Titanic : Machine Learning from Disaster

链接：[GitHub源代码]()

> ## Question
>
> - 要求你建立一个预测模型来回答这个问题：“什么样的人更有可能生存？”使用乘客数据（如姓名、年龄、性别、社会经济阶层等）。

一、导入数据包和数据集

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

> - **重点：在kaggle notebook上时，应该把** `pd.read_csv("./kaggle/input/titanic/train.csv")` **引号中第一个** `'.'` **去掉**
> - 读入训练集和测试及都需要

```
train = pd.read_csv("./kaggle/input/titanic/train.csv")
test = pd.read_csv("./kaggle/input/titanic/test.csv")
allData = pd.concat([train, test], ignore_index=True)
# dataNum = train.shape[0]
# featureNum = train.shape[1]
train.info()
```

二、数据总览

概况

- 输入`train.info()`回车可以查看数据集整体信息

```
<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
```

- 输入`train.head()`可以查看数据样例

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

特征

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class(客舱等级)	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic(旁系亲属)	
parch	# of parents / children aboard the Titanic(直系亲属)	
ticket	Ticket number	
fare	Passenger fare	

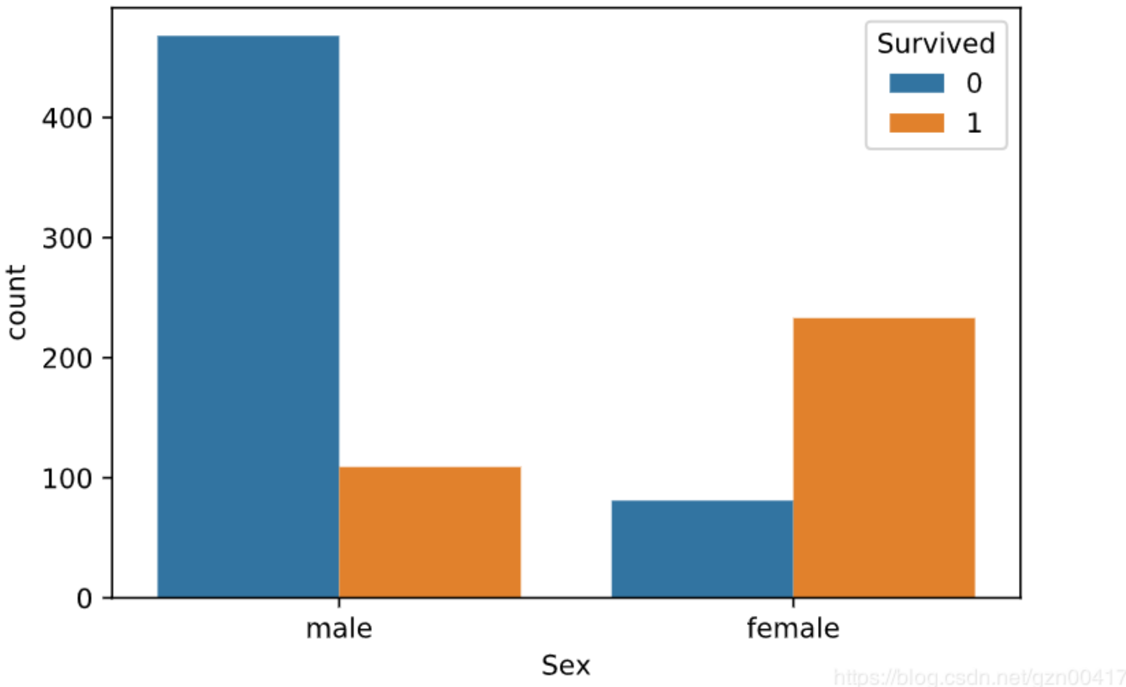
Variable	Definition	Key
cabin	Cabin number(客舱编号)	
embarked	Port of Embarkation(上船港口编号)	C = Cherbourg, Q = Queenstown, S = Southampton

三、可视化数据分析

性别特征Sex

- 女性生存率远高于男性

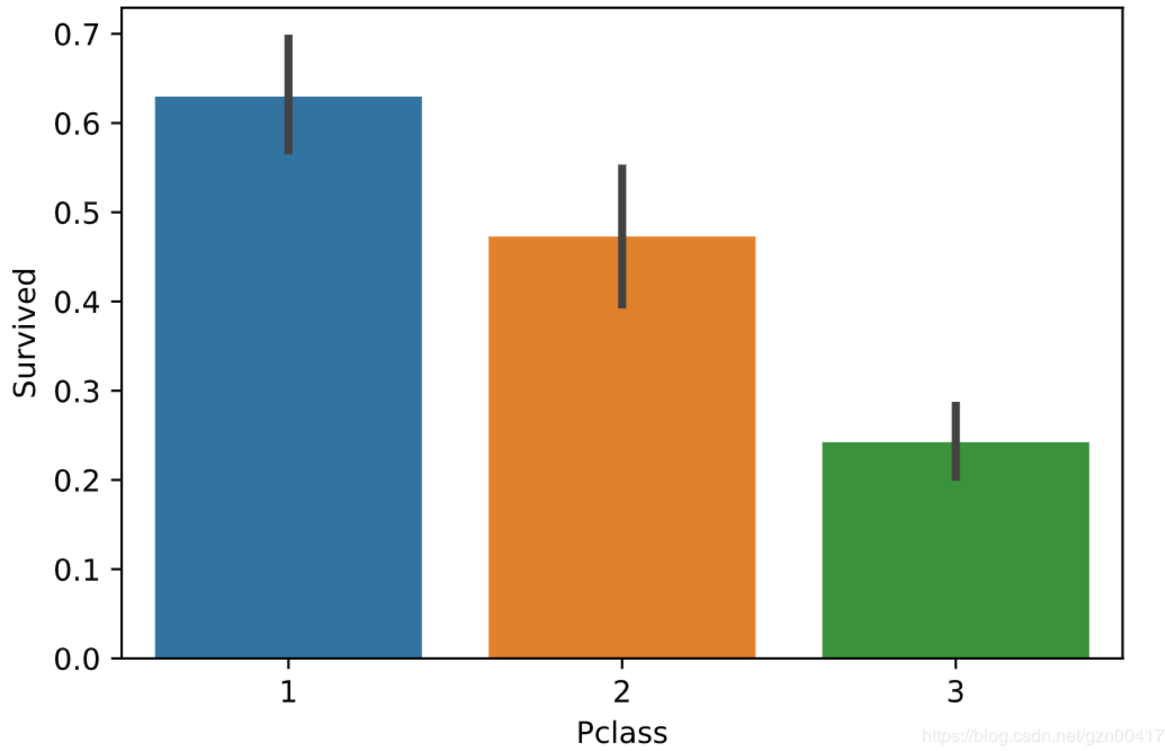
```
# Sex
sns.countplot('Sex', hue='Survived', data=train)
plt.show()
```



等级特征Pclass

- 乘客等级越高，生存率越高

```
# Pclass
sns.barplot(x='Pclass', y="Survived", data=train)
plt.show()
```

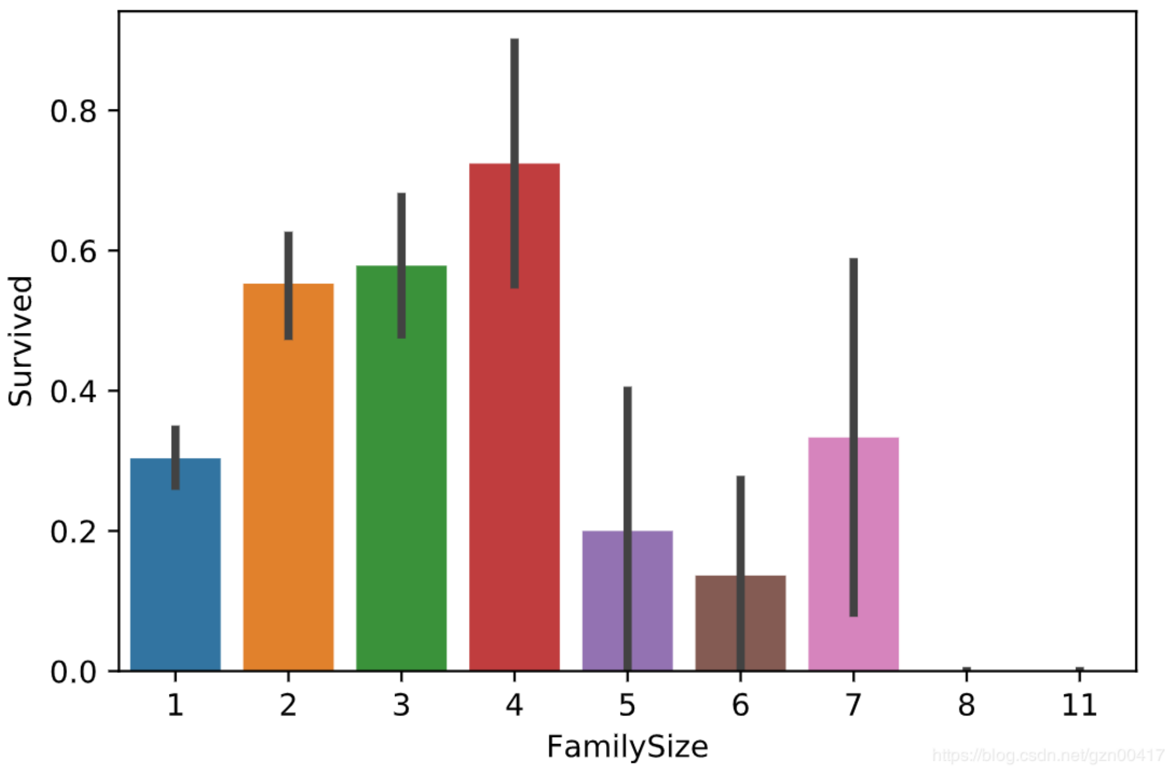


家庭成员数量特征

- **FamilySize=Parch+SibSp**

- 家庭成员数量适中，生存率高

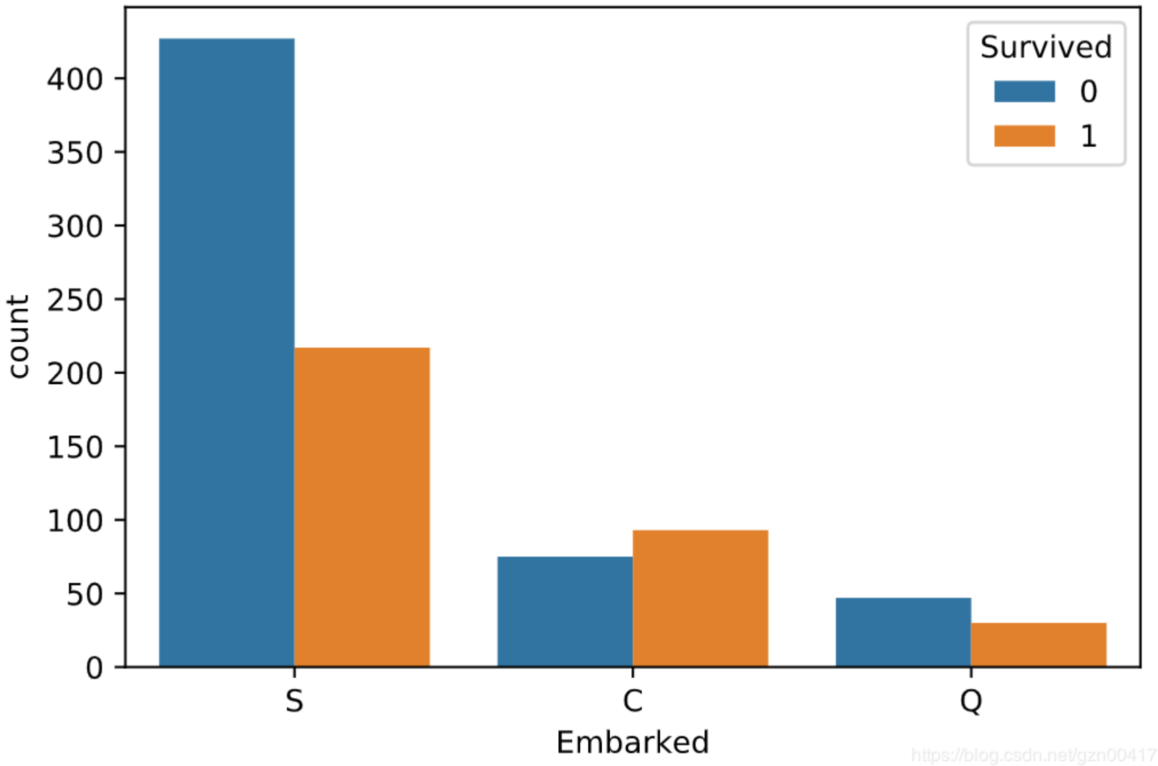
```
# FamilySize = SibSp + Parch + 1
allData['FamilySize'] = allData['SibSp'] + allData['Parch'] + 1
sns.barplot(x='FamilySize', y='Survived', data=allData)
plt.show()
```



上船港口特征Embarked

- 上船港口不同，生存率不同

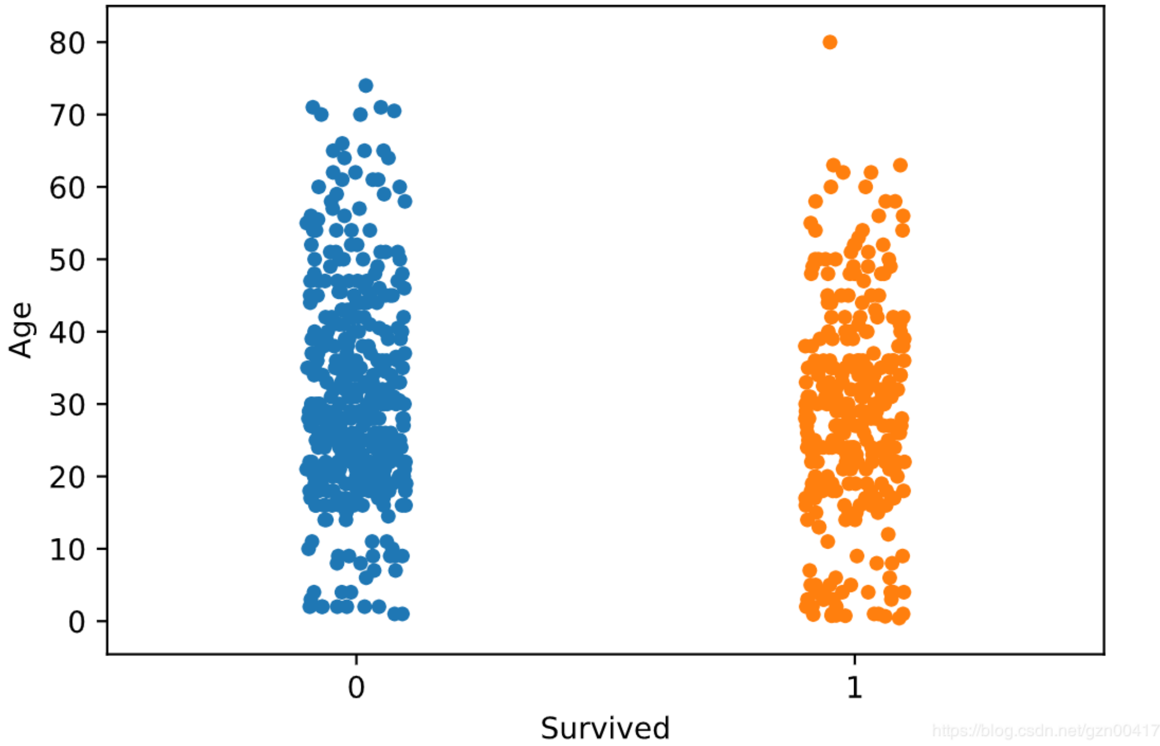
```
# Embarked
sns.countplot('Embarked', hue='Survived', data=train)
plt.show()
```



年龄特征Age

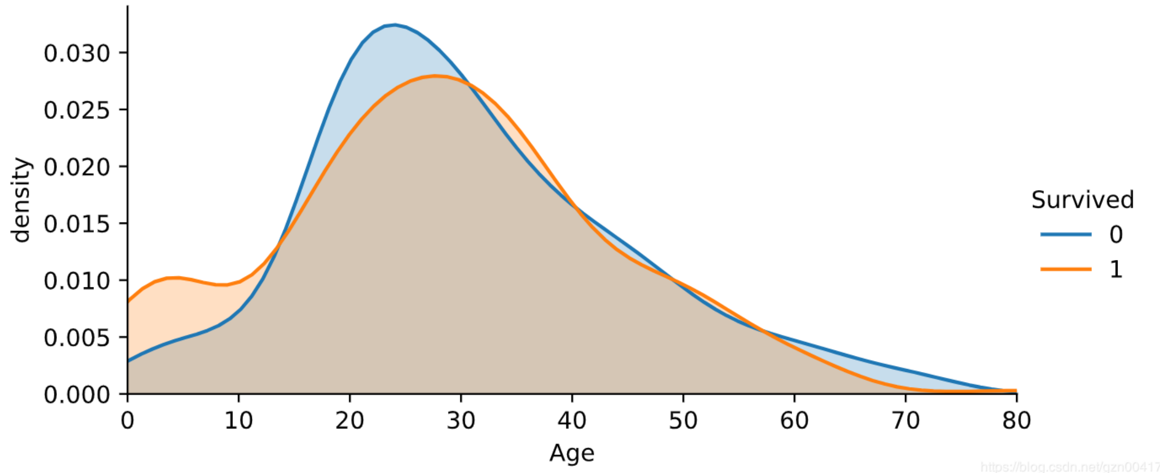
- 年龄小或者正值壮年生存率高

```
# Age
sns.stripplot(x="Survived", y="Age", data=train, jitter=True)
plt.show()
```



- 年龄生存密度

```
facet = sns.FacetGrid(train, hue="Survived",aspect=2)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, train['Age'].max()))
facet.add_legend()
plt.xlabel('Age')
plt.ylabel('density')
plt.show()
```

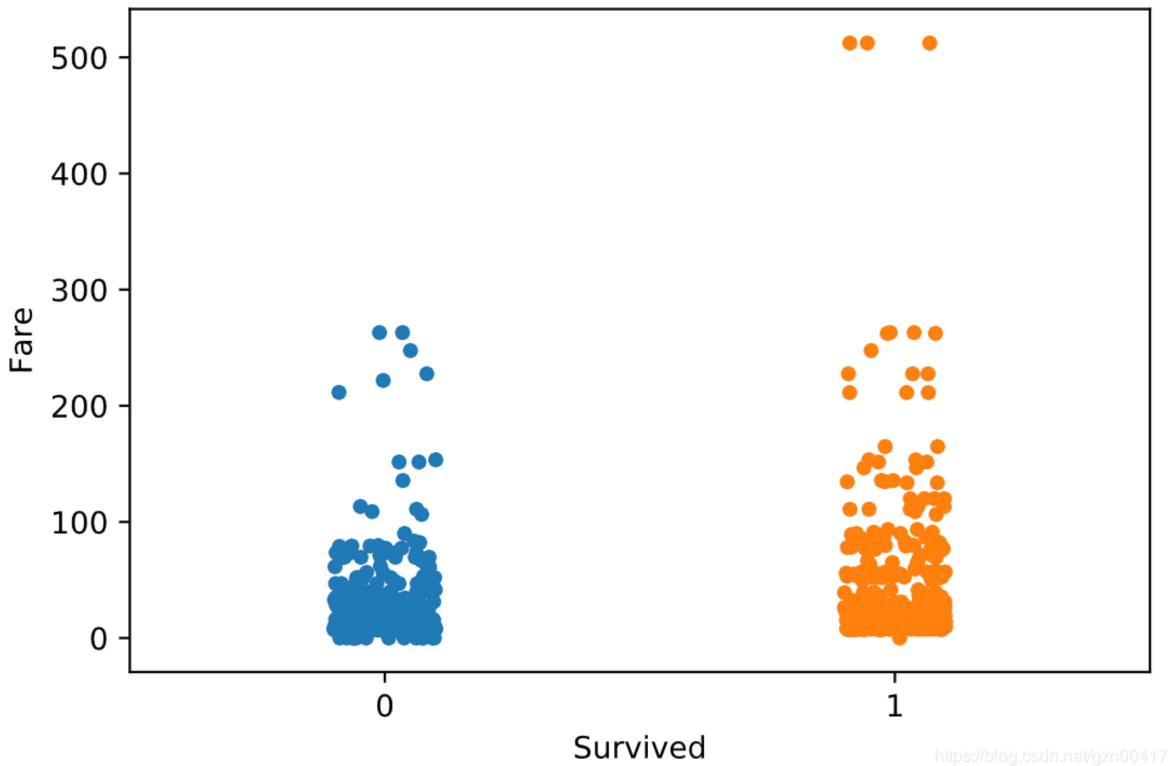


> - 儿童相对于全年龄段有特殊的生存率
> - **作者将10及以下视为儿童，设置单独标签**

费用特征Fare

- 费用越高，生存率越高

```
# Fare
sns.stripplot(x="Survived", y="Fare", data=train, jitter=True)
plt.show()
```



姓名特征Name

头衔特征Title

- 头衔由姓名的前置称谓进行分类

```
# Name
allData['Title'] = allData['Name'].apply(lambda x:x.split(',')[1].split('.')
[0].strip())
pd.crosstab(allData['Title'], allData['Sex'])
```

- 统计分析

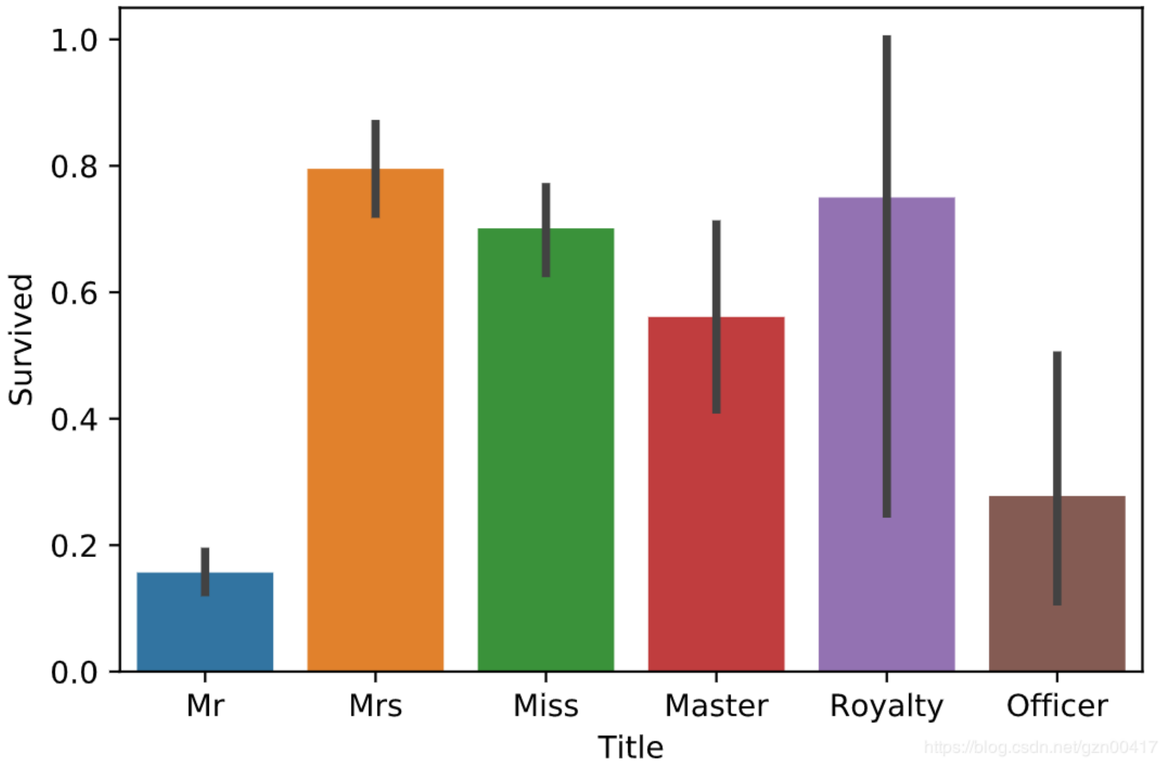
```
TitleClassification = {'Officer':['Capt', 'Col', 'Major', 'Dr', 'Rev'],
                       'Royalty':['Don', 'Sir', 'the Countess', 'Dona', 'Lady'],
                       'Mrs':['Mme', 'Ms', 'Mrs'],
                       'Miss':['Mlle', 'Miss'],
                       'Mr':['Mr'],
                       'Master':['Master','Jonkheer']}
for title in TitleClassification.keys():
    cnt = 0
    for name in TitleClassification[title]:
        cnt += allData.groupby(['Title']).size()[name]
    print (title,':',cnt)
```

- 设置标签

```python
TitleClassification = {'Officer':['Capt', 'Col', 'Major', 'Dr', 'Rev'],
                       'Royalty':['Don', 'Sir', 'the Countess', 'Dona', 'Lady'],
                       'Mrs':['Mme', 'Ms', 'Mrs'],
                       'Miss':['Mlle', 'Miss'],
                       'Mr':['Mr'],
                       'Master':['Master','Jonkheer']}
TitleMap = {}
for title in TitleClassification.keys():
    TitleMap.update(dict.fromkeys(TitleClassification[title], title))
allData['Title'] = allData['Title'].map(TitleMap)
```

- 头衔不同，生存率不同

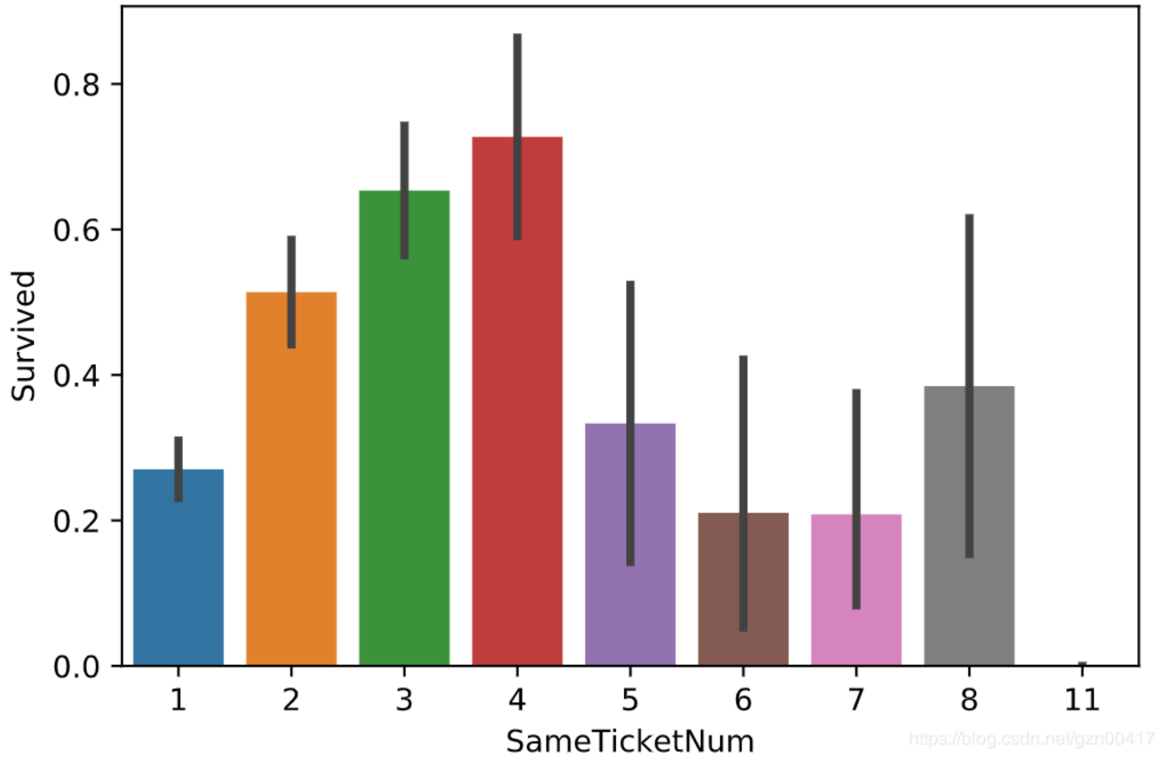
```python
sns.barplot(x="Title", y="Survived", data=allData)
plt.show()
```



票号特征Ticket

- 有一定连续座位（存在票号相同的乘客）生存率高

```python
#Ticket
TicketCnt = allData.groupby(['Ticket']).size()
allData['SameTicketNum'] = allData['Ticket'].apply(lambda x:TicketCnt[x])
sns.barplot(x='SameTicketNum', y='Survived', data=allData)
plt.show()
# allData['SameTicketNum']
```

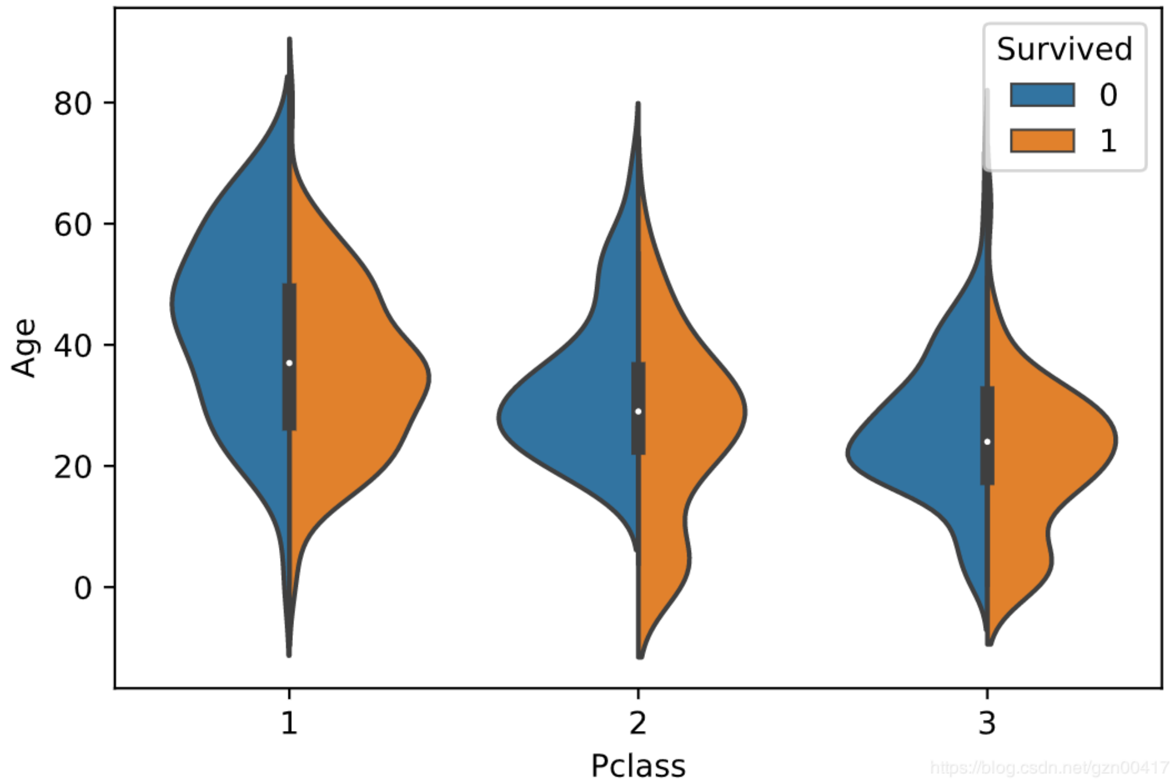


二维/多维分析

- 可以将任意两个/多个数据进行分析

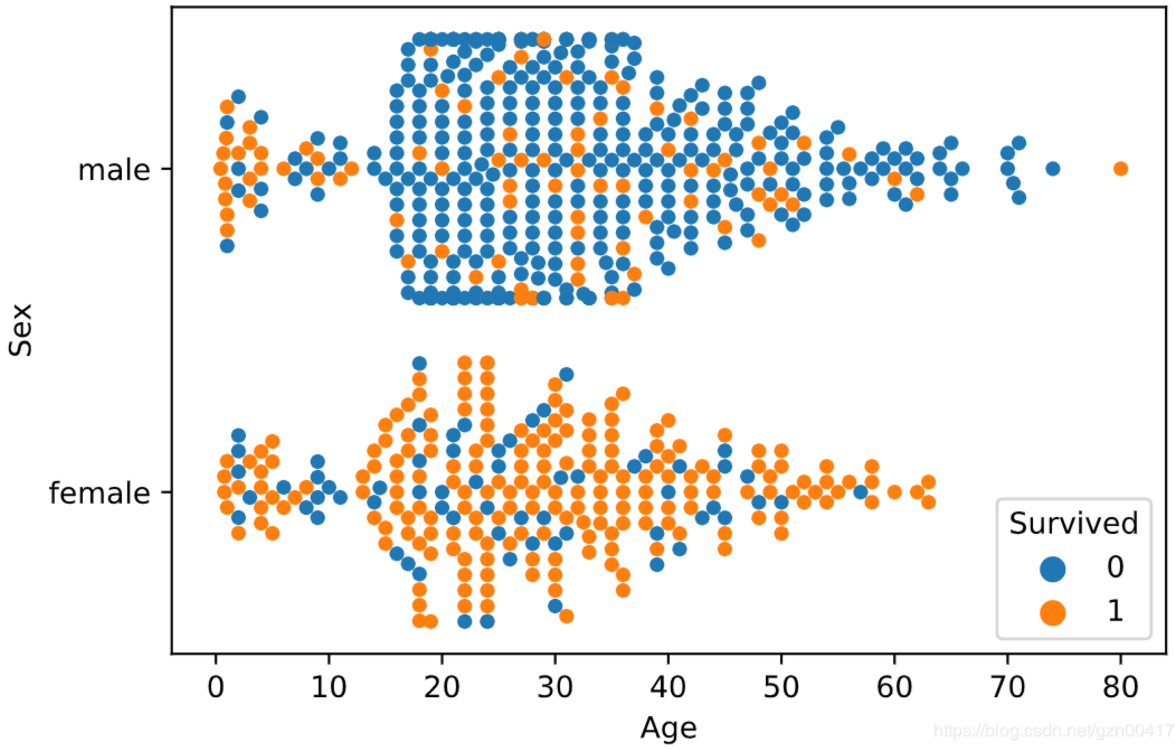
二维分析之Pclass & Age

```
# Pclass & Age
sns.violinplot("Pclass", "Age", hue="Survived", data=train, split=True)
plt.show()
```



二维分析之Age & Sex

```
# Age & Sex
sns.swarmplot(x='Age', y="Sex", data=train, hue='Survived')
plt.show()
```



四、数据清洗 & 异常处理

离散型数据

有可用标签 --> One-Hot编码

- Sex & Pclass & Embarked 都有已经设置好的标签（int或float或string等），可以直接进行get_dummies，拆分成多维向量，增加特征维度
- 其中，Embarked存在一定缺失值，通过对整体的分析，填充上估计值

```
# Sex
allData = allData.join(pd.get_dummies(allData['Sex'], prefix="Sex"))
# Pclass
allData = allData.join(pd.get_dummies(allData['Pclass'], prefix="Pclass"))
# Embarked
allData[allData['Embarked'].isnull()] # 查看缺失值
allData.groupby(by=['Pclass','Embarked']).Fare.mean() # Pclass=1, Embark=C, 中位
数=76
allData['Embarked'] = allData['Embarked'].fillna('C')
allData = allData.join(pd.get_dummies(allData['Embarked'], prefix="Embarked"))
```

无可用标签 --> 设计标签 --> One-Hot

- FamilySize & Name & Ticket需要对整体数据统一处理，再进行标记

```
# FamilySize
def FamilyLabel(s):
```

```
    if (s == 4):
        return 4
    elif (s == 2 or s == 3):
        return 3
    elif (s == 1 or s == 7):
        return 2
    elif (s == 5 or s == 6):
        return 1
    elif (s < 1 or s > 7):
        return 0
allData['FamilyLabel'] = allData['FamilySize'].apply(FamilyLabel)
allData = allData.join(pd.get_dummies(allData['FamilyLabel'], prefix="Fam"))

# Name
TitleLabelMap = {'Mr':1.0,
                 'Mrs':5.0,
                 'Miss':4.5,
                 'Master':2.5,
                 'Royalty':3.5,
                 'Officer':2.0}
def TitleLabel(s):
    return TitleLabelMap[s]
# allData['TitleLabel'] = allData['Title'].apply(TitleLabel)
allData = allData.join(pd.get_dummies(allData['Title'], prefix="Title"))

# Ticket
def TicketLabel(s):
    if (s == 3 or s == 4):
        return 3
    elif (s == 2 or s == 8):
        return 2
    elif (s == 1 or s == 5 or s == 6 or s ==7):
        return 1
    elif (s < 1 or s > 8):
        return 0
allData['TicketLabel'] = allData['SameTicketNum'].apply(TicketLabel)
allData = allData.join(pd.get_dummies(allData['TicketLabel'], prefix="TicNum"))
```

连续型数据

Age & Fare

- 进行标准化，缩小数据范围，加速梯度下降

```
# Age
allData['Child'] = allData['Age'].apply(lambda x:1 if x <= 10 else 0) # 儿童标签
allData['Age'] = (allData['Age']-allData['Age'].mean())/allData['Age'].std() #
标准化
allData['Age'].fillna(value=0, inplace=True) # 填充缺失值
# Fare
allData['Fare'] = allData['Fare'].fillna(25) # 填充缺失值
allData[allData['Survived'].notnull()]['Fare'] =
allData[allData['Survived'].notnull()]['Fare'].apply(lambda x:300.0 if x>500
else x)
allData['Fare'] = allData['Fare'].apply(lambda x:(x-
allData['Fare'].mean())/allData['Fare'].std())
```

清除无用特征

- 清除无用特征，降低算法复杂度

```
# 清除无用特征
allData.drop(['Cabin', 'PassengerId', 'Ticket', 'Name', 'Title', 'Sex', 'SibSp',
'Parch', 'FamilySize', 'Embarked', 'Pclass', 'Title', 'FamilyLabel',
'SameTicketNum', 'TicketLabel'], axis=1, inplace=True)
```

重新分割训练集/测试集

- 一开始，为了处理方便，作者将训练集和测试集合并，现在根据Survived是否缺失来讲训练集和测试集分开

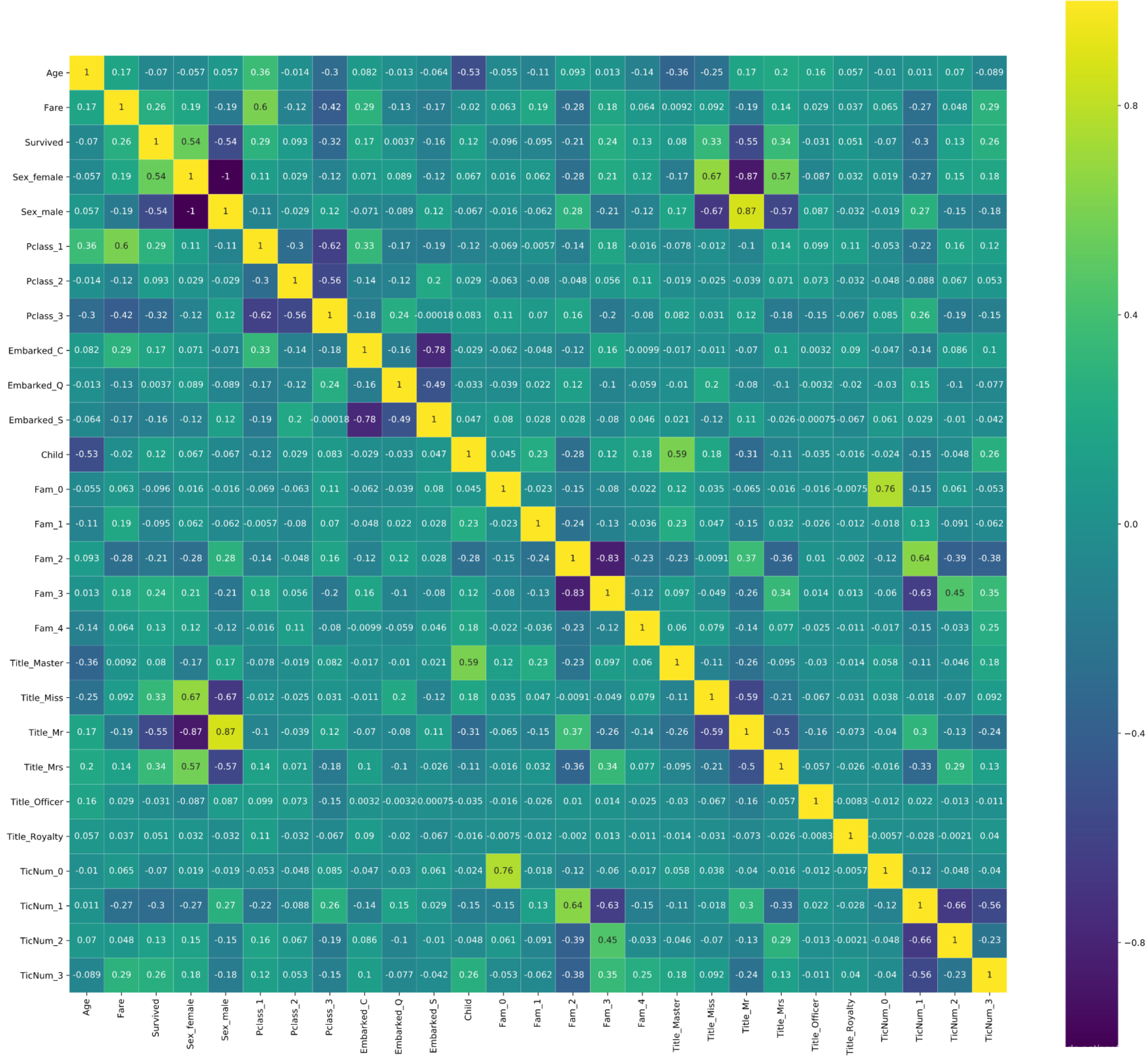
```
# 重新分割数据集
train_data = allData[allData['Survived'].notnull()]
test_data  = allData[allData['Survived'].isnull()]
test_data = test_data.reset_index(drop=True)

xTrain = train_data.drop(['Survived'], axis=1)
yTrain = train_data['Survived']
xTest  = test_data.drop( ['Survived'], axis=1)
```

特征相关性分析

- 该步骤用于筛选特征后向程序员反馈，特征是否有效、是否重叠
- 若有问题，可以修改之前的特征方案

```
# 特征间相关性分析
Correlation = pd.DataFrame(allData[allData.columns.to_list()])
colormap = plt.cm.viridis
plt.figure(figsize=(24,22))
sns.heatmap(Correlation.astype(float).corr(), linewidths=0.1, vmax=1.0,
cmap=colormap, linecolor='white', annot=True, square=True)
plt.show()
```



五、模型建立 & 参数优化

导入模型包

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest
```

> - 作者选择*随机森林分类器*

网格搜索调试参数

```
pipe = Pipeline([('select', SelectKBest(k=10)),
                 ('classify', RandomForestClassifier(random_state = 10,
max_features = 'sqrt'))])
param_test = {'classify__n_estimators':list(range(20,100,5)),
              'classify__max_depth'    :list(range(3,10,1))}
gsearch = GridSearchCV(estimator=pipe, param_grid=param_test, scoring='roc_auc',
cv=10)
gsearch.fit(xTrain, yTrain)
print (gsearch.best_params_, gsearch.best_score_)
```

- **运行时间较长**，结束后出现结果：

```
{'classify__max_depth': 6, 'classify__n_estimators': 70} 0.8790924679681529
```

建立模型

- 用以上参数进行输入模型
- 训练

```
rfc = RandomForestClassifier(n_estimators=70, max_depth=6, random_state=10,
max_features='sqrt')
rfc.fit(xTrain, yTrain)
```

导出结果

```
predictions = rfc.predict(xTest)
output = pd.DataFrame({'PassengerId':test['PassengerId'],
'Survived':predictions.astype('int64')})
output.to_csv('my_submission.csv', index=False)
```

六、提交评分

> - [官方推荐教程]

附：完整代码

> - Jupiter Notebook导出为Python Script格式，需要ipynb格式请点击
> - **[GitHub源代码]**

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'

# %%
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

# %% [markdown]
# # Features
# Variable | Definition | Key
# :-:|:-:|:-:
# survival | Survival | 0 = No, 1 = Yes
# pclass | Ticket class(客舱等级) | 1 = 1st, 2 = 2nd, 3 = 3rd
# sex | Sex
# Age | Age in years
# sibsp | # of siblings / spouses aboard the Titanic(旁系亲属)
# parch | # of parents / children aboard the Titanic(直系亲属)
# ticket | Ticket number
```

```
# fare | Passenger fare
# cabin | Cabin number(客舱编号)
# embarked | Port of Embarkation(上船的港口编号) | C = Cherbourg, Q = Queenstown,
S = Southampton

# %%
train = pd.read_csv("./kaggle/input/titanic/train.csv")
test = pd.read_csv("./kaggle/input/titanic/test.csv")
allData = pd.concat([train, test], ignore_index=True)
# dataNum = train.shape[0]
# featureNum = train.shape[1]
train.head()


# %%
# Sex
sns.countplot("Sex", hue="Survived", data=train)
plt.show()


# %%
# Pclass
sns.barplot(x="Pclass", y="Survived", data=train)
plt.show()
# Pclass & Age
sns.violinplot("Pclass", "Age", hue="Survived", data=train, split=True)
plt.show()


# %%
# FamilySize = SibSp + Parch + 1
allData["FamilySize"] = allData["SibSp"] + allData["Parch"] + 1
sns.barplot(x="FamilySize", y="Survived", data=allData)
plt.show()


# %%
# Embarked
sns.countplot("Embarked", hue="Survived", data=train)
plt.show()


# %%
# Age
sns.stripplot(x="Survived", y="Age", data=train, jitter=True)
plt.show()
facet = sns.FacetGrid(train, hue="Survived", aspect=2)
facet.map(sns.kdeplot, "Age", shade=True)
facet.set(xlim=(0, train["Age"].max()))
facet.add_legend()
plt.xlabel("Age")
plt.ylabel("density")
plt.show()
# Age & Sex
sns.swarmplot(x="Age", y="Sex", data=train, hue="Survived")
plt.show()
```

```python
# %%
# Fare
sns.stripplot(x="Survived", y="Fare", data=train, jitter=True)
plt.show()


# %%
# Name
# allData['Title'] = allData['Name'].str.extract('([A-Za-z]+)\.', expand=False)
# str.extract不知道在干嘛
allData["Title"] = allData["Name"].apply(
    lambda x: x.split(",")[1].split(".")[0].strip()
)
# pd.crosstab(allData['Title'], allData['Sex'])
TitleClassification = {
    "Officer": ["Capt", "Col", "Major", "Dr", "Rev"],
    "Royalty": ["Don", "Sir", "the Countess", "Dona", "Lady"],
    "Mrs": ["Mme", "Ms", "Mrs"],
    "Miss": ["Mlle", "Miss"],
    "Mr": ["Mr"],
    "Master": ["Master", "Jonkheer"],
}
TitleMap = {}
for title in TitleClassification.keys():
    TitleMap.update(dict.fromkeys(TitleClassification[title], title))
    """
    # cnt = 0
    for name in TitleClassification[title]:
        cnt += allData.groupby(['Title']).size()[name]
    # print (title,':',cnt)
    """
allData["Title"] = allData["Title"].map(TitleMap)
sns.barplot(x="Title", y="Survived", data=allData)
plt.show()


# %%
# Ticket
TicketCnt = allData.groupby(["Ticket"]).size()
allData["SameTicketNum"] = allData["Ticket"].apply(lambda x: TicketCnt[x])
sns.barplot(x="SameTicketNum", y="Survived", data=allData)
plt.show()
# allData['SameTicketNum']

# %% [markdown]
# # 数据清洗
# - Sex & Pclass & Embarked --> Ont-Hot
# - Age & Fare --> Standardize
# - FamilySize & Name & Ticket --> ints --> One-Hot

# %%
# Sex
allData = allData.join(pd.get_dummies(allData["Sex"], prefix="Sex"))
# Pclass
allData = allData.join(pd.get_dummies(allData["Pclass"], prefix="Pclass"))
# Embarked
allData[allData["Embarked"].isnull()]  # 查看缺失值
```

```python
allData.groupby(by=["Pclass", "Embarked"]).Fare.mean()  # Pclass=1, Embark=C, 中位数=76
allData["Embarked"] = allData["Embarked"].fillna("C")
allData = allData.join(pd.get_dummies(allData["Embarked"], prefix="Embarked"))


# %%
# Age
allData["Child"] = allData["Age"].apply(lambda x: 1 if x <= 10 else 0)  # 儿童标签
allData["Age"] = (allData["Age"] - allData["Age"].mean()) / allData["Age"].std()  # 标准化
allData["Age"].fillna(value=0, inplace=True)  # 填充缺失值
# Fare
allData["Fare"] = allData["Fare"].fillna(25)  # 填充缺失值
allData[allData["Survived"].notnull()]["Fare"] = allData[allData["Survived"].notnull()][
    "Fare"
].apply(lambda x: 300.0 if x > 500 else x)
allData["Fare"] = allData["Fare"].apply(
    lambda x: (x - allData["Fare"].mean()) / allData["Fare"].std()
)


# %%
# FamilySize
def FamilyLabel(s):
    if s == 4:
        return 4
    elif s == 2 or s == 3:
        return 3
    elif s == 1 or s == 7:
        return 2
    elif s == 5 or s == 6:
        return 1
    elif s < 1 or s > 7:
        return 0


allData["FamilyLabel"] = allData["FamilySize"].apply(FamilyLabel)
allData = allData.join(pd.get_dummies(allData["FamilyLabel"], prefix="Fam"))

# Name
TitleLabelMap = {
    "Mr": 1.0,
    "Mrs": 5.0,
    "Miss": 4.5,
    "Master": 2.5,
    "Royalty": 3.5,
    "Officer": 2.0,
}


def TitleLabel(s):
    return TitleLabelMap[s]


# allData['TitleLabel'] = allData['Title'].apply(TitleLabel)
```

```python
allData = allData.join(pd.get_dummies(allData["Title"], prefix="Title"))

# Ticket
def TicketLabel(s):
    if s == 3 or s == 4:
        return 3
    elif s == 2 or s == 8:
        return 2
    elif s == 1 or s == 5 or s == 6 or s == 7:
        return 1
    elif s < 1 or s > 8:
        return 0


allData["TicketLabel"] = allData["SameTicketNum"].apply(TicketLabel)
allData = allData.join(pd.get_dummies(allData["TicketLabel"], prefix="TicNum"))


# %%
# 清除无用特征
allData.drop(
    [
        "Cabin",
        "PassengerId",
        "Ticket",
        "Name",
        "Title",
        "Sex",
        "SibSp",
        "Parch",
        "FamilySize",
        "Embarked",
        "Pclass",
        "Title",
        "FamilyLabel",
        "SameTicketNum",
        "TicketLabel",
    ],
    axis=1,
    inplace=True,
)

# 重新分割数据集
train_data = allData[allData["Survived"].notnull()]
test_data = allData[allData["Survived"].isnull()]
test_data = test_data.reset_index(drop=True)

xTrain = train_data.drop(["Survived"], axis=1)
yTrain = train_data["Survived"]
xTest = test_data.drop(["Survived"], axis=1)

# allData.columns.to_list()


# %%
# 特征间相关性分析
Correlation = pd.DataFrame(allData[allData.columns.to_list()])
colormap = plt.cm.viridis
```

```python
plt.figure(figsize=(24, 22))
sns.heatmap(
    Correlation.astype(float).corr(),
    linewidths=0.1,
    vmax=1.0,
    cmap=colormap,
    linecolor="white",
    annot=True,
    square=True,
)
plt.show()

# %% [markdown]
# # 网格筛选随机森林参数
# - n_estimator
# - max_depth

# %%
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest


# %%

pipe = Pipeline(
    [
        ("select", SelectKBest(k=10)),
        ("classify", RandomForestClassifier(random_state=10,
max_features="sqrt")),
    ]
)
param_test = {
    "classify__n_estimators": list(range(20, 100, 5)),
    "classify__max_depth": list(range(3, 10, 1)),
}
gsearch = GridSearchCV(estimator=pipe, param_grid=param_test, scoring="roc_auc",
cv=10)
gsearch.fit(xTrain, yTrain)
print(gsearch.best_params_, gsearch.best_score_)


# %%
rfc = RandomForestClassifier(
    n_estimators=70, max_depth=6, random_state=10, max_features="sqrt"
)
rfc.fit(xTrain, yTrain)
predictions = rfc.predict(xTest)

output = pd.DataFrame(
    {"PassengerId": test["PassengerId"], "Survived":
predictions.astype("int64")}
)
output.to_csv("my_submission.csv", index=False)
```

> **链接：[GitHub源代码]()**