- [28]：import packages

  [29]：將 kaggle 提供的訓練集(train)和測試集(test)讀入，並且將兩者合併為 data 集合。

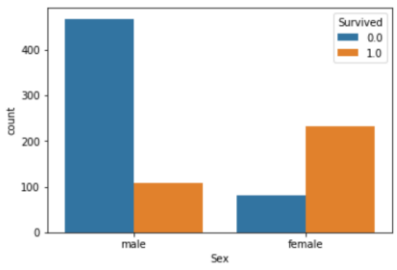
In [28]:
```
# package
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import RFECV
from sklearn.model_selection import cross_val_score, StratifiedKFold, learning_curve, train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
# import importlib; importlib.reload(pd)
```

In [29]:
```
# Load data
train = pd.read_csv("./data/train.csv")
test = pd.read_csv("./data/test.csv")
gender = pd.read_csv("./data/gender_submission.csv")
# train.info()
# 共891個index，沒有滿的就是空值
# train空值：Age, Cabin, Embarked
# test.info()
# 418
# test空值：Age, Fare, Cabin
data = train.append(test)
```

- [30]：對 data 中的性別(Sex)進行分析，繪製不同性別生存人數的直方圖，並在表格中列出不同性別生存的比例(survived_sex)。由圖中可以看出女性的生存人數約為男性的兩倍，男性的死亡人數遠高於女性。由表格中可看出女性的生存比例 0.742 遠大於男性的生存比例 0.189，因此在預測模型中，性別應該是辨別生存與否的一個顯著特徵(女性傾向辨別生存)。

In [30]:
```
#查看資料_性別
sns.countplot(data['Sex'], hue=data['Survived'])
# plt.show()
# 生存人數:女>男

survived_sex = data[['Sex','Survived']].groupby(['Sex'], as_index=True).mean().round(3)
display(survived_sex)
# 生存比例：女=>0.742，男=>0.189
```

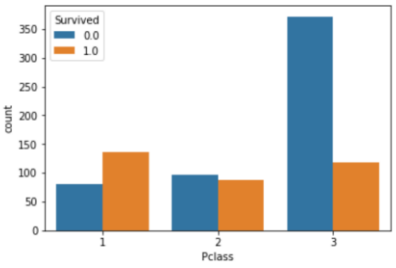
Sex	Survived
female	0.742
male	0.189



- [31]：對 data 中的艙等(Pclass)進行分析，繪製不同艙等乘客生存人數的直方圖，並在表格中列出不同艙等乘客的生存比例(survived_pclass)。由圖中可以看出艙等為 1(較高等級)的乘客生存人數較死亡人數多，艙等為 3(較低等級)的乘客死亡人數遠多於生存人數，也高於其他艙等的死亡人數。由表格中可以看出，艙等為 1(較高等級)的生存比例 0.630>艙等為 2 的生存比例 0.473>艙等為 3 的生存比例 0.242，艙等等級越高生存機率也越高，且區別顯著，在模型預測中應傾向將高艙等乘客分類為生存。

```
In [31]: #查看資料_艙等
sns.countplot(data['Pclass'], hue=data['Survived'])
# plt.show()
# 生存人數:1>2>3

survived_pclass = data[['Pclass','Survived']].groupby(['Pclass'], as_index=True).mean().round(3)
display(survived_pclass)
# 生存比例：1=>0.63，2=>0.473，3=>0.242
```

Pclass	Survived
1	0.630
2	0.473
3	0.242



- [32]：將性別(female/male)轉為編碼(1/0)存為 data[‘Sex_code’]。

  [33]：x_train 為訓練集，x_test 為測試集，y 為訓練集的答案(乘客是否存活)，是模型要去 fit 的結果。

```
In [32]: # 把性別變成0/1編碼
data['Sex_code'] = data['Sex'].map({'female':int(1), 'male':int(0)})
train['Sex_code'] = train['Sex'].map({'female':int(1), 'male':int(0)})
test['Sex_code'] = test['Sex'].map({'female':int(1), 'male':int(0)})
# print(train.head())
data.columns

Out[32]: Index(['Age', 'Cabin', 'Embarked', 'Fare', 'Name', 'Parch', 'PassengerId',
       'Pclass', 'Sex', 'SibSp', 'Survived', 'Ticket', 'Sex_code'],
      dtype='object')

In [33]: # x = train.drop(labels=['Survived','PassengerId'],axis=1)
x_train = train
y = train['Survived']
x_test = test
```

- [34]：CASE 1 中使用性別做為唯一的區分特徵，先建立一個基本的 base_model 查看模型的預測效能，之後再逐漸疊加特徵，看其他特徵能不能提高預測準確率，如果不行就是有太多 noise 或冗餘特徵影響了預測，需要拿掉該項特徵。在 base_model 中使用隨機森林進行分類，用 x_train 中的性別編碼去 fit 訓練集的答案 y，並印出 oob score 作為模型預測準確率的參考，ans_1 則是將 x_test 丟入模型中預測出的答案。將答案存成 csv 並上傳 kaggle，可以得到預測準確率為 0.76555。

In [34]:
```
# CASE 1

# 特徵使用：性別
base = ['Sex_code']
# random_state：建樹資料的隨機性, n_estimators：樹數量, min_samples_split：節點資料數
base_model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
base_model.fit(x_train[base],y)
print('base oob score = {}'.format(base_model.oob_score_))
ans_1 = base_model.predict(x_test[base])
# display(ans_1)

# print(len(test.PassengerId))
# print(len(ans_1))
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_1}).set_index('PassengerId').to_csv('./result/sexcode.csv')
```

```
base oob score = 0.7867564534231201
```

- [35]：CASE 2 中加入艙等做為區分特徵，base_model 改用 x_train 中性別編碼與艙等編號兩種特徵去 fit 答案 y，得到 oob score 約 0.7318，比起只用性別作為區分特徵的 oob score 略低一些，但實際在 kaggle 網站上的預測準確度一樣是 0.76555，檢查預測結果 ans_1 和 ans_2，兩者預測出來的答案其實一樣，沒有不同，所以在使用性別區分後，再使用艙等區分其實沒辦法提供足夠多不同的資訊，讓預測結果更準確。

In [35]:
```
# CASE 2

# 特徵使用：性別、艙等
base1 = ['Sex_code','Pclass']
base_model.fit(x_train[base1],y)
print('base oob score = {}'.format(base_model.oob_score_))
ans_2 = base_model.predict(x_test[base1])

if ans_1.all() == ans_2.all():
    print('預測結果 CASE1 = CASE2')

pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_1}).set_index('PassengerId').to_csv('./result/sexcode_pclass.csv')
```

```
base oob score = 0.7317620650953984
預測結果 CASE1 = CASE2
```

sexcode_pclass.csv 0.76555

5 days ago by 林子淩(Lin Tzu Ling0712118)

sex + pclass

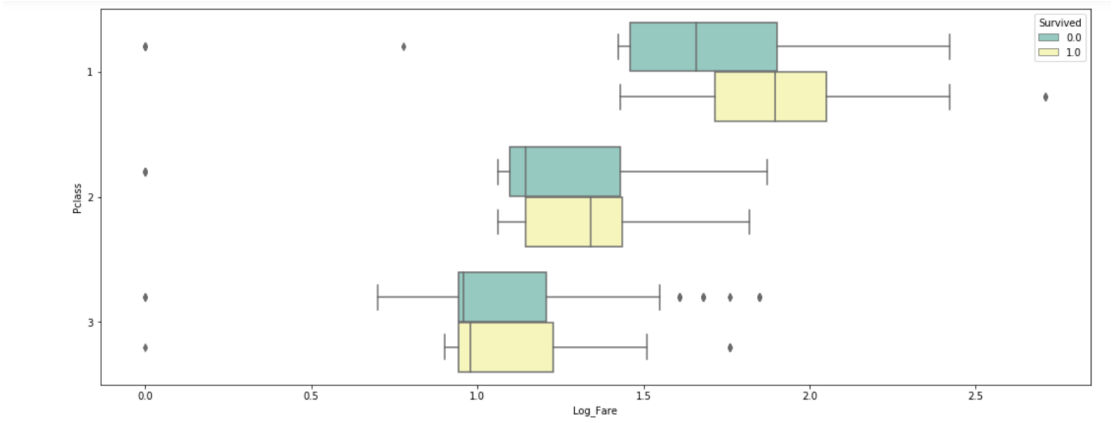
- [36]： 對 data 中的票價(Fare)進行分析，可以看到票價的區間差異很大，max 和 min 相差很多，標準差大，且平均比 max 小很多可以看出有極端值存在。因此先將票價同取 Log 降低這種影響，並繪製票價與艙等的盒方圖，圖中可以看出票價高與艙等等級高存在正向關係，而生存的乘客也比死亡的乘客付出較高的票價。在表格中列出不同艙等，生存和死亡的乘客中，付出票價的中位數，可以看出生存的乘客普遍付出較高票價，推測當票價較高時，該名乘客的生還率也會更高。

In [36]:
```
#查看資料_票價
print('max fare = {}'.format(data['Fare'].max()))
print('min fare = {}'.format(data['Fare'].min()))
print('mean fare = {}'.format(data['Fare'].mean()))
print('std fare = {}'.format(data['Fare'].std()))
print('median fare = {}'.format(data['Fare'].median()))


# sns.countplot(data['Fare'], hue=data['Survived'])
survived_fare = data[['Fare','Survived']].groupby(['Fare'], as_index=True).mean().round(3)
# display(survived_fare.head())

# 票價差異很大，取Log解決
fig, ax = plt.subplots(figsize=(18,7))
data['Log_Fare'] = (data['Fare']+1).map(lambda x: np.log10(x))
# data['Log_Fare'].min()
sns.boxplot(y='Pclass', x='Log_Fare', hue='Survived', data=data, orient='h', ax=ax, palette='Set3' )
pd.pivot_table(data, values=['Fare'], index=['Pclass'], columns=['Survived'], aggfunc='median').round(3)
```

```
max fare = 512.3292
min fare = 0.0
mean fare = 33.2954792813456
std fare = 51.75866823917414
median fare = 14.4542
```

Out[36]:

	Fare	
Survived	0.0	1.0
Pclass		
1	44.75	77.958
2	13.00	21.000
3	8.05	8.517



- [37]：處理 data 中票價缺失的資訊，將缺失的值以中位數補上。切分票價區間，但切分區間太少可能會因為區間內資訊平均後，沒有辦法看出資訊間較細部的差異。切分區間太多又可能放大資訊間的差異，受到極端/非正常值的影響而導致 overfitting。所以在這邊做了將票價切分為 4、5、6 個區間的三種實驗，以 pd.qcut 以分位數切分票價，並使用 labelEncoder 標記每筆資料分在哪個區間。使用 crosstab 交叉表格找出不同票價區間與艙等間的關係。

In [37]:
```
# Fare資料處理
data['Fare'] = data['Fare'].fillna(data['Fare'].median())

# 切分票價區間
# 區間太少：區間資料平均後無法看出差異
# 區間太多：資料差異放大，造成overfitting
# qcut：分位數
data['Fare_bin_4'] = pd.qcut(data['Fare'],4)
data['Fare_bin_5'] = pd.qcut(data['Fare'],5)
data['Fare_bin_6'] = pd.qcut(data['Fare'],6)
# display(data['Fare_bin_4'])
# display(data['Fare_bin_5'])

# LabelEncoder：把每個類別 mapping 到某個整數，不會增加新欄位
label = LabelEncoder()
data['Fare_Code_4'] = label.fit_transform(data['Fare_bin_4'])
data['Fare_Code_5'] = label.fit_transform(data['Fare_bin_5'])
data['Fare_Code_6'] = label.fit_transform(data['Fare_bin_6'])

# cross tab
df_4 = pd.crosstab(data['Fare_bin_4'], data['Pclass'])
df_5 = pd.crosstab(data['Fare_bin_5'], data['Pclass'])
df_6 = pd.crosstab(data['Fare_bin_6'], data['Pclass'])
```

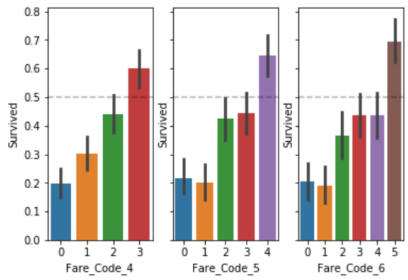
```
from IPython.display import display_html
from itertools import chain,cycle
def display_side_by_side(*args,titles=cycle(['']):
    html_str=''
    for df,title in zip(args, chain(titles,cycle(['</br>'])) ):
        html_str+='<th style="text-align:center"><td style="vertical-align:top">'
        html_str+=f'<h2>{title}</h2>'
        html_str+=df.to_html().replace('table','table style="display:inline"')
        html_str+='</td></th>'
    display_html(html_str,raw=True)
display_side_by_side(df_4,df_5,df_6)

# plot
fig, [ax1,ax2,ax3] = plt.subplots(1, 3, sharey=True)
# fig.set_figwidth(18)
for axi in [ax1,ax2,ax3]:
    axi.axhline(0.5,linestyle='dashed',c='black',alpha=0.3)
g1 = sns.factorplot(x='Fare_Code_4', y='Survived', data=data, kind='bar', ax=ax1)
g2 = sns.factorplot(x='Fare_Code_5', y='Survived', data=data, kind='bar', ax=ax2)
g3 = sns.factorplot(x='Fare_Code_6', y='Survived', data=data, kind='bar', ax=ax3)
plt.close(g1.fig)
plt.close(g2.fig)
plt.close(g3.fig)
```

- [37]：表格中可以看出票價越高的乘客越多在高等艙。圖則呈現不同區間票價與該乘客生存機率的關係，可以看到大致走向皆為票價越高，生存機率越高，但區分區間較多時，可以看出較為細部的差異(在低票價區間中，只高一點點的票價不會有更高的生存率)。

Pclass	1	2	3
Fare_bin_4			
(-0.001, 7.896]	8	6	323
(7.896, 14.454]	0	128	193
(14.454, 31.275]	77	104	147
(31.275, 512.329]	238	39	46

Pclass	1	2	3
Fare_bin_5			
(-0.001, 7.854]	8	6	261
(7.854, 10.5]	0	36	218
(10.5, 21.558]	0	124	132
(21.558, 41.579]	95	99	71
(41.579, 512.329]	220	12	27

Pclass	1	2	3
Fare_bin_6			
(-0.001, 7.775]	8	6	222
(7.775, 8.662]	0	0	218
(8.662, 14.454]	0	128	76
(14.454, 26.0]	14	83	128
(26.0, 53.1]	118	48	46
(53.1, 512.329]	183	12	19



- [38]：同樣將 data 切分回訓練集 x_train 跟測試集 x_test，以及訓練集答案 y，並確定 x_train 中有剛剛切分出的新特徵(Fare_Code_4、Fare_Code_5、Fare_Code_6)。

In [38]:
```
x_train = data[:len(train)]
y = x_train['Survived']
x_test = data[len(train):]
x_train = x_train.drop(labels=['Survived','PassengerId'],axis=1)
x_train.columns

# x_train['Sex_Code'] = x_train['Sex'].map({'female':int(1), 'male':int(0)})
# x_test['Sex_Code'] = x_test['Sex'].map({'female':int(1), 'male':int(0)})

# display(x_train.columns)
# display(x_test.columns)
# y = train.Survived
# display(y.head())
```

Out[38]:
```
Index(['Age', 'Cabin', 'Embarked', 'Fare', 'Name', 'Parch', 'Pclass', 'Sex',
       'SibSp', 'Ticket', 'Sex_code', 'Log_Fare', 'Fare_bin_4', 'Fare_bin_5',
       'Fare_bin_6', 'Fare_Code_4', 'Fare_Code_5', 'Fare_Code_6'],
      dtype='object')
```

- [39]：使用 RFECV 套件，交叉驗證以得出使用不同特徵組合會得到多高的準確度預測，在 grid_scores_結果中可以看到，使用票價分區特徵會對預測準確率有所提升，其中使用六個區間的票價切分效果最好，但在這裡沒有考慮到 selector 的隨機狀態和不同 validation 集合的切分造成的影響。

In [39]:
```
# 選擇切分特徵的區間數
compare = ['Sex_code', 'Pclass', 'Fare_Code_4', 'Fare_Code_5', 'Fare_Code_6']
# RFECV使用交叉驗證來選擇有最好準確率的訓練特徵數目。而交叉驗證也可以幫助我們避免訓練時造成過度訓練(overfitting)的現象
selector = RFECV(RandomForestClassifier(n_estimators=250, min_samples_split=20,), cv=10, n_jobs=-1)
selector.fit(x_train[compare], y)
# support：是否保留特徵，ranking：特徵重要程度排名
print(selector.support_)
print(selector.ranking_)
print(selector.grid_scores_)

# 6區間的準確率最好，但這裡沒有考慮到模型的random_state以及Cross-Validation切分的方式
```

```
[ True  True  True  True  True]
[1 1 1 1 1]
[0.78669816 0.77333986 0.79347747 0.79355295 0.80143088]
```

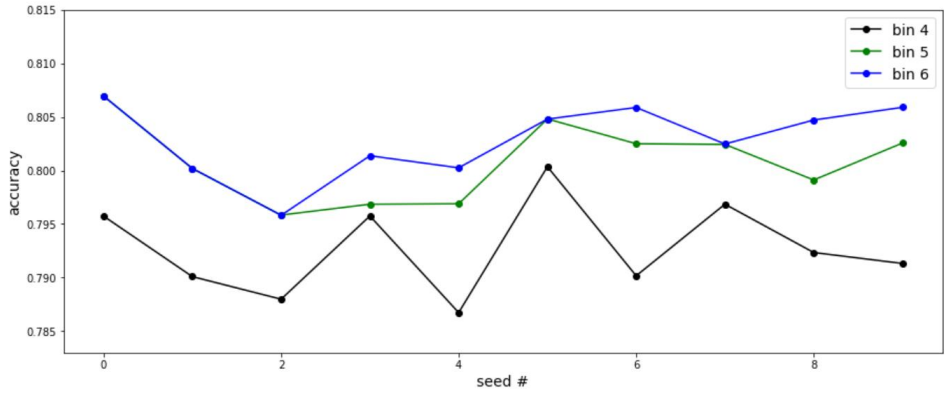
- [40]：在[39]中提到要考慮 selector 每次隨機出來的狀態和訓練資料堆切分對實驗的影響。所以在這邊進行 CV 隨機實驗，用 seeds 控制十次實驗中 selector 的隨機狀態。diff_cv 在每次實驗中將資料分成 10 堆並隨機抽 9 堆用來訓練模型，剩下一堆作為預測使用，重複十次讓每個組合都能被考慮到。並用 score_b4/score_b5/score_b6 儲存十次實驗產生的 score。

In [40]:
```
# 進行CV實驗

score_b4, score_b5, score_b6 = [], [], []
seeds=10
for i in range(seeds):

    # StratifiedKFold：K堆疊(Fold)的交叉驗證，將資料分為 K堆，一堆作為預測用，剩下的(K-1)堆則用來訓練，
    # 經過計算後，再以另外一堆作為預測，重複K次。
    diff_cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=i)

    # cv: 若無輸入，預設為3-fold的交叉驗證。輸入整數k，則做k-fold交叉驗證。若為物件，則以該物件做為交叉驗證產生器。
    selector = RFECV(RandomForestClassifier(random_state=i, n_estimators=250, min_samples_split=20,), cv=diff_cv, n_jobs=-1)
    selector.fit(x_train[compare], y)

    # gridscores: 從最有影響力的特徵開始加入，計算使用多少個特徵對應得到的準確率。
    score_b4.append(selector.grid_scores_[2])
    score_b5.append(selector.grid_scores_[3])
    score_b6.append(selector.grid_scores_[4])
```

- [41]：以[40]中產生的 score 繪製折線圖，不同條折線代表不同的票價區間切分數，每個點則是不同次隨機實驗中得出的 grid_score。由圖中可以看出將票價區間切分為 5 或 6 個區間的預測效果會大於切分為 4 個區間的預測效果，而切分為 6 個區間的預測效果在十次的隨機實驗中效果均高於 5 個區間，這邊推測使用票價 6 個區間作為特徵會對模型預測更有利。

```
In [41]: # np.array
score_list = [score_b4, score_b5, score_b6]
for item in score_list:
    item = np.array(item*100)

# print(len(score_b4))
# print(len(score_b5))


# 作圖看實驗結果
fig = plt.figure(figsize=(18,8))
ax = plt.gca()
ax.plot(range(seeds), score_b4, '-ok', label='bin 4')
ax.plot(range(seeds), score_b5, '-og', label='bin 5')
ax.plot(range(seeds), score_b6, '-ob', label='bin 6')

# display(ax)
ax.set_xlabel('seed #', fontsize='14')
ax.set_ylabel('accuracy', fontsize='14')
ax.set_ylim(0.783,0.815)
plt.legend(fontsize=14,loc='upper right')
```

```
Out[55]: <matplotlib.legend.Legend at 0x26f472b5438>
```

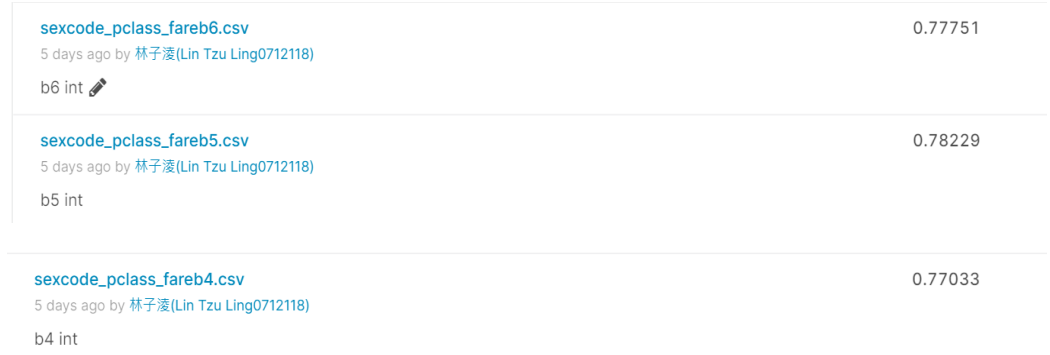


- [42]：實際使用性別、艙等、票價區間，作為訓練模型的特徵。並考量不同票價區間切分數(切分成 4/5/6 區間)，設置三個不同的模型，以前面說過的方法同樣用隨機森林搭建模型，可以看到實際搭建模型的 oob score 結果中，反而是將票價切分為 5 個區間的分數最高。上傳 kaggle 後同樣可以看到使用 5 區間切分票價的特徵預測準確度最高。

In [42]:
```
# b6 > b5 > b4 (原使用性別、艙等的模型，另外再加入票價特徵)
b4 = ['Sex_code', 'Pclass', 'Fare_Code_4']
b5 = ['Sex_code', 'Pclass', 'Fare_Code_5']
b6 = ['Sex_code', 'Pclass', 'Fare_Code_6']

b4_Model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
b4_Model.fit(x_train[b4], y)
ans_b4 = b4_Model.predict(x_test[b4])
print('b4 oob score = {}'.format(b4_Model.oob_score_.round(4)))

b5_Model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
b5_Model.fit(x_train[b5], y)
ans_b5 = b5_Model.predict(x_test[b5])
print('b5 oob score = {}'.format(b5_Model.oob_score_.round(4)))

b6_Model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
b6_Model.fit(x_train[b6], y)
ans_b6 = b6_Model.predict(x_test[b6])
print('b6 oob score = {}'.format(b6_Model.oob_score_.round(4)))
```

```
b4 oob score = 0.8058
b5 oob score = 0.8103
b6 oob score = 0.8013
```

In [43]:
```
# display(ans_1)
# display(f4)
# display(ans_b4)
# display(type(ans_b4))
# display(type(ans_1))
# display(type(ans_b4))
# a1 = pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_1}).set_index('PassengerId')
# display(a1)

# 記得輸出要是整數(不然submit score=0)
ans_b4 = ans_b4.astype(int)
ans_b5 = ans_b5.astype(int)
ans_b6 = ans_b6.astype(int)

pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_b4}).set_index('PassengerId').to_csv('./result/sexcode_pclass_fareb
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_b5}).set_index('PassengerId').to_csv('./result/sexcode_pclass_fareb
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_b6}).set_index('PassengerId').to_csv('./result/sexcode_pclass_fareb
```



- [44]：分析 data 中的票號資訊，因為推測票號相同的乘客可能是一起同行的家人或朋友(團體)，在逃生的過程中如果有認識的人互相幫助，生存的機率可能會增加。因此在這裡對 data 中的票號(Ticket)進行分析，並將 data 提供每名乘客的同行手足人數(SibSp)和同行親子人數(Parch)相加，作為新特徵 Family_size。若票號相同且 Family_size>1，代表同行除了自己以外還有其他家人，為家庭團體。若票號相同但 Family_size=1，代表同行中沒有家人，但票號相同可能是一起買的票，推測為彼此互相熟悉的

朋友團體。在 for 迴圈中，找出票號相同(皆為 tk_no)的資料 tmp，如果 tmp 的數量>1，代表找到票號相同的團體，並存入 deplicate_ticket。從輸出結果可以看到，具有相同票號的團體共有 596 個，其中 127 個為家庭團體，469 個為朋友團體。

In [44]:
```
# 票號(相同船票：家人(familysize>1)或朋友(familysize=1)，可能一起活或爽命)
x_train['Ticket'].describe()

# family size = sib + parch
data['Family_size'] = data['SibSp'] + data['Parch'] + 1

deplicate_ticket = []
for tk_no in data.Ticket.unique():
    # 票號和票價都相同
    tmp = data.loc[data.Ticket==tk_no, 'Fare']
    if tmp.count() > 1:
        deplicate_ticket.append(data.loc[data.Ticket==tk_no,['Name','Age','Ticket', 'Fare', 'Cabin', 'Family_size', 'Survived']])
# display(pd.concat(deplicate_ticket))
deplicate_ticket = pd.concat(deplicate_ticket)
deplicate_ticket.head(10)

print(len(deplicate_ticket))
# family
print(len(deplicate_ticket[deplicate_ticket.Family_size==1]))
# friend
print(len(deplicate_ticket[deplicate_ticket.Family_size>1]))
```

```
596
127
469
```

- [45]：分離出朋友團體(friend)和家庭團體(family)中具有生存與否(Survived)的資料，分別為 89 筆和 321 筆，代表在資料中需要特別注意補空值的問題(Survived 是否 nan)。

In [45]:
```
friend = deplicate_ticket.loc[(deplicate_ticket.Family_size==1) & (deplicate_ticket.Survived.notnull())]
family = deplicate_ticket.loc[(deplicate_ticket.Family_size>1) & (deplicate_ticket.Survived.notnull())]
# display(friend)
# display(family)
print('friends = {}'.format(len(friend)))
print('family = {}'.format(len(family)))
```

```
friends = 89
family = 321
```

- [46]：建立新特徵 data[‘connect_survived’]，先統一將值設為 0.5(解決同行人 Survived=nan 無法判斷的問題)。在第一層 for 迴圈中，首先判斷每一個票號對應的乘客數量是否>1(為團體)，第二層 for 迴圈中，則在該群體中分別拉出每一個人，判斷對於該乘客，其他同行者是否有人生存。若有同行者生存則 fill_max=1，與此同時令該筆資料(該乘客)的

connect_survived=1。若無同行者生存則 fill_max=0，並令該筆資料(該乘客)的 connect_survived=0。若其他同行者的生存資料皆缺失(Survived=nan)，則讓該乘客的 connect_survived 維持 0.5。

In [46]:
```
# 建立新特徵(朋友和家人活下來的數量)
data['connect_survived'] = 0.5
# 同樣票號群組中，有人生還為1，無人生還為0
data_groupbytk = data.groupby('Ticket')
# data_groupbytk.agg({'Survived':'max'}).fillna(0.5)

# iterate over each group
for group_idx, group_data in data_groupbytk:
    # 有朋友or家人
    if len(group_data) > 1:
        for row_idx, row in group_data.iterrows():
            # 除了自己以外的團體成員
            others = group_data.drop(row_idx)
            fill_max = others['Survived'].max()
            #  fill_min = others['Survived'].min()
            passid = row['PassengerId']
            #  print('{} , {}'.format(fill_max, fill_min))
            #  print(row_idx)
            if fill_max == 1.0:
                data.loc[data['PassengerId']==passid, 'connect_survived'] = 1
                # data.loc[row_idx, 'connect_survived'] = 1
            elif fill_max == 0.0:
                data.loc[data['PassengerId']==passid, 'connect_survived'] = 0
                # data.loc[row_idx, 'connect_survived'] = 0
            # 如果nan算不出fill_max => 維持原本的0.5
data['connect_survived'].describe()
```

Out[46]:
```
count    1309.000000
mean        0.535141
std         0.305885
min         0.000000
25%         0.500000
50%         0.500000
75%         0.500000
max         1.000000
Name: connect_survived, dtype: float64
```

- [47]：檢查在 596 筆群體成員資料中，有多少人的 connect_survived 資料不為 0.5(同行人有人生還或同行人全部罹難)，由輸出結果可以看出有 496 筆資料的 connect_survived 為 0/1，可以比較顯著的區分不同。另外，查看以不同 connect_survived 狀態(0/0.5/1)去區分 data 時，不同分群的 Survived 平均機率。可以看出當乘客的同行團體中有其他成員生還(connect_survived=1)時，該乘客的生存機會會比較大(0.727223)。

In [47]:
```
print(len(deplicate_ticket))
print(data[data['connect_survived']!=0.5].shape[0])
# 有同行人生還，自己生還的可能性也比較大
data.groupby('connect_survived')['Survived'].mean()
```

```
596
496
```

Out[47]:
```
connect_survived
0.0    0.225352
0.5    0.297989
1.0    0.727723
Name: Survived, dtype: float64
```

- [49]：採用性別編碼、艙等、票價(分為五區間)和同行者是否生還(connect_survived)作為特徵，搭建隨機森林模型 connect_model。由輸出可以看到加入 connect_survived 特徵後，oob_score 從 0.8103 上升到 0.8204。上傳至 kaggle 後，預測準確度也從 0.78229 上升到 0.79665。

In [49]:
```
x_train = data[:len(train)]
x_test = data[len(train):]
y = x_train['Survived']
x_train = x_train.drop(labels=['Survived','PassengerId'],axis=1)
# x_train.columns

connect = ['Sex_code', 'Pclass', 'Fare_Code_5', 'connect_survived']

connect_model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
connect_model.fit(x_train[connect], y)
ans_connect = connect_model.predict(x_test[connect])
print('connect oob score = {}'.format(connect_model.oob_score_.round(4)))

ans_connect = ans_connect.astype(int)
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_connect}).set_index('PassengerId').to_csv('./result/sexcode_pclass_
```

```
connect oob score = 0.8204
```

sexcode_pclass_fareb5_connect.csv 0.79665

4 days ago by 林子淩(Lin Tzu Ling0712118)

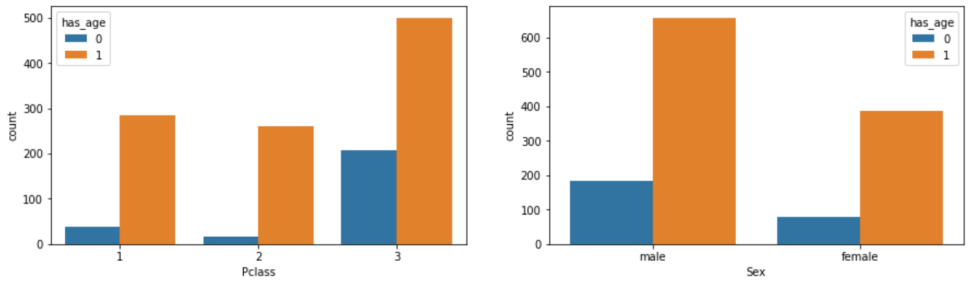
re

- [50]：查看 data 中的年齡(Age)資訊，首先按照性別和艙等去劃分，觀察不同性別和艙等的乘客中有多少人沒有年齡資訊(Age=nan)。在性別劃分中，男性缺失年齡的人數較多，若能補充此特徵，可以更好的預測一位男性乘客是否存活(推測男性小孩的存活率可能大於男性成人)。在以艙等劃分中，可以看到艙等=3 的乘客缺失年齡資訊的資料筆數多很多，因此在使用年齡作為特徵預測時，對艙等=3 的乘客預測結果可能會有較多偏誤。

In [50]:
```
# 查看年齡資訊
data['has_age'] = data['Age'].isnull().map(lambda x: 0 if x==True  else 1 )
# display(data['has_age'])
fig, [ax1,ax2] = plt.subplots(1,2)
fig.set_figwidth(15)
ax1 = sns.countplot(x=data['Pclass'], hue=data['has_age'],ax=ax1)
ax2 = sns.countplot(x=data['Sex'], hue=data['has_age'], ax=ax2)
pd.crosstab(data['has_age'], data['Sex'], margins=True)

# 第三艙沒有年齡數量多，可能數據預測失真
# 男性沒有年齡數量多，加入年齡可能預測更準(ex.男性小孩存活>男性成人存活)
```

Out[50]:

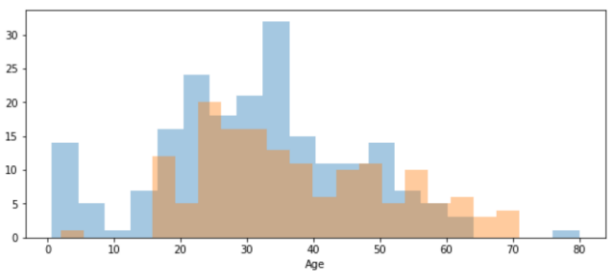
Sex	female	male	All
has_age			
0	78	185	263
1	388	658	1046
All	466	843	1309



- [51]：因為第三艙等缺失年齡的人數較多，觀察第三艙存活乘客的年齡對預測效果可能不會很好，因此先針對第一、二艙具有年齡資訊且生還的乘客進行觀察。在繪製的圖形中，藍色為生存乘客的年齡，紅色為死亡乘客的年齡，可以看到藍色在 16 歲之前的人數比紅色多，紅色 16 歲之前的人數分布將近為 0，因此推測 16 歲以下的乘客存活率較高。

In [51]:
```
# 看存活和年齡間的關係
# 生存且非第三艙
hue_survived = data.loc[ ((data['Pclass']!=3) & (data['has_age']==1) & (data['Survived']==1)),'Age' ]
hue_dead = data.loc[ ((data['Pclass']!=3) & (data['has_age']==1) & (data['Survived']==0)),'Age' ]

# display(hue_survived)
fig, ax1 = plt.subplots(1,1)
fig.set_figwidth(10)
sns.distplot(hue_survived,kde=False,bins=20,ax=ax1)
sns.distplot(hue_dead,kde=False,bins=20,ax=ax1)

# 16歲以下年齡影響生存機率很大
```

Out[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f47132860>
```



- [52]：要使用年齡作為分類特徵前，需要先補滿缺失的年齡資訊空值。從 data 中的 Name 可以看出每名乘客的頭銜(例如：Mr., Mrs., Miss. , Master.等)。這些頭銜與年齡有相對關係，可以看出該名乘客的大致年齡，因此在這邊將頭銜(title)從名字(Name)中提取出來，將比較不常見的頭銜(例如：Capt., Col.等)或通用的頭銜(例如：Mlle., Ms., Mme.與 Miss.、Lady.與 Mrs.)經過整理合併，轉換成 0 到 4 的編碼後，根據每種頭銜群組找出該頭銜的年齡中位數，作為同樣有該頭銜但年齡資訊為空的空值補充數值。

In [52]:
```
data['title'] = data['Name'].str.extract(pat = '([A-Z][a-z]+)\.',expand=False)

title1 = ['Capt','Col','Countess','Don','Dr','Dona','Jonkheer','Major','Rev','Sir']
title2 = ['Mlle','Ms','Mme']
title3 = ['Lady']

data['title'] = data['title'].replace(title1,'Rare')
data['title'] = data['title'].replace(title2,'Miss')
data['title'] = data['title'].replace(title3,'Mrs')

# display(data['Title'])

title_dict = dict()
title_dict = {'Mr':0,'Mrs':1,'Miss':2,'Master':3,'Rare':4}
data['title_code'] = 0


for i in range(len(data['title'])):
    data['title_code'].iloc[i] = title_dict[data['title'].iloc[i]]

data.groupby('title')['Age'].median()

# display(data.title_code.head(20))
# display(data['Title'].unique())
```

Out[52]:
```
title
Master     4.0
Miss      22.0
Mr        29.0
Mrs       36.0
Rare      47.0
Name: Age, dtype: float64
```

- [53]：根據上面算出的頭銜與該頭銜的年齡中位數，建立一個名為 age_fill 的 dict，用來儲存每個頭銜對應的年齡中位數。同時建立特徵 title_age，跟 Age 一樣儲存每位乘客的年齡，但另外使用 for 迴圈，找出年齡空值並根據頭銜補上年齡中位數，使得 title_age 中不具有空值存在。最後設立特徵 young，以 0/1 記錄每位乘客是否年齡<16 歲，作為分類的依據。

In [53]:
```
# 建立頭銜對應年齡中位數的dict
age_fill = dict()
key = list(data.groupby('title')['Age'].groups.keys())
value = data.groupby('title')['Age'].median().values
for i in range(len(key)):
    age_fill[key[i]] = value[i]
# display(age_fill)

# 保留原本的年齡特徵不動
data['title_age'] = data['Age']

for i in range(len(data)):
    if pd.isna(data.iloc[i]['Age']):
        title = data.iloc[i]['title']
        data['title_age'].iloc[i] = age_fill[title]


data['title_age'].describe()
# display(data['title_age'].head(20))
data['title_age'] = data['title_age'].astype(int)
data['young'] = ((data['title_age'])<16)*1
# display(data['young'].head(20))
```

- [54]：以同樣的方法建立隨機森林分類模型，此時的使用特徵共有性別編碼、艙等、票價分區(五區間劃分)、同行者是否生還(connect_survived)和年齡是否在 16 歲以下(young)這六個特徵。模型 age_model 的 oob_score 從 0.8204 上升到 0.8418。上傳至 kaggle 後，預測準確度也從 0.79665 上升到 0.81100，準確率突破了 0.8，排名在整個競賽中位於第 316 名。

In [54]:
```
x_train = data[:len(train)]
x_test = data[len(train):]
y = x_train['Survived']
x_train = x_train.drop(labels=['Survived','PassengerId'],axis=1)

age = ['Sex_code', 'Pclass', 'Fare_Code_5', 'connect_survived', 'young']

age_model = RandomForestClassifier(random_state=2, n_estimators=250, min_samples_split=20, oob_score=True)
age_model.fit(x_train[age], y)
ans_age = age_model.predict(x_test[age])
print('age oob score = {}'.format(age_model.oob_score_.round(4)))

ans_age = ans_age.astype(int)
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':ans_age}).set_index('PassengerId').to_csv('./result/sexcode_pclass_farel
```

```
age oob score = 0.8418
```

sexcode_pclass_fareb5_connect_age.csv 0.81100

a day ago by 林子淩(Lin Tzu Ling0712118)

with age

316 林子淩(Lin Tzu Ling0712118) 0.81100 16 1d

Your Best Entry

Your submission scored 0.81100, which is an improvement of your previous score of 0.79665. Great job! Tweet this!