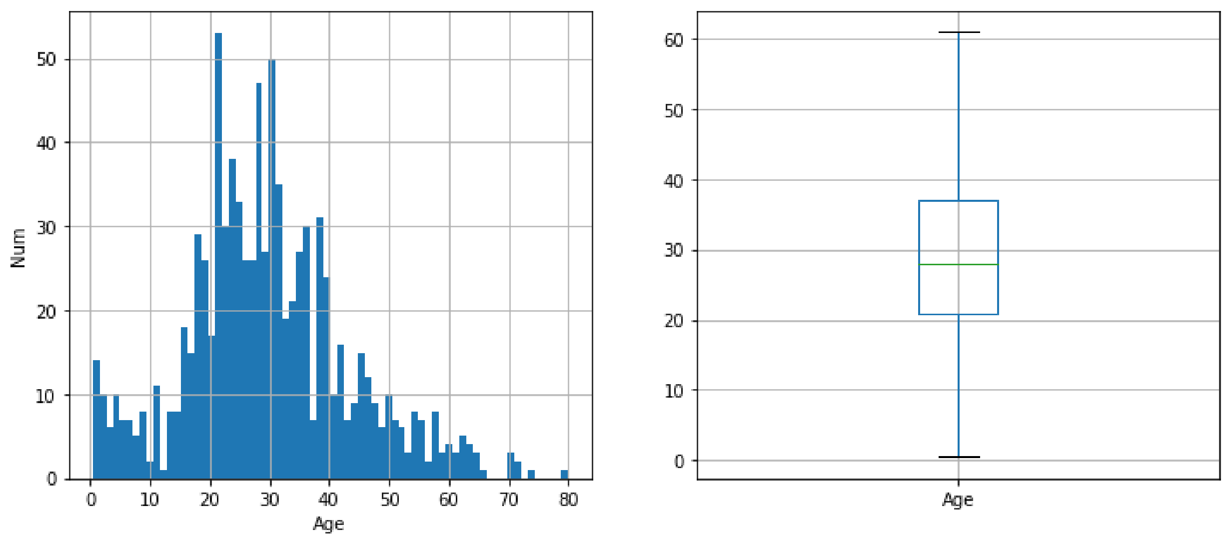
ax[1].set_title('Sex and Age vs Survived')

ax[1].set_yticks(range(0, 110, 10))

plt.show()




```
#分析总体的年龄分布

plt.figure(figsize=(12,5))

plt.subplot(121)

train_data['Age'].hist(bins=70)

plt.xlabel('Age')

plt.ylabel('Num')

plt.subplot(122)

train_data.boxplot(column='Age', showfliers=False)

plt.show()
```



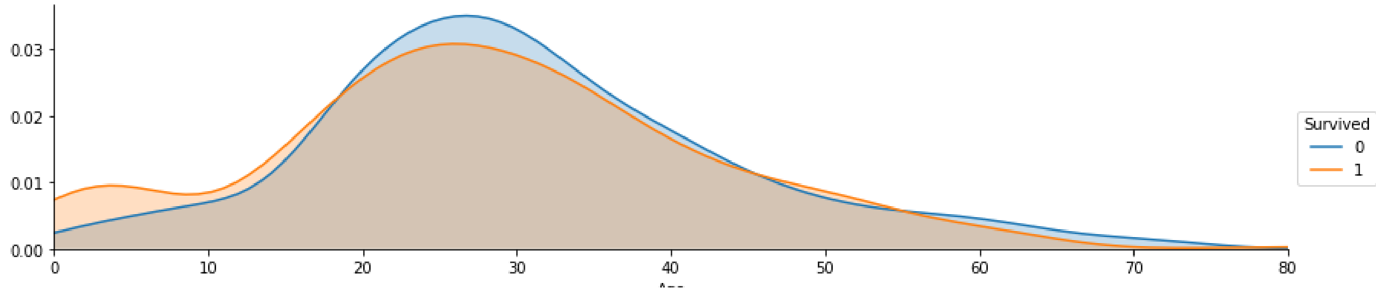
#不同年龄下的生存和非生存的分布情况：

facet = sns.FacetGrid(train_data, hue="Survived",aspect=4)

facet.map(sns.kdeplot,'Age',shade= True)

facet.set(xlim=(0, train_data['Age'].max()))

facet.add_legend()



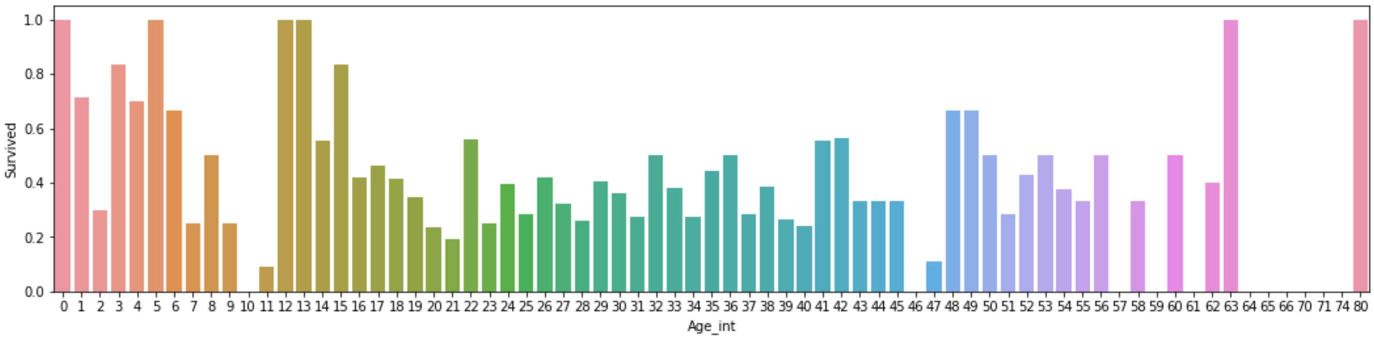
#不同年龄下的平均生存率

fig, axis1 = plt.subplots(1,1,figsize=(18,4))

train_data["Age_int"] = train_data["Age"].astype(int)

average_age = train_data[["Age_int", "Survived"]].groupby(['Age_int'],as_index=False).mean()

sns.barplot(x='Age_int', y='Survived', data=average_age)



输出年龄特征的统计信息

train_data['Age'].describe()

#输出

count 891.000000

mean 29.668231

std 13.739002

min 0.420000

25% 21.000000

50% 28.000000

75% 37.000000

max 80.000000

Name: Age, dtype: float64

#样本有 891，平均年龄约为 30 岁，标准差 13.5 岁，最小年龄为 0.42，最大年龄 80.

#按照年龄，将乘客划分为儿童、少年、成年和老年，分析四个群体的生还情况

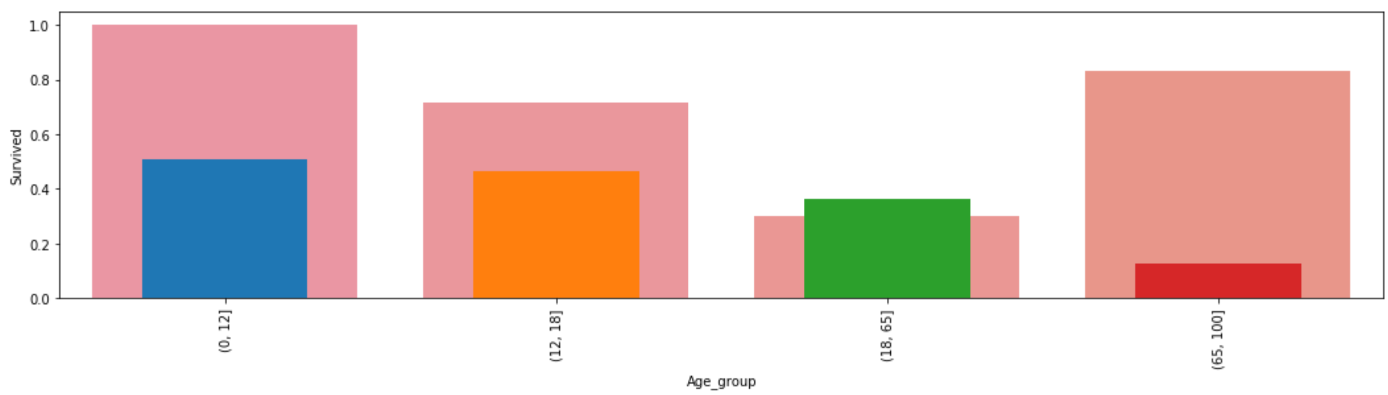
```
bins = [0, 12, 18, 65, 100]

train_data['Age_group'] = pd.cut(train_data['Age'], bins)

by_age = train_data.groupby('Age_group')['Survived'].mean()

by_age
```

by_age - Series

Index	Survived
(0, 12]	0.506173
(12, 18]	0.466667
(18, 65]	0.364512
(65, 100]	0.125

#按照年龄，将乘客划分为儿童、少年、成年和老年，分析四个群体的生还情况分布图

```
by_age.plot(kind = 'bar')
```



乘客的姓名称呼与存活与否的关系 Name

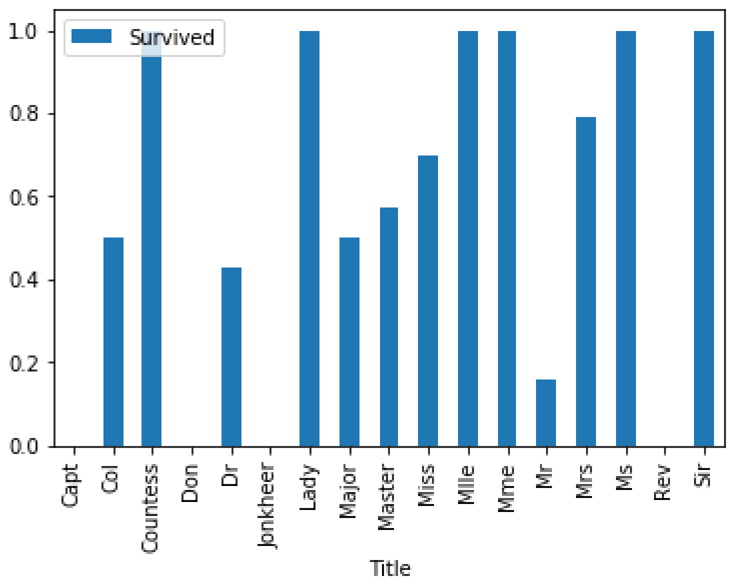
#通过观察名字数据，我们可以看出其中包括对乘客的称呼，如：Mr、Miss、Mrs 等，称呼信息包含了乘客的年龄、性别，同时也包含了如社会地位等的称呼，如：Dr,、Lady、Major、Master 等的称呼。

train_data['Title'] = train_data['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train_data['Title'], train_data['Sex'])

#观察不同称呼与生存率的关系

train_data[['Title','Survived']].groupby(['Title']).mean().plot.bar()



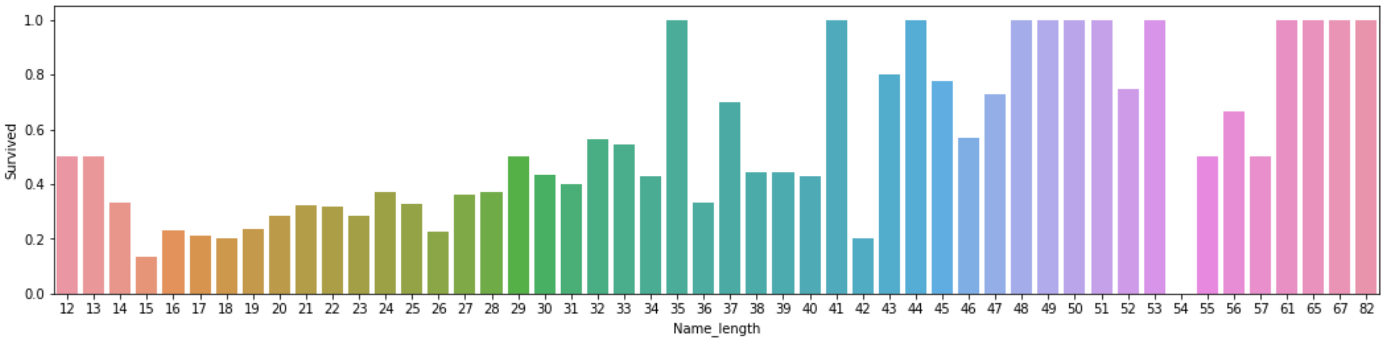
#同时，对于名字，我们还可以观察名字长度和生存率之间存在关系的可能

fig, axis1 = plt.subplots(1,1,figsize=(18,4))

train_data['Name_length'] = train_data['Name'].apply(len)

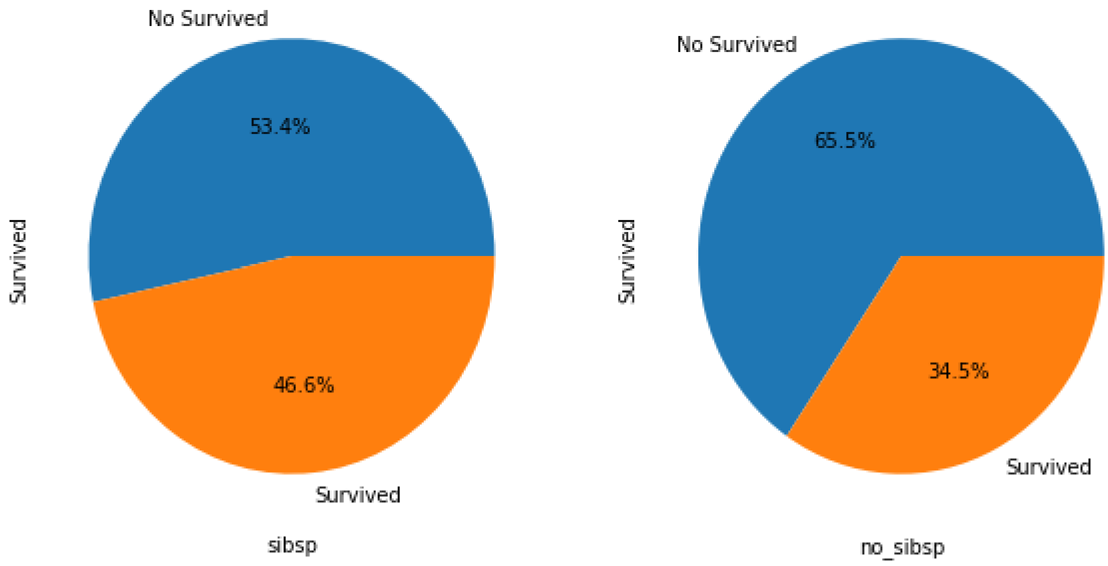
name_length = train_data[['Name_length','Survived']].groupby(['Name_length'],as_index=False).mean()

sns.barplot(x='Name_length', y='Survived', data=name_length)



#从上面的图片可以看出，名字长度和生存与否确实也存在一定的相关性

```
# 有无兄弟姐妹和存活与否的关系 SibSp

# 将数据分为有兄弟姐妹的和没有兄弟姐妹的两组

sibsp_df = train_data[train_data['SibSp'] != 0]

no_sibsp_df = train_data[train_data['SibSp'] == 0]


plt.figure(figsize=(10,5))

plt.subplot(121)

sibsp_df['Survived'].value_counts().plot.pie(labels=['No Survived', 'Survived'], autopct = '%1.1f%%')

plt.xlabel('sibsp')


plt.subplot(122)

no_sibsp_df['Survived'].value_counts().plot.pie(labels=['No Survived', 'Survived'], autopct =
'%1.1f%%')

plt.xlabel('no_sibsp')

plt.show()

# 结果如图显示，有兄弟姐妹的获救率高些
```



#有无父母子女和存活与否的关系 Parch，和有无兄弟姐妹一样，同样分析可以得到：

parch_df = train_data[train_data['Parch'] != 0]

no_parch_df = train_data[train_data['Parch'] == 0]

plt.figure(figsize=(10,5))

plt.subplot(121)

parch_df['Survived'].value_counts().plot.pie(labels=['No Survived', 'Survived'], autopct = '%1.1f%%')

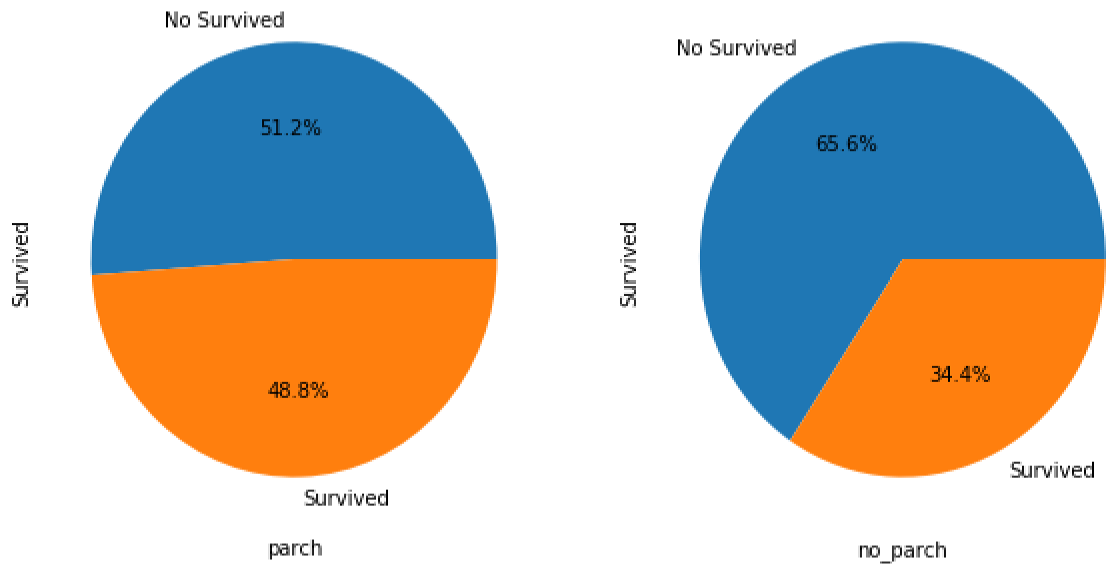
plt.xlabel('parch')

plt.subplot(122)

no_parch_df['Survived'].value_counts().plot.pie(labels=['No Survived', 'Survived'], autopct = '%1.1f%%')

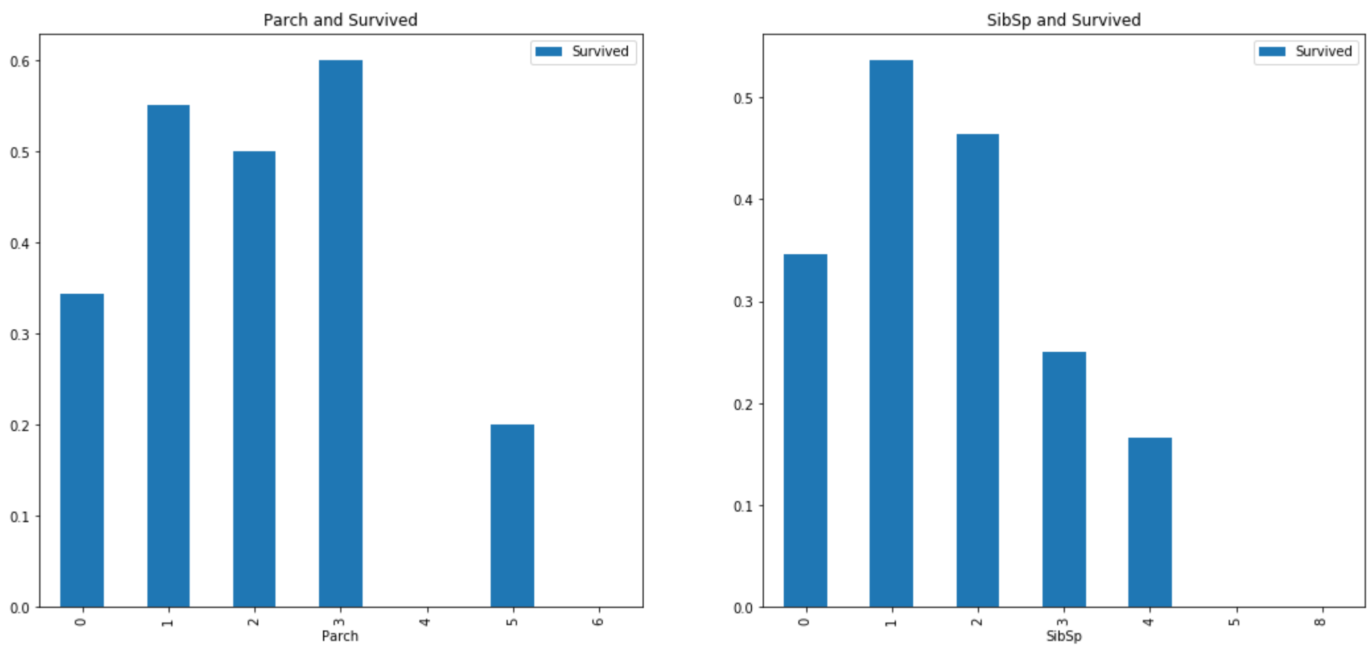
plt.xlabel('no_parch')

plt.show()



#亲友的人数和存活与否的关系 SibSp & Parch

```
fig,ax=plt.subplots(1,2,figsize=(18,8))

train_data[['Parch','Survived']].groupby(['Parch']).mean().plot.bar(ax=ax[0])

ax[0].set_title('Parch and Survived')

train_data[['SibSp','Survived']].groupby(['SibSp']).mean().plot.bar(ax=ax[1])

ax[1].set_title('SibSp and Survived')
```

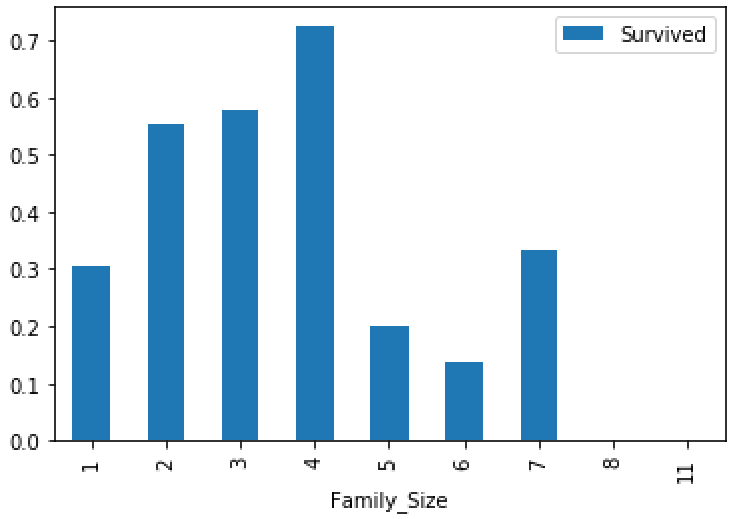


train_data['Family_Size'] = train_data['Parch'] + train_data['SibSp'] + 1

train_data[['Family_Size','Survived']].groupby(['Family_Size']).mean().plot.bar()

#从图表中可以看出，若独自一人，那么其存活率比较低；

#但是如果亲友太多的话，存活率也会很低。



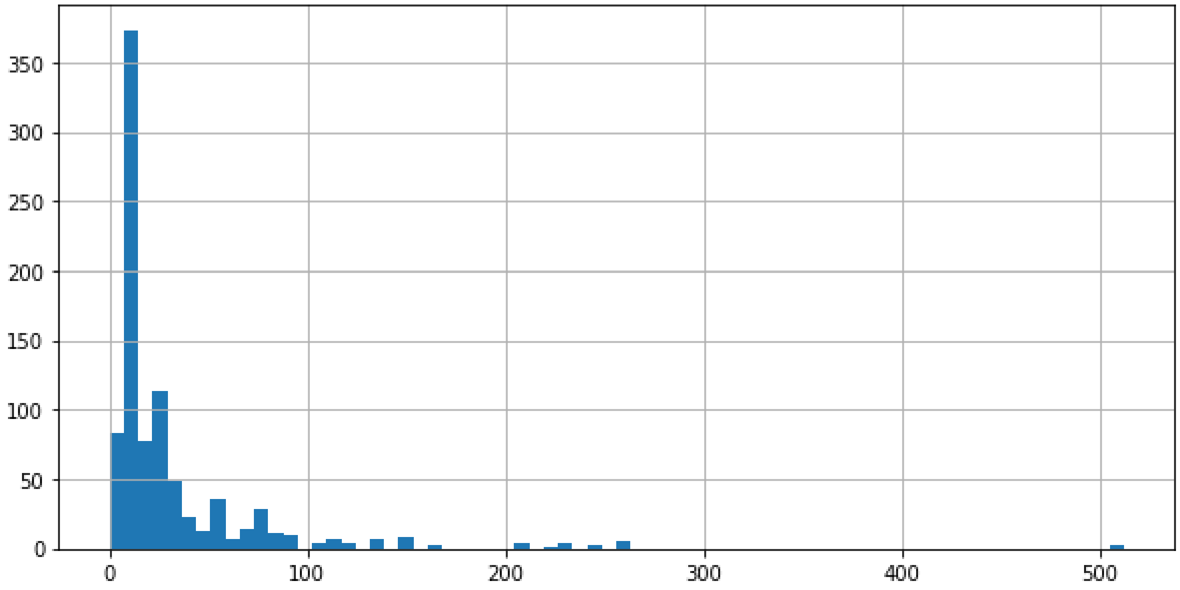
票价分布和存活与否的关系 Fare，首先绘制票价的分布情况：

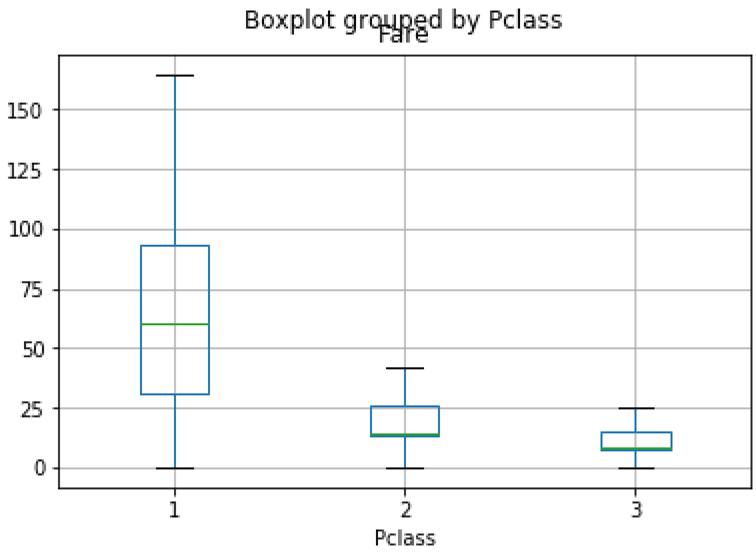
plt.figure(figsize=(10,5))

train_data['Fare'].hist(bins = 70)

train_data.boxplot(column='Fare', by='Pclass', showfliers=False)

plt.show()





train_data['Fare'].describe()   #查看 Fare 的数据分布

#绘制生存与否与票价均值和方差的关系：

fare_not_survived = train_data['Fare'][train_data['Survived'] == 0]

fare_survived = train_data['Fare'][train_data['Survived'] == 1]

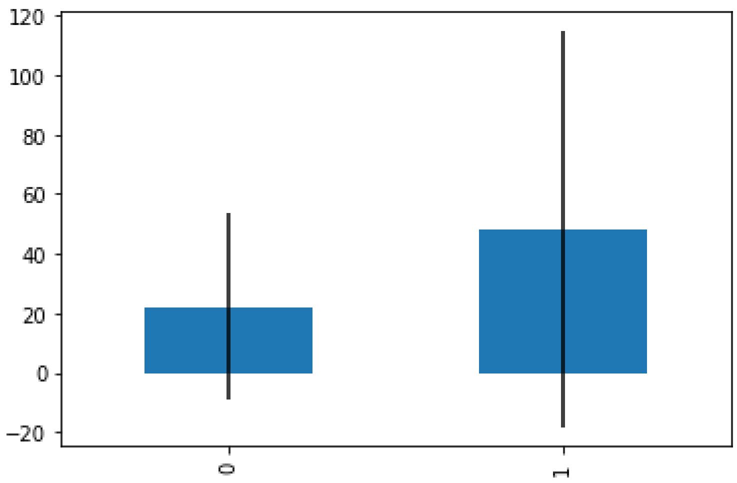
average_fare = pd.DataFrame([fare_not_survived.mean(), fare_survived.mean()])

std_fare = pd.DataFrame([fare_not_survived.std(), fare_survived.std()])

average_fare.plot(yerr=std_fare, kind='bar', legend=False)

plt.show()

#由图可知，票价与是否生还有一定的相关性，生还者的平均票价要大于未生还者的平均票价。



#船舱类型和存活与否的关系 Cabin

#由于船舱的缺失值确实太多，有效值仅仅有 204 个，很难分析出不同的船舱和存活的关系，所以在做特征工程的时候，可以直接将该组特征丢弃。

#当然也可以对其进行分析，对于缺失的数据都分为一类。

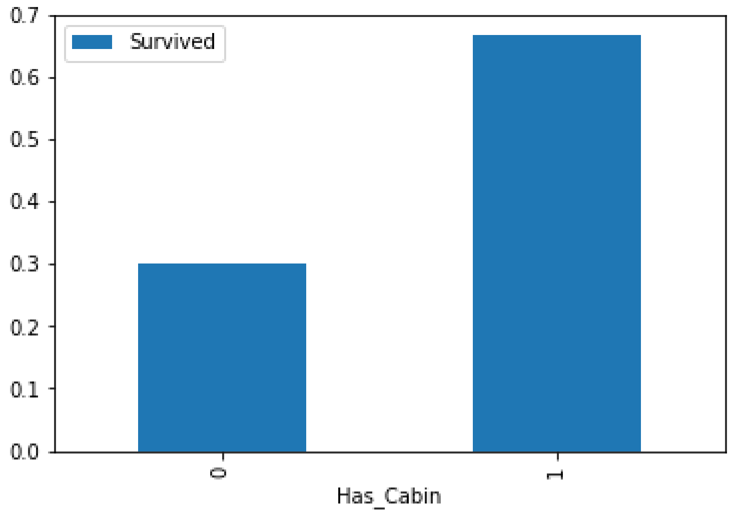
#简单地将数据分为是否有 Cabin 记录作为特征，与生存与否进行分析：

把 Cabin 缺失值替换为 "U0"

train_data.loc[train_data.Cabin.isnull(), 'Cabin'] = 'U0'

train_data['Has_Cabin'] = train_data['Cabin'].apply(lambda x: 0 if x == 'U0' else 1)

train_data[['Has_Cabin','Survived']].groupby(['Has_Cabin']).mean().plot.bar()



#对不同类型的船舱进行分析：

create feature for the alphabetical part of the cabin number

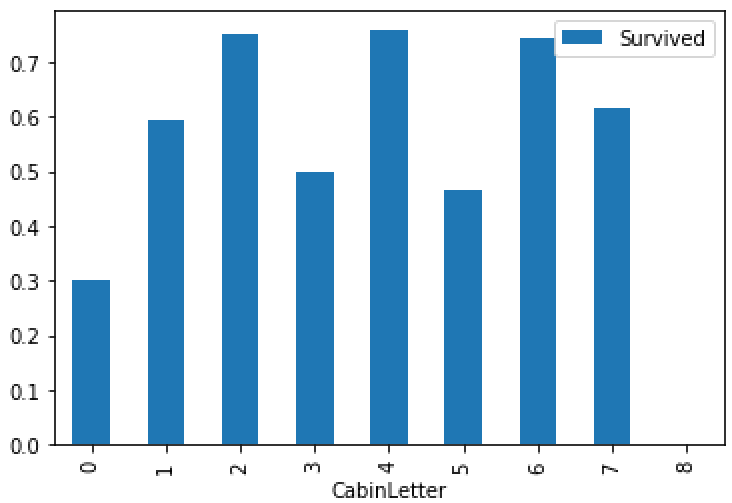
train_data['CabinLetter'] = train_data['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).group())

convert the distinct cabin letters with incremental integer values

train_data['CabinLetter'] = pd.factorize(train_data['CabinLetter'])[0]

train_data[['CabinLetter','Survived']].groupby(['CabinLetter']).mean().plot.bar()

#可见，不同的船舱生存率也有不同，但是差别不大。所以在处理中，可以直接将删除该特征。

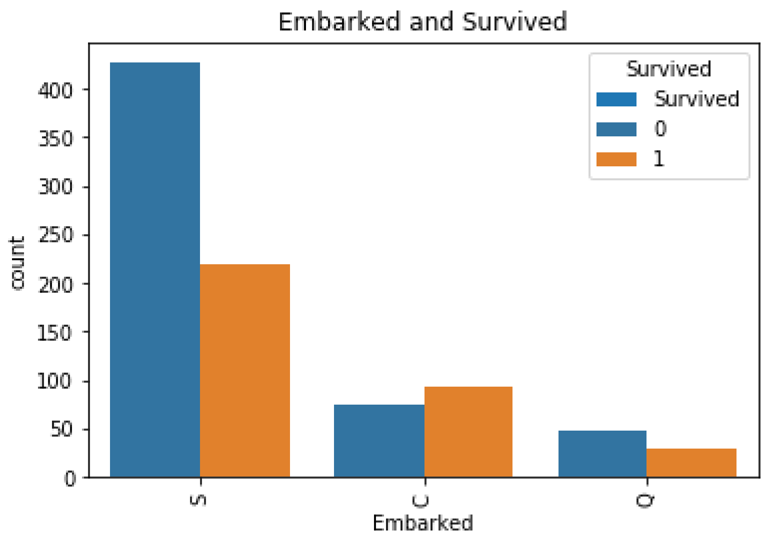


#港口和存活与否的关系 Embarked

#泰坦尼克号从英国的南安普顿港出发，途径法国瑟堡和爱尔兰昆士敦，那么在昆士敦之前上船的人，有可能在瑟堡或昆士敦下船，这些人将不会遇到海难

sns.countplot('Embarked', hue='Survived', data=train_data)

plt.title('Embarked and Survived')

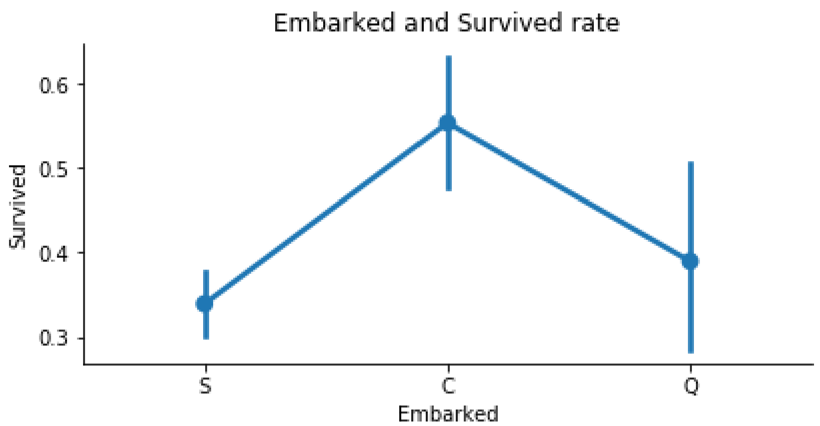


#分析得出在不同的港口上船，生还率不同，C 最高，Q 次之，S 最低

sns.factorplot('Embarked', 'Survived', data=train_data, size=3, aspect=2)

plt.title('Embarked and Survived rate')

plt.show()



#以上为所给出的数据特征与生还与否的分析。据 Kaggle 数据集表明，泰坦尼克号上共有 2224 名乘客。本训练数据只给出了 891 名乘客的信息，如果该数据集是从总共的 2224 人中随机选出

的，根据中心极限定理，该样本的数据也足够大，那么我们的分析结果就具有代表性

#其他对于数据集中没有给出的特征信息，我们还可以联想其他可能会对模型产生影响的特征因素。如：乘客的国籍、乘客的身高、乘客的体重、乘客是否会游泳、乘客职业等等。

4、 变量转换

变量转换的目的是将数据转换为适用于模型使用的数据，不同模型接受不同类型的数据， 下面对数据的转换进行介绍，以在特征工程可使用。

所有的数据可以分为两类：

1、定性(Qualitative)转换：Dummy Variables

就是类别变量或者二元变量，当 qualitative variable 是一些频繁出现的几个独立变量时，Dummy Variables 比较适合使用。我们以 Embarked 为例，Embarked 只包含三个值'S','C','Q'，我们可以使用下面的代码将其转换为 dummies:

```
embark_dummies  = pd.get_dummies(train_data['Embarked'])
train_data = train_data.join(embark_dummies)
train_data.drop(['Embarked'], axis=1,inplace=True)


embark_dummies = train_data[['S', 'C', 'Q']]
embark_dummies.head()
   S  C  Q
0  1  0  0
1  0  1  0
2  1  0  0
3  1  0  0
```

2、定性转换：Factorizing

dummy 不好处理 Cabin（船舱号）这种标称属性，因为他出现的变量比较多。所以 Pandas 有一个方法叫做 factorize()，它可以创建一些数字，来表示类别变量，对每一个类别映射一个 ID，这种映射最后只生成一个特征，不像 dummy 那样生成多个特征。

Replace missing values with "U0"

train_data['Cabin'][train_data.Cabin.isnull()] = 'U0'

create feature for the alphabetical part of the cabin number

train_data['CabinLetter'] = train_data['Cabin'].map(lambda x : re.compile("([a-zA-Z]+)").search(x).group())

convert the distinct cabin letters with incremental integer values

train_data['CabinLetter'] = pd.factorize(train_data['CabinLetter'])[0]

train_data['CabinLetter'].head()

#3、定量(Quantitative)转换：Scaling

Scaling 可以将一个很大范围的数值映射到一个很小的范围(通常是-1 - 1，或则是 0 - 1)，很多情况下我们需要将数值做 Scaling 使其范围大小一样，否则大范围数值特征将会由更高的权重。比如：Age 的范围可能只是 0-100，而 income 的范围可能是 0-10000000，在某些对数组大小敏感的模型中会影响其结果。

#下面对 Age 进行 Scaling：

from sklearn import preprocessing

assert np.size(train_data['Age']) == 891

StandardScaler will subtract the mean from each value then scale to the unit variance

scaler = preprocessing.StandardScaler()

train_data['Age_scaled'] = scaler.fit_transform(train_data['Age'].values.reshape(-1, 1))

train_data['Age_scaled'].head()

#　定量(Quantitative)转换：Binning 分箱

Binning 通过观察“邻居”(即周围的值)将连续数据离散化。存储的值被分布到一些“桶”或“箱“”中，就像直方图的 bin 将数据划分成几块一样。下面的代码对 Fare 进行 Binning。

```
# Divide all fares into quartiles
train_data['Fare_bin'] = pd.qcut(train_data['Fare'], 5)
train_data['Fare_bin'].head()

# 在将数据分箱处理后，要么将数据 factorize 化，要么 dummies 化。
# qcut() creates a new variable that identifies the quartile range, but we can't use the string
# so either factorize or create dummies from the result
# factorize
train_data['Fare_bin_id'] = pd.factorize(train_data['Fare_bin'])[0]
# dummies
fare_bin_dummies_df = pd.get_dummies(train_data['Fare_bin']).rename(columns=lambda x: 'Fare_' + str(x))
train_data = pd.concat([train_data, fare_bin_dummies_df], axis=1)
```

5、 特征工程

在进行特征工程的时候，不仅需要对训练数据进行处理，还需要同时将测试数据同训练数据一起处理，使得二者具有相同的数据类型和数据分布。

```
train_df_org = pd.read_csv('data/train.csv')
test_df_org = pd.read_csv('data/test.csv')
test_df_org['Survived'] = 0
```

combined_train_test = train_df_org.append(test_df_org)

PassengerId = test_df_org['PassengerId']

#对数据进行特征工程，也就是从各项参数中提取出对输出结果有或大或小的影响的特征，将这些特征作为训练模型的依据。 一般来说，我们会先从含有缺失值的特征开始。

#因为“Embarked”字段的缺失值不多，所以这里我们以众数来填充：

combined_train_test['Embarked'].fillna(combined_train_test['Embarked'].mode().iloc[0], inplace=True)

为了后面的特征分析，这里我们将 Embarked 特征进行 facrorizing

combined_train_test['Embarked'] = pd.factorize(combined_train_test['Embarked'])[0]

使用 pd.get_dummies 获取 one-hot 编码

emb_dummies_df = pd.get_dummies(combined_train_test['Embarked'], prefix=combined_train_test[['Embarked']].columns[0])

combined_train_test = pd.concat([combined_train_test, emb_dummies_df], axis=1)

#对 Sex 也进行 one-hot 编码，也就是 dummy 处理：

为了后面的特征分析，这里我们也将 Sex 特征进行 facrorizing

combined_train_test['Sex'] = pd.factorize(combined_train_test['Sex'])[0]

sex_dummies_df = pd.get_dummies(combined_train_test['Sex'], prefix=combined_train_test[['Sex']].columns[0])

combined_train_test = pd.concat([combined_train_test, sex_dummies_df], axis=1)

Name 字段，首先先从名字中提取各种称呼：

combined_train_test['Title'] = combined_train_test['Name'].map(lambda x: re.compile(", (.*?)\.").findall(x)[0])

#将各式称呼进行统一化处理：

```
title_Dict = {}

title_Dict.update(dict.fromkeys(['Capt', 'Col', 'Major', 'Dr', 'Rev'], 'Officer'))

title_Dict.update(dict.fromkeys(['Don', 'Sir', 'the Countess', 'Dona', 'Lady'], 'Royalty'))

title_Dict.update(dict.fromkeys(['Mme', 'Ms', 'Mrs'], 'Mrs'))

title_Dict.update(dict.fromkeys(['Mlle', 'Miss'], 'Miss'))

title_Dict.update(dict.fromkeys(['Mr'], 'Mr'))

title_Dict.update(dict.fromkeys(['Master','Jonkheer'], 'Master'))


combined_train_test['Title'] = combined_train_test['Title'].map(title_Dict)


#使用 dummy 对不同的称呼进行分列

#  为了后面的特征分析，这里我们也将  Title  特征进行 facrorizing

combined_train_test['Title'] = pd.factorize(combined_train_test['Title'])[0]
title_dummies_df = pd.get_dummies(combined_train_test['Title'], prefix=combined_train_test[['Title']].columns[0])
combined_train_test = pd.concat([combined_train_test, title_dummies_df], axis=1)

#增加名字长度的特征：

combined_train_test['Name_length'] = combined_train_test['Name'].apply(len)



# Fare 字段，由前面分析可以知道，Fare 项在测试数据中缺少一个值，所以需要对该值进行填充。

#我们按照一二三等舱各自的均价来填充：

#下面 transform 将函数 np.mean 应用到各个 group 中。

combined_train_test['Fare']                                                          =

combined_train_test[['Fare']].fillna(combined_train_test.groupby('Pclass').transform(np.mean))
```

#通过对 Ticket 数据的分析，我们可以看到部分票号数据有重复，同时结合亲属人数及名字的数据，和票价船舱等级对比，我们可以知道购买的票中有家庭票和团体票，所以我们需要将团体票的票价分配到每个人的头上。

```
combined_train_test['Group_Ticket'] =
combined_train_test['Fare'].groupby(by=combined_train_test['Ticket']).transform('count')
combined_train_test['Fare'] = combined_train_test['Fare'] / combined_train_test['Group_Ticket']
combined_train_test.drop(['Group_Ticket'], axis=1, inplace=True)
```

#使用 binning 给票价分等级

```
combined_train_test['Fare_bin'] = pd.qcut(combined_train_test['Fare'], 5)
```

#对于 5 个等级的票价我们也可以继续使用 dummy 为票价等级分列：

```
combined_train_test['Fare_bin_id'] = pd.factorize(combined_train_test['Fare_bin'])[0]

fare_bin_dummies_df = pd.get_dummies(combined_train_test['Fare_bin_id']).rename(columns=lambda x: 'Fare_' + str(x))
combined_train_test = pd.concat([combined_train_test, fare_bin_dummies_df], axis=1)
combined_train_test.drop(['Fare_bin'], axis=1, inplace=True)
```

#Pclass 字段，这一项其实已经可以不用继续处理了，我们只需要将其转换为 dummy 形式即可

#但是为了更好的分析问题，我们这里假设对于不同等级的船舱，各船舱内部的票价也说明了各等级舱的位置，那么也就很有可能与逃生的顺序有关系。所以这里分出每等舱里的高价和低价位。

```
from sklearn.preprocessing import LabelEncoder

# 建立 PClass Fare Category
def pclass_fare_category(df, pclass1_mean_fare, pclass2_mean_fare, pclass3_mean_fare):
```

```
    if df['Pclass'] == 1:

        if df['Fare'] <= pclass1_mean_fare:

            return 'Pclass1_Low'

        else:

            return 'Pclass1_High'

    elif df['Pclass'] == 2:

        if df['Fare'] <= pclass2_mean_fare:

            return 'Pclass2_Low'

        else:

            return 'Pclass2_High'

    elif df['Pclass'] == 3:

        if df['Fare'] <= pclass3_mean_fare:

            return 'Pclass3_Low'

        else:

            return 'Pclass3_High'


Pclass1_mean_fare =

combined_train_test['Fare'].groupby(by=combined_train_test['Pclass']).mean().get([1]).values[0]

Pclass2_mean_fare =

combined_train_test['Fare'].groupby(by=combined_train_test['Pclass']).mean().get([2]).values[0]

Pclass3_mean_fare =

combined_train_test['Fare'].groupby(by=combined_train_test['Pclass']).mean().get([3]).values[0]


# 建立 Pclass_Fare Category
combined_train_test['Pclass_Fare_Category'] = combined_train_test.apply(pclass_fare_category, args=(
```

```
    Pclass1_mean_fare, Pclass2_mean_fare, Pclass3_mean_fare), axis=1)
pclass_level = LabelEncoder()


# 给每一项添加标签
pclass_level.fit(np.array( ['Pclass1_Low', 'Pclass1_High', 'Pclass2_Low', 'Pclass2_High', 'Pclass3_Low', 'Pclass3_High']))


# 转换成数值
combined_train_test['Pclass_Fare_Category'] = pclass_level.transform(combined_train_test['Pclass_Fare_Category'])


# dummy 转换
pclass_dummies_df = pd.get_dummies(combined_train_test['Pclass_Fare_Category']).rename(columns=lambda x: 'Pclass_' + str(x))
combined_train_test = pd.concat([combined_train_test, pclass_dummies_df], axis=1)


#同时，我们将 Pclass 特征 factorize 化：
combined_train_test['Pclass'] = pd.factorize(combined_train_test['Pclass'])[0]




#Parch and SibSp 字段
#由前面的分析知道，亲友的数量没有或者太多会影响到 Survived。所以将二者合并为 FamliySize 这一组合项，同时也保留这两项。
def family_size_category(family_size):
    if family_size <= 1:
        return 'Single'
    elif family_size <= 4:
        return 'Small_Family'
    else:
```

```
        return 'Large_Family'

combined_train_test['Family_Size'] = combined_train_test['Parch'] + combined_train_test['SibSp'] + 1
combined_train_test['Family_Size_Category'] = combined_train_test['Family_Size'].map(family_size_category)

le_family = LabelEncoder()

le_family.fit(np.array(['Single', 'Small_Family', 'Large_Family']))
combined_train_test['Family_Size_Category'] = le_family.transform(combined_train_test['Family_Size_Category'])

family_size_dummies_df = pd.get_dummies(combined_train_test['Family_Size_Category'],

prefix=combined_train_test[['Family_Size_Category']].columns[0])

combined_train_test = pd.concat([combined_train_test, family_size_dummies_df], axis=1)
```

#Age 字段，因为 Age 项的缺失值较多，所以不能直接填充 age 的众数或者平均数。

#常见的有两种对年龄的填充方式：一种是根据 Title 中的称呼，如 Mr，Master、Miss 等称呼不同类别的人的平均年龄来填充；一种是综合几项如 Sex、Title、Pclass 等其他没有缺失值的项，使用机器学习算法来预测 Age。

#这里我们使用后者来处理。以 Age 为目标值，将 Age 完整的项作为训练集，将 Age 缺失的项作为测试集。

```
missing_age_df = pd.DataFrame(combined_train_test[
    ['Age', 'Embarked', 'Sex', 'Title', 'Name_length', 'Family_Size', 'Family_Size_Category','Fare', 'Fare_bin_id', 'Pclass']])

missing_age_train = missing_age_df[missing_age_df['Age'].notnull()]

missing_age_test = missing_age_df[missing_age_df['Age'].isnull()]

missing_age_test.head()
```

#输入结果：

	Age	Embarked	Sex	Title	Name_length	Family_Size	Family_Size_Category	Fare	Fare_bin_id	Pclass
5	NaN	2	0	0	16	1	1	8.4583	2	0
17	NaN	0	0	0	28	1	1	13.0000	3	2
19	NaN	1	1	1	23	1	1	7.2250	4	0
26	NaN	1	0	0	23	1	1	7.2250	4	0

#建立 Age 的预测模型，我们可以多模型预测，然后再做模型的融合，提高预测的精度。

```
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor


def fill_missing_age(missing_age_train, missing_age_test):
    missing_age_X_train = missing_age_train.drop(['Age'], axis=1)
    missing_age_Y_train = missing_age_train['Age']
    missing_age_X_test = missing_age_test.drop(['Age'], axis=1)


    # model 1   gbm
    gbm_reg = GradientBoostingRegressor(random_state=42)
    gbm_reg_param_grid = {'n_estimators': [2000], 'max_depth': [4], 'learning_rate': [0.01], 'max_features': [3]}
    gbm_reg_grid = model_selection.GridSearchCV(gbm_reg, gbm_reg_param_grid, cv=10, n_jobs=25, verbose=1, scoring='neg_mean_squared_error')
    gbm_reg_grid.fit(missing_age_X_train, missing_age_Y_train)
    print('Age feature Best GB Params:' + str(gbm_reg_grid.best_params_))
    print('Age feature Best GB Score:' + str(gbm_reg_grid.best_score_))
    print('GB Train Error for "Age" Feature Regressor:' + str(gbm_reg_grid.score(missing_age_X_train, missing_age_Y_train)))
    missing_age_test.loc[:, 'Age_GB'] = gbm_reg_grid.predict(missing_age_X_test)
```

```
    print(missing_age_test['Age_GB'][:4])


    # model 2 rf
    rf_reg = RandomForestRegressor()
    rf_reg_param_grid = {'n_estimators': [200], 'max_depth': [5], 'random_state': [0]}
    rf_reg_grid = model_selection.GridSearchCV(rf_reg, rf_reg_param_grid, cv=10, n_jobs=25, verbose=1, scoring='neg_mean_squared_error')
    rf_reg_grid.fit(missing_age_X_train, missing_age_Y_train)
    print('Age feature Best RF Params:' + str(rf_reg_grid.best_params_))
    print('Age feature Best RF Score:' + str(rf_reg_grid.best_score_))
    print('RF Train Error for "Age" Feature Regressor' + str(rf_reg_grid.score(missing_age_X_train, missing_age_Y_train)))
    missing_age_test.loc[:, 'Age_RF'] = rf_reg_grid.predict(missing_age_X_test)
    print(missing_age_test['Age_RF'][:4])


    # two models merge
    print('shape1', missing_age_test['Age'].shape, missing_age_test[['Age_GB', 'Age_RF']].mode(axis=1).shape)
    # missing_age_test['Age'] = missing_age_test[['Age_GB', 'Age_LR']].mode(axis=1)

    missing_age_test.loc[:, 'Age'] = np.mean([missing_age_test['Age_GB'], missing_age_test['Age_RF']])
    print(missing_age_test['Age'][:4])

    missing_age_test.drop(['Age_GB', 'Age_RF'], axis=1, inplace=True)

    return missing_age_test
```

#利用融合模型预测的结果填充 Age 的缺失值：

```
combined_train_test.loc[(combined_train_test.Age.isnull()), 'Age'] = fill_missing_age(missing_age_train, missing_age_test)
```

#输出结果：

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
[Parallel(n_jobs=25)]: Done    5 out of   10 | elapsed:    3.9s remaining:    3.9s
[Parallel(n_jobs=25)]: Done   10 out of   10 | elapsed:    6.9s finished
Age feature Best GB Params:{'n_estimators': 2000, 'learning_rate': 0.01, 'max_features': 3, 'max_depth': 4}
Age feature Best GB Score:-130.295677599
GB Train Error for "Age" Feature Regressor:-64.6566961723
5      35.773942
17     31.489153
19     34.113840
26     28.621281
Name: Age_GB, dtype: float64
Fitting 10 folds for each of 1 candidates, totalling 10 fits
[Parallel(n_jobs=25)]: Done    5 out of   10 | elapsed:    6.2s remaining:    6.2s
[Parallel(n_jobs=25)]: Done   10 out of   10 | elapsed:   10.7s finished

Age feature Best RF Params:{'n_estimators': 200, 'random_state': 0, 'max_depth': 5}
Age feature Best RF Score:-119.094956052
RF Train Error for "Age" Feature Regressor-96.0603148448
5      33.459421
17     33.076798
19     34.855942
26     28.146718
Name: Age_RF, dtype: float64
shape1 (263,) (263, 2)
5      30.000675
17     30.000675
19     30.000675
26     30.000675
Name: Age, dtype: float64
```

#Ticket 字段

#观察 Ticket 的值，我们可以看到，Ticket 有字母和数字之分，而对于不同的字母，可能在很大程度上就意味着船舱等级或者不同船舱的位置，也会对 Survived 产生一定的影响，所以我们将 Ticket 中的字母分开，为数字的部分则分为一类。

```
combined_train_test['Ticket_Letter'] = combined_train_test['Ticket'].str.split().str[0]
combined_train_test['Ticket_Letter'] = combined_train_test['Ticket_Letter'].apply(lambda x: 'U0' if x.isnumeric() else x)
```

如果要提取数字信息，则也可以这样做，现在我们对数字票单纯地分为一类。

combined_train_test['Ticket_Number'] = combined_train_test['Ticket'].apply(lambda x: pd.to_numeric(x, errors='coerce'))

combined_train_test['Ticket_Number'].fillna(0, inplace=True)

将 Ticket_Letter factorize

combined_train_test['Ticket_Letter'] = pd.factorize(combined_train_test['Ticket_Letter'])[0]

Cabin 字段，因为 Cabin 项的缺失值确实太多了，我们很难对其进行分析，或者预测。所以这里我们可以直接将 Cabin 这一项特征去除。但通过上面的分析，可以知道，该特征信息的有无也与生存率有一定的关系，所以这里我们暂时保留该特征，并将其分为有和无两类。

combined_train_test.loc[combined_train_test.Cabin.isnull(), 'Cabin'] = 'U0'

combined_train_test['Cabin'] = combined_train_test['Cabin'].apply(lambda x: 0 if x == 'U0' else 1)

#特征间相关性分析

#我们挑选一些主要的特征，生成特征之间的关联图，查看特征与特征之间的相关性：

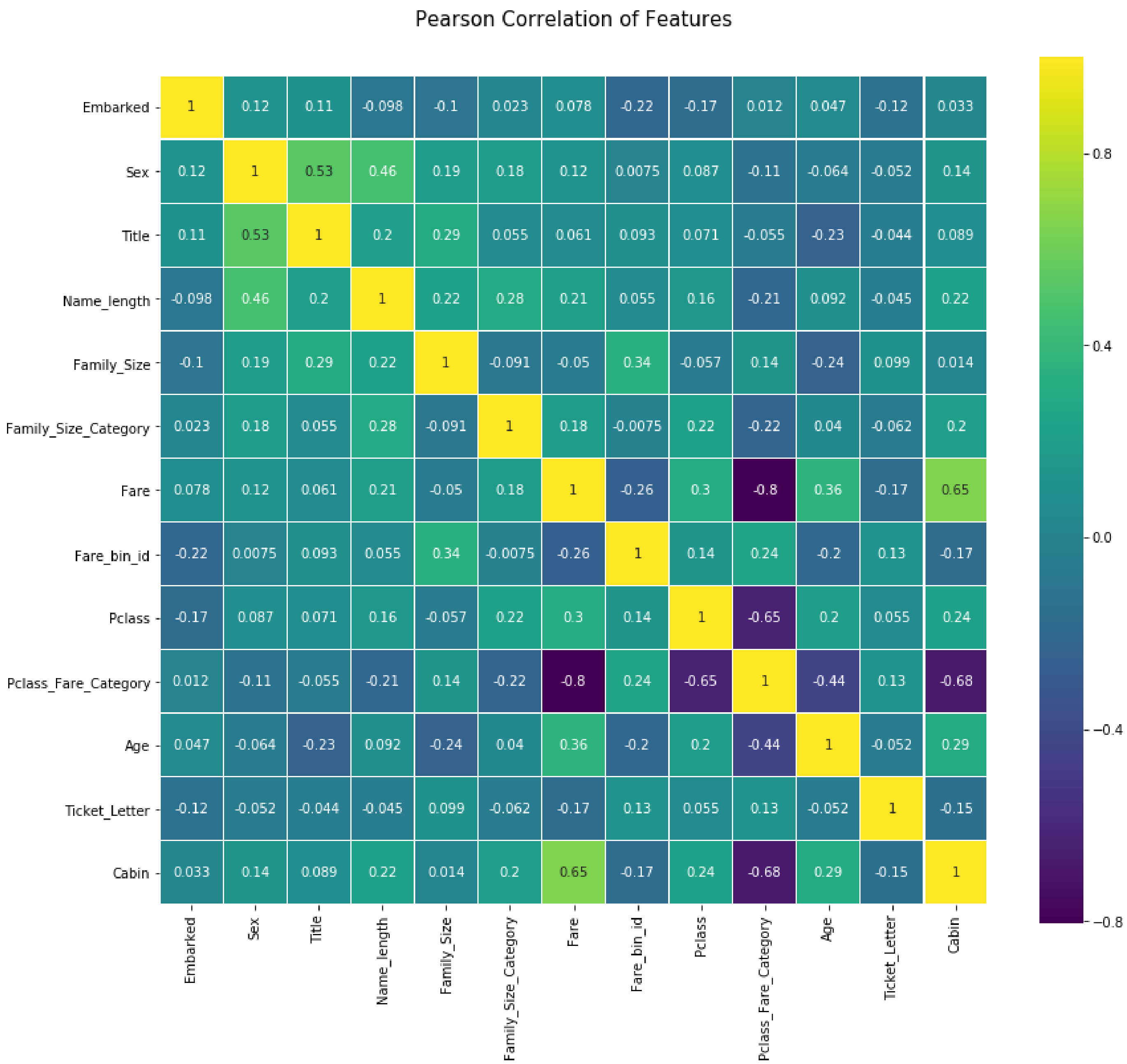
```
Correlation = pd.DataFrame(combined_train_test[
    ['Embarked', 'Sex', 'Title', 'Name_length', 'Family_Size', 'Family_Size_Category','Fare', 'Fare_bin_id', 'Pclass',
     'Pclass_Fare_Category', 'Age', 'Ticket_Letter', 'Cabin']])
```

colormap = plt.cm.viridis

plt.figure(figsize=(14,12))

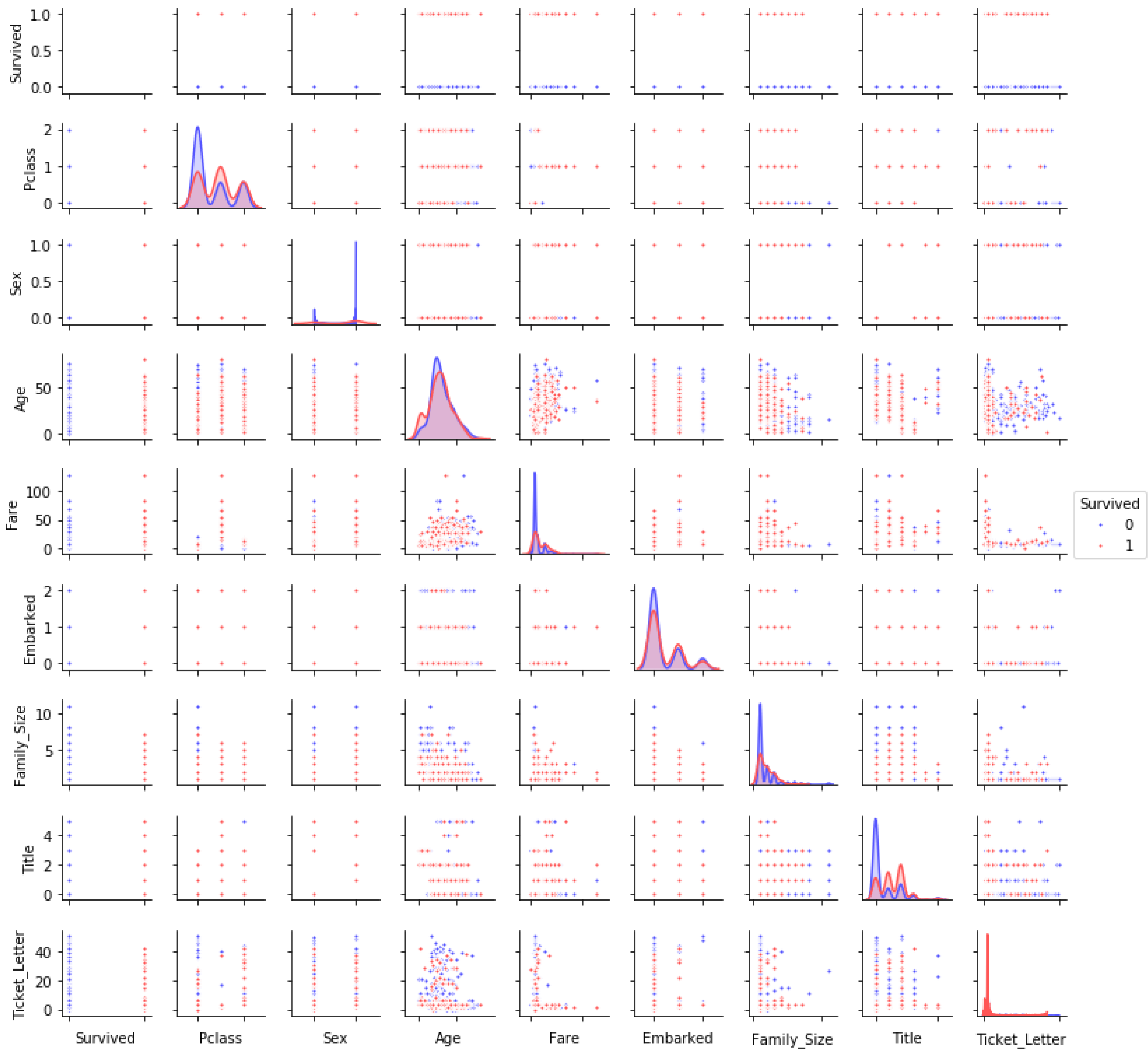
plt.title('Pearson Correlation of Features', y=1.05, size=15)

sns.heatmap(Correlation.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)



#特征之间的数据分布图

```
g = sns.pairplot(combined_train_test[[u'Survived', u'Pclass', u'Sex', u'Age', u'Fare', u'Embarked',
        u'Family_Size', u'Title', u'Ticket_Letter']], hue='Survived', palette = 'seismic',size=1.2,diag_kind =
'kde',diag_kws=dict(shade=True),plot_kws=dict(s=10) )
g.set(xticklabels=[])
```



#输入模型前的一些处理：

#一些数据的正则化：这里我们将 Age 和 fare 进行正则化

scale_age_fare = preprocessing.StandardScaler().fit(combined_train_test[['Age','Fare', 'Name_length']])

combined_train_test[['Age','Fare', 'Name_length']] =

scale_age_fare.transform(combined_train_test[['Age','Fare', 'Name_length']])

#弃掉无用特征

#对于上面的特征工程中，我们从一些原始的特征中提取出了很多要融合到模型中的特征，但是我们需要剔除那些原本的我们用不到的或者非数值特征：

#首先对我们的数据先进行一下备份，以便后期的再次分析：

```
combined_data_backup = combined_train_test

combined_train_test.drop(['PassengerId', 'Embarked', 'Sex', 'Name', 'Title', 'Fare_bin_id', 'Pclass_Fare_Category',
                          'Parch', 'SibSp', 'Family_Size_Category', 'Ticket'],axis=1,inplace=True)
```

#将训练数据和测试数据分开

```
train_data = combined_train_test[:891]

test_data = combined_train_test[891:]


titanic_train_data_X = train_data.drop(['Survived'],axis=1)

titanic_train_data_Y = train_data['Survived']

titanic_test_data_X = test_data.drop(['Survived'],axis=1)

titanic_train_data_X.shape
```

#输出结果如下：

    #(891, 32)

#模型融合及测试

#利用不同的模型来对特征进行筛选，选出较为重要的特征：

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.tree import DecisionTreeClassifier


def get_top_n_features(titanic_train_data_X, titanic_train_data_Y, top_n_features):

    # random forest
    rf_est = RandomForestClassifier(random_state=0)
    rf_param_grid = {'n_estimators': [500], 'min_samples_split': [2, 3], 'max_depth': [20]}
    rf_grid = model_selection.GridSearchCV(rf_est, rf_param_grid, n_jobs=25, cv=10, verbose=1)
    rf_grid.fit(titanic_train_data_X, titanic_train_data_Y)
    print('Top N Features Best RF Params:' + str(rf_grid.best_params_))
    print('Top N Features Best RF Score:' + str(rf_grid.best_score_))
    print('Top N Features RF Train Score:' + str(rf_grid.score(titanic_train_data_X, titanic_train_data_Y)))
    feature_imp_sorted_rf = pd.DataFrame({'feature': list(titanic_train_data_X),
                                          'importance': rf_grid.best_estimator_.feature_importances_}).sort_values('importance',
ascending=False)
    features_top_n_rf = feature_imp_sorted_rf.head(top_n_features)['feature']
    print('Sample 10 Features from RF Classifier')
    print(str(features_top_n_rf[:10]))

    # AdaBoost
    ada_est =AdaBoostClassifier(random_state=0)
    ada_param_grid = {'n_estimators': [500], 'learning_rate': [0.01, 0.1]}
    ada_grid = model_selection.GridSearchCV(ada_est, ada_param_grid, n_jobs=25, cv=10, verbose=1)
    ada_grid.fit(titanic_train_data_X, titanic_train_data_Y)
    print('Top N Features Best Ada Params:' + str(ada_grid.best_params_))
    print('Top N Features Best Ada Score:' + str(ada_grid.best_score_))
    print('Top N Features Ada Train Score:' + str(ada_grid.score(titanic_train_data_X, titanic_train_data_Y)))
    feature_imp_sorted_ada = pd.DataFrame({'feature': list(titanic_train_data_X),
                                           'importance':
ada_grid.best_estimator_.feature_importances_}).sort_values('importance', ascending=False)
    features_top_n_ada = feature_imp_sorted_ada.head(top_n_features)['feature']
    print('Sample 10 Feature from Ada Classifier:')
    print(str(features_top_n_ada[:10]))

    # ExtraTree
    et_est = ExtraTreesClassifier(random_state=0)
    et_param_grid = {'n_estimators': [500], 'min_samples_split': [3, 4], 'max_depth': [20]}
    et_grid = model_selection.GridSearchCV(et_est, et_param_grid, n_jobs=25, cv=10, verbose=1)
    et_grid.fit(titanic_train_data_X, titanic_train_data_Y)
    print('Top N Features Best ET Params:' + str(et_grid.best_params_))
    print('Top N Features Best ET Score:' + str(et_grid.best_score_))
    print('Top N Features ET Train Score:' + str(et_grid.score(titanic_train_data_X, titanic_train_data_Y)))
    feature_imp_sorted_et = pd.DataFrame({'feature': list(titanic_train_data_X),
                                          'importance': et_grid.best_estimator_.feature_importances_}).sort_values('importance',
ascending=False)
    features_top_n_et = feature_imp_sorted_et.head(top_n_features)['feature']
    print('Sample 10 Features from ET Classifier:')
```

```
    print(str(features_top_n_et[:10]))

    # GradientBoosting
    gb_est =GradientBoostingClassifier(random_state=0)
    gb_param_grid = {'n_estimators': [500], 'learning_rate': [0.01, 0.1], 'max_depth': [20]}
    gb_grid = model_selection.GridSearchCV(gb_est, gb_param_grid, n_jobs=25, cv=10, verbose=1)
    gb_grid.fit(titanic_train_data_X, titanic_train_data_Y)
    print('Top N Features Best GB Params:' + str(gb_grid.best_params_))
    print('Top N Features Best GB Score:' + str(gb_grid.best_score_))
    print('Top N Features GB Train Score:' + str(gb_grid.score(titanic_train_data_X, titanic_train_data_Y)))
    feature_imp_sorted_gb = pd.DataFrame({'feature': list(titanic_train_data_X),
                                          'importance': gb_grid.best_estimator_.feature_importances_}).sort_values('importance',
ascending=False)
    features_top_n_gb = feature_imp_sorted_gb.head(top_n_features)['feature']
    print('Sample 10 Feature from GB Classifier:')
    print(str(features_top_n_gb[:10]))

    # DecisionTree
    dt_est = DecisionTreeClassifier(random_state=0)
    dt_param_grid = {'min_samples_split': [2, 4], 'max_depth': [20]}
    dt_grid = model_selection.GridSearchCV(dt_est, dt_param_grid, n_jobs=25, cv=10, verbose=1)
    dt_grid.fit(titanic_train_data_X, titanic_train_data_Y)
    print('Top N Features Best DT Params:' + str(dt_grid.best_params_))
    print('Top N Features Best DT Score:' + str(dt_grid.best_score_))
    print('Top N Features DT Train Score:' + str(dt_grid.score(titanic_train_data_X, titanic_train_data_Y)))
    feature_imp_sorted_dt = pd.DataFrame({'feature': list(titanic_train_data_X),
                                          'importance':   dt_grid.best_estimator_.feature_importances_}).sort_values('importance',
ascending=False)
    features_top_n_dt = feature_imp_sorted_dt.head(top_n_features)['feature']
    print('Sample 10 Features from DT Classifier:')
    print(str(features_top_n_dt[:10]))

    # merge the three models
    features_top_n = pd.concat([features_top_n_rf, features_top_n_ada, features_top_n_et, features_top_n_gb, features_top_n_dt],
                               ignore_index=True).drop_duplicates()

    features_importance = pd.concat([feature_imp_sorted_rf, feature_imp_sorted_ada, feature_imp_sorted_et,
                                     feature_imp_sorted_gb, feature_imp_sorted_dt],ignore_index=True)

    return features_top_n , features_importance
```

#依据我们筛选出的特征构建训练集和测试集

#但如果在进行特征工程的过程中，产生了大量的特征，而特征与特征之间会存在一定的相关性。

#太多的特征一方面会影响模型训练的速度，另一方面也可能会使得模型过拟合。

#所以在特征太多的情况下，我们可以利用不同的模型对特征进行筛选，选取出我们想要的前 n 个特征。

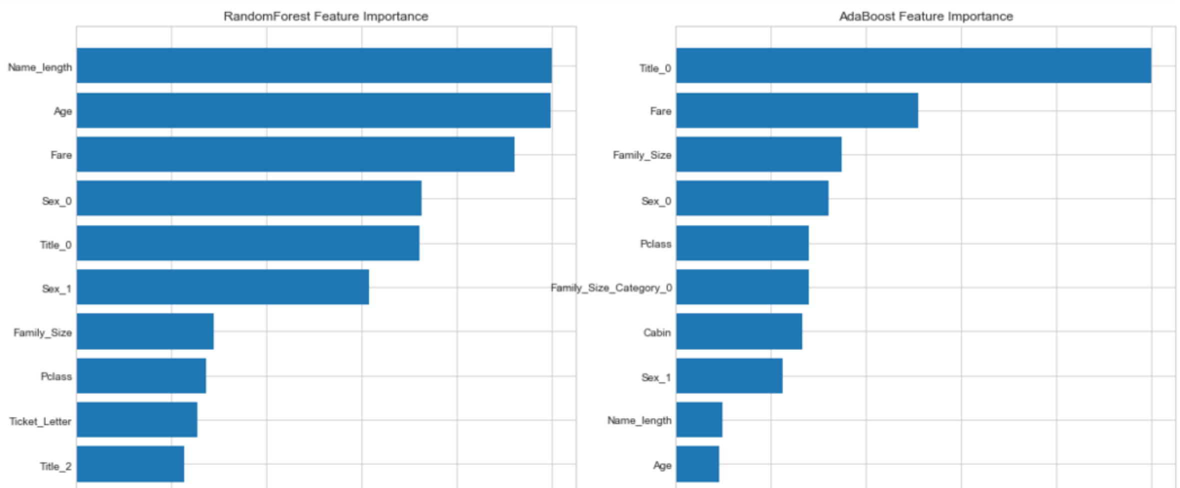
```
feature_to_pick = 30
feature_top_n, feature_importance = get_top_n_features(titanic_train_data_X, titanic_train_data_Y, feature_to_pick)
titanic_train_data_X = pd.DataFrame(titanic_train_data_X[feature_top_n])
titanic_test_data_X = pd.DataFrame(titanic_test_data_X[feature_top_n])
```

#用视图可视化不同算法筛选的特征排序：

```
rf_feature_imp = feature_importance[:10]
Ada_feature_imp = feature_importance[32:32+10].reset_index(drop=True)


# make importances relative to max importance
rf_feature_importance = 100.0 * (rf_feature_imp['importance'] / rf_feature_imp['importance'].max())
Ada_feature_importance = 100.0 * (Ada_feature_imp['importance'] /
Ada_feature_imp['importance'].max())


# Get the indexes of all features over the importance threshold
rf_important_idx = np.where(rf_feature_importance)[0]
Ada_important_idx = np.where(Ada_feature_importance)[0]


pos = np.arange(rf_important_idx.shape[0]) + .5


plt.figure(1, figsize = (18, 8))


plt.subplot(121)
plt.barh(pos, rf_feature_importance[rf_important_idx][::-1])
```

```
plt.yticks(pos, rf_feature_imp['feature'][::-1])

plt.xlabel('Relative Importance')

plt.title('RandomForest Feature Importance')


plt.subplot(122)

plt.barh(pos, Ada_feature_importance[Ada_important_idx][::-1])

plt.yticks(pos, Ada_feature_imp['feature'][::-1])

plt.xlabel('Relative Importance')

plt.title('AdaBoost Feature Importance')

plt.show()
```

#得到不同特征针对目标的重要性排序



6、　模型融合

模型融合（Model Ensemble）

常见的模型融合方法有：Bagging、Boosting、Stacking、Blending。

(6-1):Bagging

Bagging 将多个模型，也就是多个基学习器的预测结果进行简单的加权平均或者投票。它的好处是可以并行地训练基学习器。Random Forest 就用到了 Bagging 的思想。

(6-2): Boosting

Boosting 的思想，每个基学习器是在上一个基学习器学习的基础上，对上一个基学习器的错误进行弥补。我们将会用到的 AdaBoost，Gradient Boost 就用到了这种思想。

(6-3): Stacking

Stacking 是用新的学习器去学习如何组合上一层的基学习器。如果把 Bagging 看作是多个基分类器的线性组合，那么 Stacking 就是多个基分类器的非线性组合。Stacking 可以将学习器一层一层地堆砌起来，形成一个网状的结构。

相比来说 Stacking 的融合框架相对前面的二者来说在精度上确实有一定的提升，所以在下面的模型融合上，我们也使用 Stacking 方法。

(6-4): Blending

Blending 和 Stacking 很相似，但同时它可以防止信息泄露的问题。

Stacking 框架融合:

这里我们使用了两层的模型融合：

Level 1 使用了：RandomForest、AdaBoost、ExtraTrees、GBDT、DecisionTree、KNN、SVM ，一共 7 个模型

Level 2 使用了 XGBoost 使用第一层预测的结果作为特征对最终的结果进行预测。

Level 1：

Stacking 框架是堆叠使用基础分类器的预测作为对二级模型的训练的输入。 然而，我们不能简单地在全部训练数据上训练基本模型，产生预测，输出用于第二层的训练。如果我们在 Train Data 上训练，然后在 Train Data 上预测，就会造成标签。为了避免标签，我们需要对每个基学习器使用 K-fold，将 K 个模型对 Valid Set 的预测结果拼起来，作为下一层学习器的输入。

所以这里我们建立输出 fold 预测方法：

```
## L1:这里我们建立输出 fold 预测方法
from sklearn.model_selection import KFold


# Some useful parameters which will come in handy later on
ntrain = titanic_train_data_X.shape[0]
ntest = titanic_test_data_X.shape[0]
SEED = 0 # for reproducibility
NFOLDS = 7 # set folds for out-of-fold prediction
kf = KFold(n_splits = NFOLDS, random_state=SEED, shuffle=False)


def get_out_fold(clf, x_train, y_train, x_test):
    oof_train = np.zeros((ntrain,))
    oof_test = np.zeros((ntest,))
    oof_test_skf = np.empty((NFOLDS, ntest))


    for i, (train_index, test_index) in enumerate(kf.split(x_train)):
        x_tr = x_train[train_index]
        y_tr = y_train[train_index]
        x_te = x_train[test_index]


        clf.fit(x_tr, y_tr)


        oof_train[test_index] = clf.predict(x_te)
```

```
            oof_test_skf[i, :] = clf.predict(x_test)

        oof_test[:] = oof_test_skf.mean(axis=0)
        return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)
```

#构建不同的基学习器，这里我们使用了 RandomForest、AdaBoost、ExtraTrees、GBDT、DecisionTree、KNN、SVM 七个基学习器：(这里的模型可以使用如上面的 GridSearch 方法对模型的超参数进行搜索选择)

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

rf = RandomForestClassifier(n_estimators=500, warm_start=True, max_features='sqrt',max_depth=6,
                            min_samples_split=3, min_samples_leaf=2, n_jobs=-1, verbose=0)

ada = AdaBoostClassifier(n_estimators=500, learning_rate=0.1)

et = ExtraTreesClassifier(n_estimators=500, n_jobs=-1, max_depth=8, min_samples_leaf=2, verbose=0)

gb = GradientBoostingClassifier(n_estimators=500, learning_rate=0.008, min_samples_split=3, min_samples_leaf=2, max_depth=5, verbose=0)

dt = DecisionTreeClassifier(max_depth=8)

knn = KNeighborsClassifier(n_neighbors = 2)
```

```
svm = SVC(kernel='linear', C=0.025)

#将 pandas 转换为 arrays：
# Create Numpy arrays of train, test and target (Survived) dataframes to feed into our models
x_train = titanic_train_data_X.values # Creates an array of the train data
x_test = titanic_test_data_X.values # Creats an array of the test data
y_train = titanic_train_data_Y.values

## Create our OOF train and test predictions. These base results will be used as new features
rf_oof_train, rf_oof_test = get_out_fold(rf, x_train, y_train, x_test) # Random Forest
ada_oof_train, ada_oof_test = get_out_fold(ada, x_train, y_train, x_test) # AdaBoost
et_oof_train, et_oof_test = get_out_fold(et, x_train, y_train, x_test) # Extra Trees
gb_oof_train, gb_oof_test = get_out_fold(gb, x_train, y_train, x_test) # Gradient Boost
dt_oof_train, dt_oof_test = get_out_fold(dt, x_train, y_train, x_test) # Decision Tree
knn_oof_train, knn_oof_test = get_out_fold(knn, x_train, y_train, x_test) # KNeighbors
svm_oof_train, svm_oof_test = get_out_fold(svm, x_train, y_train, x_test) # Support Vector
print("Training is complete")

#预测并生成提交文件
#Level 2：
#利用 XGBoost，使用第一层预测的结果作为特征对最终的结果进行预测。
x_train = np.concatenate((rf_oof_train, ada_oof_train, et_oof_train, gb_oof_train, dt_oof_train, knn_oof_train, svm_oof_train), axis=1)
x_test = np.concatenate((rf_oof_test, ada_oof_test, et_oof_test, gb_oof_test, dt_oof_test, knn_oof_test, svm_oof_test), axis=1)
```

```
from xgboost import XGBClassifier

gbm = XGBClassifier( n_estimators= 2000, max_depth= 4, min_child_weight= 2, gamma=0.9, subsample=0.8,
                        colsample_bytree=0.8, objective= 'binary:logistic', nthread= -1, scale_pos_weight=1).fit(x_train, y_train)
predictions = gbm.predict(x_test)

StackingSubmission = pd.DataFrame({'PassengerId': PassengerId, 'Survived': predictions})
StackingSubmission.to_csv('StackingSubmission.csv',index=False,sep=',')
```

7、 验证训练误差和泛化能力：学习曲线

在我们对数据不断地进行特征工程，产生的特征越来越多，用大量的特征对模型进行训练，会使我们的训练集拟合得越来越好，但同时也可能会逐渐丧失泛化能力，从而在测试数据上表现不佳，发生过拟合现象。当然我们建立的模型可能不仅在预测集上表现不好，也很可能是因为在训练集上的表现就不佳，处于欠拟合状态。

对于 Stacking 框架中第一层的各个基学习器我们都应该对其学习曲线进行观察，从而去更好地调节超参数，进而得到更好的最终结果。

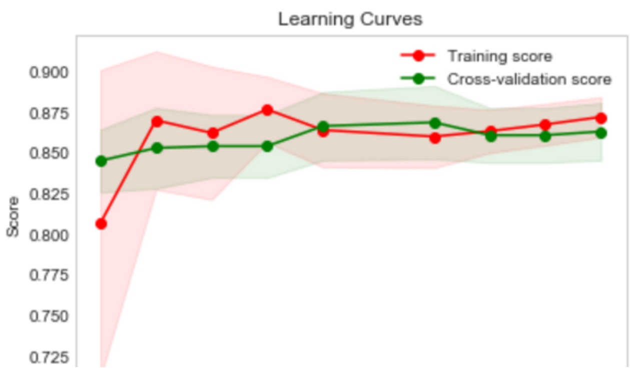
构建绘制学习曲线的函数：

```
#画出学习曲线：训练误差，泛化误差
from sklearn.learning_curve import learning_curve

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5), verbose=0):
    """
    Generate a simple plot of the test and traning learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
```

```
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : integer, cross-validation generator, optional
        If an integer is passed, it is the number of folds (defaults to 3).
        Specific cross-validation objects can be passed, see
        sklearn.cross_validation module for the list of possible objects

    n_jobs : integer, optional
        Number of jobs to run in parallel (default 1).
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt

#逐一观察不同模型的学习曲线：
X = x_train
Y = y_train

# RandomForest
```

```
rf_parameters = {'n_jobs': -1, 'n_estimators': 500, 'warm_start': True, 'max_depth': 6, 'min_samples_leaf': 2,
                 'max_features' : 'sqrt','verbose': 0}

# AdaBoost
ada_parameters = {'n_estimators':500, 'learning_rate':0.1}

# ExtraTrees
et_parameters = {'n_jobs': -1, 'n_estimators':500, 'max_depth': 8, 'min_samples_leaf': 2, 'verbose': 0}

# GradientBoosting
gb_parameters = {'n_estimators': 500, 'max_depth': 5, 'min_samples_leaf': 2, 'verbose': 0}

# DecisionTree
dt_parameters = {'max_depth':8}

# KNeighbors
knn_parameters = {'n_neighbors':2}

# SVM
svm_parameters = {'kernel':'linear', 'C':0.025}

# XGB
gbm_parameters = {'n_estimators': 2000, 'max_depth': 4, 'min_child_weight': 2, 'gamma':0.9, 'subsample':0.8,
                  'colsample_bytree':0.8, 'objective': 'binary:logistic', 'nthread':-1, 'scale_pos_weight':1}


title = "Learning Curves"
plot_learning_curve(RandomForestClassifier(**rf_parameters), title, X, Y, cv=None, n_jobs=4, train_sizes=[50, 100,
150, 200, 250, 350, 400, 450, 500])
plt.show()
```

#从图中看出，训练和测试样本准确性的评分



可以根据以上思路进一步的调整特征工程和模型的参数，直至训练误差和测试误差达到最小。