Titanic 練習

0853412 資管碩一 吳宛儒

1. 前置作業

1.1. 引入資料處理相關套件

```
# loading package
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(font_scale=1.56)
from sklearn.ensemble import RandomForestClassifier # 隨機森林
from sklearn.model_selection import cross_val_score,
                                    StratifiedKFold,
                                    learning_curve,
                                    train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import RFECV
```

1.2. 引入圖表相關套件和自定義函數

```
# for display dataframe
from IPython.display import display
from IPython.display import display_html
```

```
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table',
                                  'table style="display:inline"'),
                                  raw=True)
```

1.3. 引入其他處理套件

```
import warnings
warnings.filterwarnings("ignore")
```

2. 讀取資料

```
# loading data
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")
df_data = df_train.append(df_test)
```

3. 第一次 - 資料前處理

第一次，先建立Base Model

3.1. 顯示資料欄位

```
list(df_data)
```

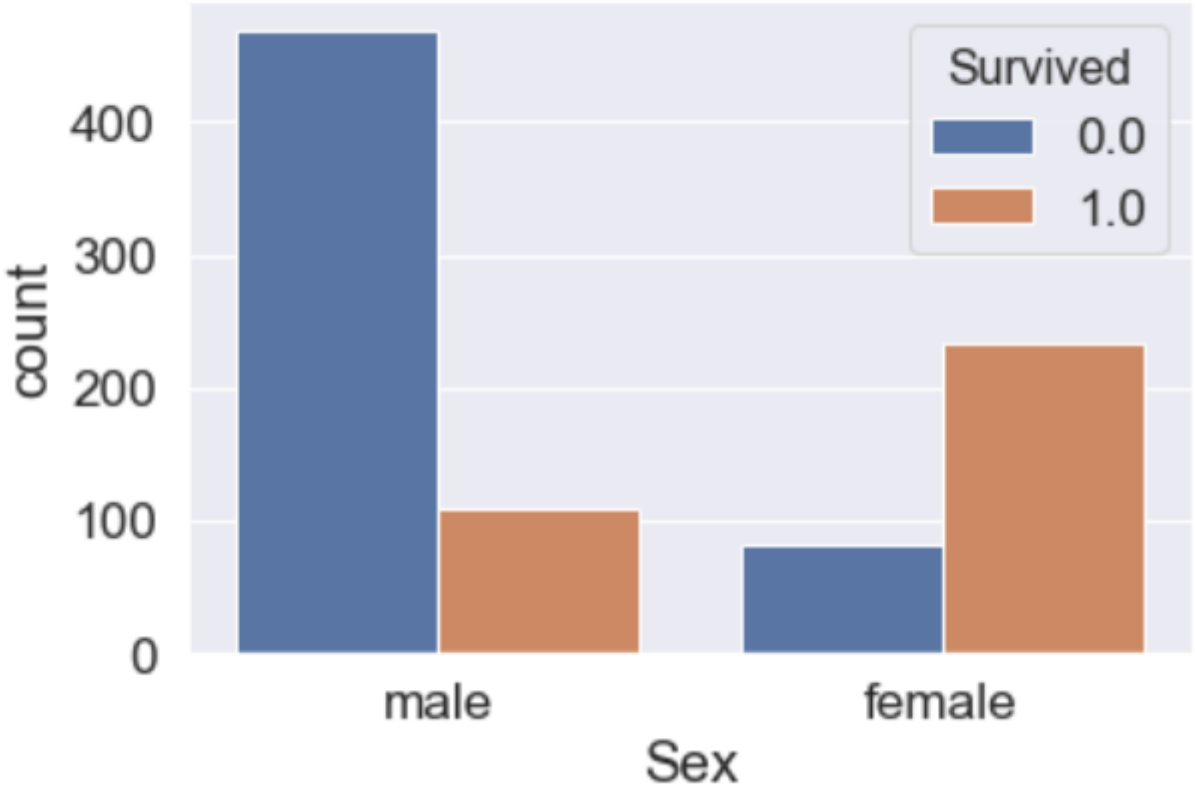
['Age',
'Cabin',
'Embarked',
'Fare',
'Name',
'Parch',
'PassengerId',
'Pclass',
'Sex',
'SibSp',
'Survived',
'Ticket']

3.2. 觀察資料欄位：性別 Sex

3.2.1. 計算並觀察各個性別的生死數量

```
sns.countplot(df_data['Sex'], hue=df_data['Survived'])
```

[5]: <matplotlib.axes._subplots.AxesSubplot at 0x24acbddd550>



3.2.2. 計算各個性別的生存率

```
display(df_data[["Sex", "Survived"]]\
        .groupby(['Sex'], as_index=False).mean().round(3))
```

	Sex	Survived
0	female	0.742
1	male	0.189

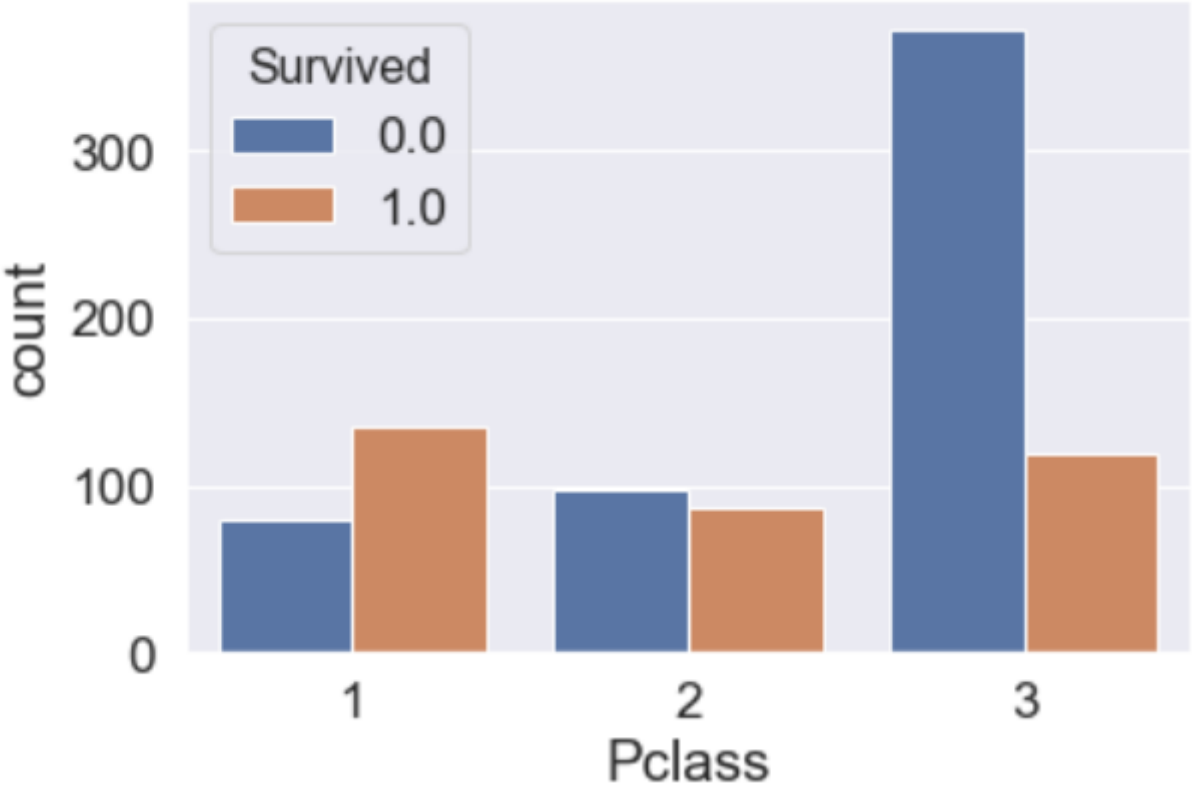
由此可見，大部分的男性都死亡了(僅18%存活)，而女性則是大部分存活(75%)

3.3. 觀察資料欄位：艙等 Pclass

3.3.1. 計算各個艙等的生死數量

```
sns.countplot(df_data['Pclass'], hue=df_data['Survived'])
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x24acc0d0e48>
```



3.3.2. 計算各個艙等的生存率

```
display(df_data[["Pclass", "Survived"]]\
        .groupby(['Pclass'],\
        as_index=False).mean().round(3))
```

	Pclass	Survived
0	1	0.630
1	2	0.473
2	3	0.242

由此可見，頭等艙生存率較高，次等艙次之

3.4. 處理資料欄位：性別 Sex

3.4.1. 觀察原本性別欄位：是由male,female組成

```
df_data['Sex']
```

```
[10]: 0        male
      1      female
      2      female
      3      female
```

3.4.2. 新建欄位Sex_Code，將female轉成0，male轉成1存入

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

```
[12]: 0      0
      1      1
      2      1
      3      1
```

4. 第一次 - 建立模型

4.1. 切出訓練/測試集

```
df_train = df_data[:len(df_train)]
df_test = df_data[len(df_train):]
```

4.2. 取出label/target: Survived 生存與否

4.2.1. 生還與否作為label/target： Y

```
Y = df_train['Survived']
```

4.2.2. 其餘切開待下一步處理欄位： X

```
X = df_train.drop(labels = ['Survived','PassengerId'], axis = 1)
```

4.3. 特徵選取

4.3.1. 以性別和艙等作為特徵

```
Base = ['Sex_Code', 'Pclass']
```

4.4. 建立模型

4.4.1. 建立隨機森林

參數	意義
random_state	產生隨機數字的seed
n_estimators	決策樹的個數，通常越多越好，但效能會下降。 有經驗指出至少要100左右才會有可接受的性能和誤差率。
min_samples_split	根據屬性劃分節點時，每個劃分最少的樣本數。
oob_score	隨機森林演算法是由一群決策樹組成， 單顆決策樹並不會用所有的訓練資料， 每一次選一塊資料來訓練一顆決策樹時， 剩下的資料並沒有被拿來訓練該顆決策樹，因此可以做為驗證資料， 即Out-Of-Bag。

```
Base_Model = RandomForestClassifier(random_state = 2,
                                    n_estimators = 250,
                                    min_samples_split = 20,
                                    oob_score = True)
```

```
# fit model
Base_Model.fit(X[Base], Y)
print('Base oob score : %.5f' % (Base_Model.oob_score_))
```

得到分數：`Base oob score : 0.73176`

先由Sex_Code和Pclass當作基準值，後面加入的特徵若是使得此準確率下降，代表加入了噪聲較大的特徵，或是已經overfitting，應該要回到最簡單的model慢慢一個個特徵加入並做觀察。

5. 第二次 - 資料前處理

有了第一次做出來的Base Model以後，再去觀察其他可能有助於提升預測準確率的欄位，以Base Model作為基準評斷加入的特徵是否噪聲太大降低準確率。

5.1. 觀察資料欄位：票價 Fare

- 票價和艙等都是屬於彰顯乘客社會地位的一個特徵
- 假設：買票價格較高的乘客，生存機率可能也較高

5.1.1. 觀察各種票價的生存率：分布廣且傾斜

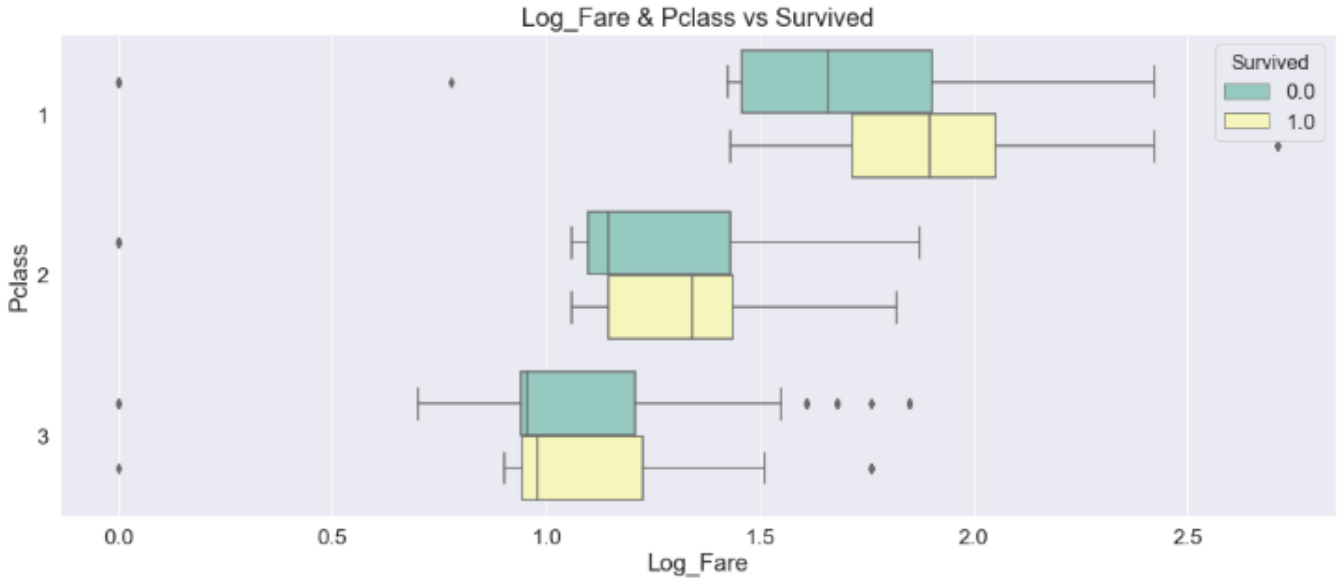
```
display(df_data[["Fare", "Survived"]]\
        .groupby(['Fare'], as_index=False).mean().round(3))
```

	Fare	Survived
0	0.000	0.067
1	3.171	NaN
2	4.012	0.000
3	5.000	0.000
4	6.238	0.000

281 rows × 2 columns

5.1.2. 示意圖：取log解決傾斜問題以後再繪製

```
fig, ax = plt.subplots(figsize = (18,7))
df_data['Log_Fare'] = (df_data['Fare']+1)\
                      .map(lambda x: np.log10(x) if x > 0 else 0)
sns.boxplot(y='Pclass', x='Log_Fare',
            hue='Survived', data=df_data,
            orient='h', ax=ax, palette="Set3")
ax.set_title(' Log_Fare & Pclass vs Survived ', fontsize = 20)
```




```
pd.pivot_table(df_data,values = ['Fare'],
               index = ['Pclass'],
               columns=['Survived'],
               aggfunc = 'median').round(3)
```

[32]:

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

可以看出存活下來的乘客確實平均而言付出較高的票價
因此決定考量票價這項特徵

5.2. 處理資料欄位：票價 Fare

將票價切成區間，以避免overfitting

5.2.1. 先填補缺失值

```
df_data['Fare'] = df_data['Fare']\
                  .fillna(df_data['Fare'].median())
```

5.2.2. 考量票價區間

- 區間太少：
  - 區間內的資料太多一起平均，無法看出差異性，使得特徵失真
- 區間太多：
  - 一點點票價的不同，都影響了生存率的高低，容易overfitting
  - 切分區間趨近於無限大時，就回到了原本的數值特徵

5.2.3. 將票價分別切成4,5,6個區間當作新的特徵，分別觀察好壞

方法	意義
pandas.qcut(欄位, 切成幾等分)	qcut是以累積百分比來切分的，例如將參數=4，就會以0%~25%, 25%~50%, 50%~75%, 75%~100% 來切分資料， 好處是我們可以避免某個區間內的資料過少(skew problem)

```
df_data['FareBin_4'] = pd.qcut(df_data['Fare'], 4)
df_data['FareBin_5'] = pd.qcut(df_data['Fare'], 5)
df_data['FareBin_6'] = pd.qcut(df_data['Fare'], 6)
```

方法	意義
Label Encoder	對分類型特徵值進行編碼，即對不連續的數值或文本進行編碼
fit(y)	fit可看做一本空字典，y可看作要塞到字典中的詞
transform(y)	將y轉變成索引值
fit_transform(y)	相當於先進行fit再進行transform， 即把y塞到字典中去以後再進行transform得到索引值
inverse_transform(y)	根據索引值y獲得原始數據

```
label = LabelEncoder()
df_data['FareBin_Code_4'] = label.fit_transform(df_data['FareBin_4'])
df_data['FareBin_Code_5'] = label.fit_transform(df_data['FareBin_5'])
df_data['FareBin_Code_6'] = label.fit_transform(df_data['FareBin_6'])
```

處理過後，資料的樣子為：

```
df_data['FareBin_Code_4']
```

```
[41]: 0    0
      1    3
      2    1
      3    3
      4    1
```

5.2.4. 列出交叉表觀察相互關係

方法	意義
透視表pivot_table()	它根據一個或多個鍵對數據進行聚合， 並根據行和列上得分組建將數據分配到各個矩形區域中。
交叉表crosstab()	是一種特殊的pivot_table()，專用於計算分組頻率， 前兩個參數可以是數組、Series或數組列表。

```
# cross tab
df_4 = pd.crosstab(df_data['FareBin_Code_4'],df_data['Pclass'])
df_5 = pd.crosstab(df_data['FareBin_Code_5'],df_data['Pclass'])
df_6 = pd.crosstab(df_data['FareBin_Code_6'],df_data['Pclass'])
```

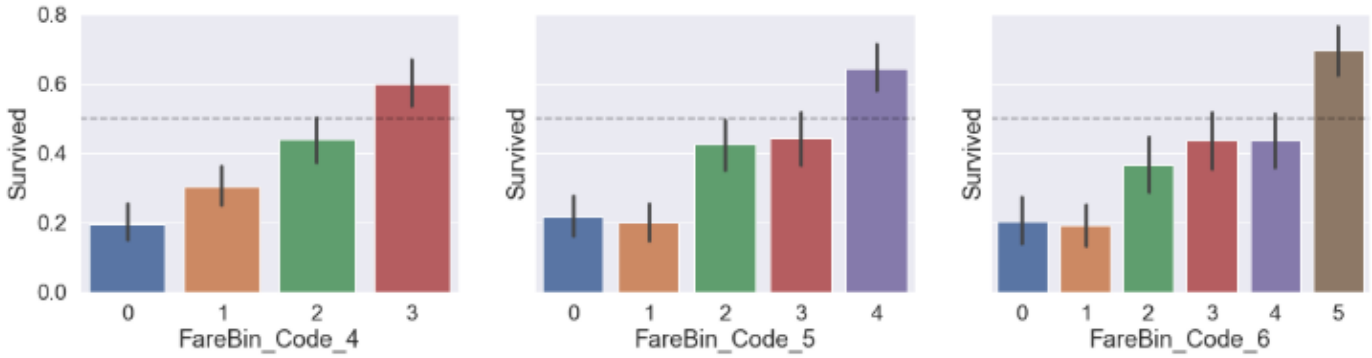
```
display_side_by_side(df_4,df_5,df_6)
```

Pclass	1	2	3
FareBin_Code_4			
0	8	6	323
1	0	128	193
2	77	104	147
3	238	39	46

Pclass	1	2	3
FareBin_Code_5			
0	8	6	261
1	0	36	218
2	0	124	132
3	95	99	71
4	220	12	27

Pclass	1	2	3
FareBin_Code_6			
0	8	6	222
1	0	0	218
2	0	128	76
3	14	83	128
4	118	48	46
5	183	12	19

```
# plots
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=.3)

g1 = sns.factorplot(x='FareBin_Code_4',y="Survived",
                    data=df_data,kind='bar',ax=ax1)
g2 = sns.factorplot(x='FareBin_Code_5',y="Survived",
                    data=df_data,kind='bar',ax=ax2)
g3 = sns.factorplot(x='FareBin_Code_6',y="Survived",
                    data=df_data,kind='bar',ax=ax3)

# close FacetGrid object
plt.close(g1.fig)
plt.close(g2.fig)
plt.close(g3.fig)
```



6. 第二次 - 建立模型

6.1. 切出訓練/測試集

```
df_train = df_data[:len(df_train)]
df_test = df_data[len(df_train):]
```

6.2. 取得label/target：Survived 生存與否

```
X = df_train.drop(labels=['Survived', 'PassengerId'], axis=1)
Y = df_train['Survived']
```

備註：

欄位	意義
X['FareBin_6']	顯示被分到哪個區間
X['FareBin_Code_6']	顯示被分到第幾個區間

6.3. 特徵選取

6.3.1. 票價欄位究竟切成幾分才是最好的

- feature selection特徵選擇問題
  - 可用方法：單變數的Chi square或information gain
  - 選用方法：前項選擇RFE
    - 可以考慮特徵之間的交互作用，但比較吃資源
    - 不過鐵達尼資料及比較少所以影響比較不大，在sklearn上實作也很容易

6.3.2. 列出要比較的相關欄位

```
compare = ['Sex_Code','Pclass',
            'FareBin_Code_4','FareBin_Code_5','FareBin_Code_6']
```

6.3.3. 利用RFECV找出相互關係

方法	意義
RFECV	比REF多一個交叉比對的分數(gridscores)，代表選擇多少特徵後的準確率。 但REFCV不用像REF要給定選擇多少特徵， 而是會依照交叉比對的分數而自動選擇訓練的特徵數。

```
selector = RFECV(RandomForestClassifier(n_estimators=250,
                                        min_samples_split=20),
                                        cv=10,n_jobs=-1)
selector.fit(X[compare],Y)
```

6.3.4. 印出各自得分

```
print(selector.support_)
print(selector.ranking_)
print(selector.grid_scores_*100)
```

```
[ True  True  True  True  True]
[1 1 1 1 1]
[78.66981614 77.33398593 79.45885825 79.5762683  80.47642152]
```

由此可見切成6份可以得到比較好的CV分數
但是尚未考慮到模型的random_state以及Cross-Validation切分的方式

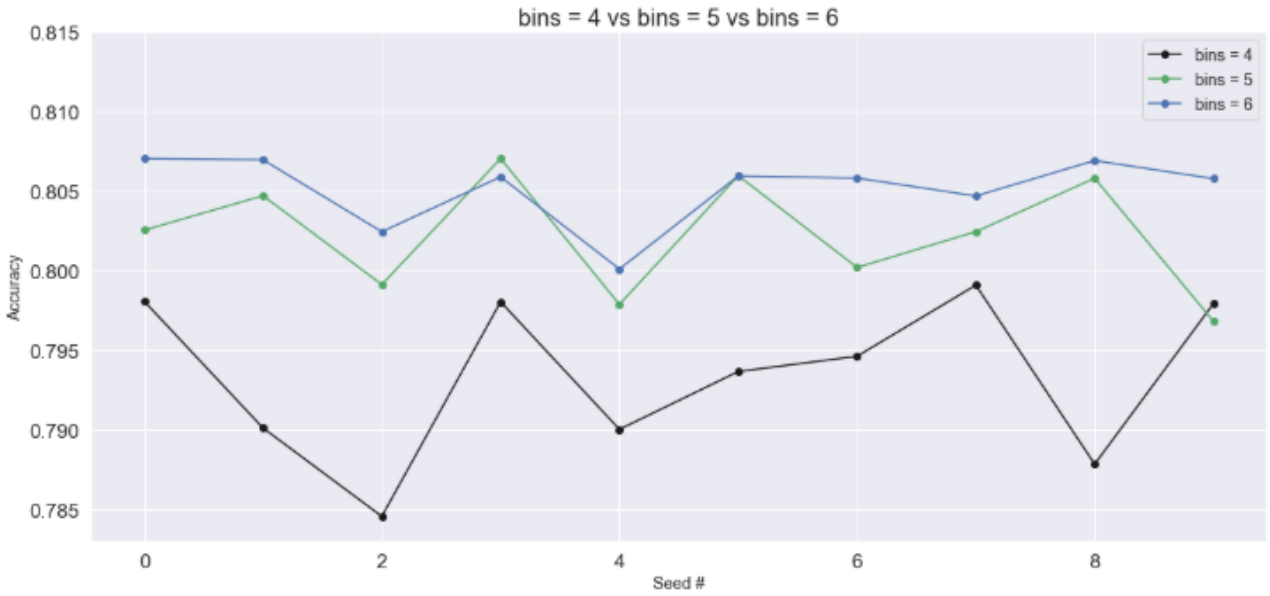
6.3.5. 進一步針對CV及模型的random_state進行實驗

方法	意義
StratifiedKFold	用法類似Kfold，但是他是分層採樣，確保訓練集， 測試集中各類別樣本的比例與原始數據集中相同。

```
score_b4, score_b5, score_b6 = [],[],[]
seeds=10
for i in range(seeds):
    diff_cv = StratifiedKFold(n_splits=10,
                              shuffle=True,
                              random_state=i)
    selector = RFECV(RandomForestClassifier(random_state=i,
                                            n_estimators=250,
                                            min_samples_split=20),
                     cv=diff_cv, n_jobs=-1)
    selector.fit(X[compare],Y)
    score_b4.append(selector.grid_scores_[2])
    score_b5.append(selector.grid_scores_[3])
    score_b6.append(selector.grid_scores_[4])
```

```
score_list = [score_b4, score_b5, score_b6]
for item in score_list:
    item = np.array(item*100)
#plot
fig = plt.figure(figsize=(18,8))
ax = plt.gca()
ax.plot(range(seeds), score_b4, '-ok', label='bins = 4')
ax.plot(range(seeds), score_b5, '-og', label='bins = 5')
ax.plot(range(seeds), score_b6, '-ob', label='bins = 6')
ax.set_xlabel("Seed #", fontsize= '14')
ax.set_ylim(0.783,0.815)
ax.set_ylabel("Accuracy", fontsize = '14')
ax.set_title('bins = 4 vs bins = 5 vs bins = 6', fontsize = '20')
plt.legend(fontsize = 14, loc='upper right')
```

[73]: <matplotlib.legend.Legend at 0x24ad1722f98>



由上圖我們可以看出切分成4份的準確率較低，6份比5份稍微好一點

6.3.6. 提交到Kaggle上並顯示oob分數以及提交的結果

```
b4,b5,b6 = ['Sex_Code','Pclass','FareBin_Code_4'],\
['Sex_Code','Pclass','FareBin_Code_5'],\
['Sex_Code','Pclass','FareBin_Code_6']
```

```
b4_Model = RandomForestClassifier(random_state=2,
                                  n_estimators=250,
                                  min_samples_split=20,
                                  oob_score=True)
b4_Model.fit(X[b4], Y)
b5_Model = RandomForestClassifier(random_state=2,
                                  n_estimators=250,
                                  min_samples_split=20,
                                  oob_score=True)
b5_Model.fit(X[b5], Y)
b6_Model = RandomForestClassifier(random_state=2,
                                  n_estimators=250,
                                  min_samples_split=20,
                                  oob_score=True)
b6_Model.fit(X[b6], Y)
```

```
print('b4 oob score :%.5f' %(b4_Model.oob_score_),
      '  LB_PUBLIC : 0.7790')
print('b5 oob score :%.5f' %(b5_Model.oob_score_),
      '  LB_PUBLIC : 0.79425')
print('b6 oob score :%.5f' %(b6_Model.oob_score_),
      '  LB_PUBLIC : 0.77033')
```

```
b4 oob score :0.80584   LB_PUBLIC : 0.7790
b5 oob score :0.81033   LB_PUBLIC : 0.79425
b6 oob score :0.80135   LB_PUBLIC : 0.77033
```

在排行榜上的分數反而是切分成5份最高分，而不是6份，
這個情況可能是特徵工程過度所帶來的overfitting，
因此在CV以及Kaggle排行榜上測試都是必要且值得參考的。

6.4. 上傳Kaggle

6.4.1. 最終決定將b5_Model提交至Kaggle

```
X_Submit = df_test.drop(labels=['PassengerId'],axis=1)

b5_pred = b5_Model.predict(X_Submit[b5])

submit = pd.DataFrame({"PassengerId": df_test['PassengerId'],
                       "Survived":b5_pred.astype(int)})
submit.to_csv("submit_b5.csv", index=False)
```

7. 第三次 - 資料前處理

7.1. 觀察資料欄位：票根 Ticket

7.1.1. 列出票根欄位敘述

發現了乘客持有相同的船票代表他們可能是家人或是朋友，
而在訓練集上這些互相有連結的人常常是一起活下來或是一起喪命

方法	意義
describe()	描述統計

```
df_train['Ticket'].describe()
```

```
[92]: count         891
      unique        681
      top        347082
      freq            7
      Name: Ticket, dtype: object
```

可見獨立的票根資訊有681項
代表有乘客是持有相同票根上船的，也可能分享某一區的座位

7.2. 處理資料欄位：票根 Ticket

7.2.1. 建立新的特徵：Family_size 家庭人數

家庭人數 = 兄弟姊妹數 + 父母小孩數 + 1

```
df_data['Family_size'] = df_data['SibSp'] + df_data['Parch'] + 1
```

7.2.2. 建立持有相同票根的DataFrame，並顯示姓名、票價、艙位、家庭人數

```
deplicate_ticket = []
for tk in df_data.Ticket.unique():
    tem = df_data.loc[df_data.Ticket == tk, 'Fare']
    #print(tem.count())
    if tem.count() > 1:
        deplicate_ticket.append(df_data.loc[df_data.Ticket == tk,
                                            ['Name','Ticket',
                                             'Fare','Cabin',
                                             'Family_size',
                                             'Survived']])
# 具有相同票根 (dataframe)
deplicate_ticket = pd.concat(deplicate_ticket)

# 列出前14筆具有相同票根的人類
deplicate_ticket.head(14)
```

[108]:

	Name	Ticket	Fare	Cabin	Family_size	Survived
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	PC 17599	71.2833	C85	2	1.0
234	Cumings, Mr. John Bradley	PC 17599	71.2833	C85	2	NaN
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	113803	53.1000	C123	2	1.0
137	Futrelle, Mr. Jacques Heath	113803	53.1000	C123	2	0.0
6	McCarthy, Mr. Timothy J	17463	51.8625	E46	1	0.0
146	Hilliard, Mr. Herbert Henry	17463	51.8625	E46	1	NaN
7	Palsson, Master. Gosta Leonard	349909	21.0750	NaN	5	0.0
24	Palsson, Miss. Torborg Danira	349909	21.0750	NaN	5	0.0
374	Palsson, Miss. Stina Viola	349909	21.0750	NaN	5	0.0
567	Palsson, Mrs. Nils (Alma Cornelia Berglund)	349909	21.0750	NaN	5	0.0
389	Palsson, Master. Paul Folke	349909	21.0750	NaN	5	NaN
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	347742	11.1333	NaN	3	1.0
172	Johnson, Miss. Eleanor Ileen	347742	11.1333	NaN	3	1.0
869	Johnson, Master. Harold Theodor	347742	11.1333	NaN	3	1.0

觀察各個群組，可以看出大部分皆有同生同死的狀況，
同一群組有可能會是家庭關係(從姓氏看出來)，也可能不是(姓氏不同)
可能是共同原因一起搭船的人，也會在船難時互相幫忙

7.2.3. 新建特徵：家庭成員人數

- Family_size = 1 但是又在群組內的
  - 即非親屬關係，我們歸類為朋友
- Family_size > 1
  - 則為家人

```
df_fri = deplicate_ticket.loc[(deplicate_ticket.Family_size == 1)
                  & (deplicate_ticket.Survived.notnull())].head(7)
df_fami = deplicate_ticket.loc[(deplicate_ticket.Family_size > 1)
                  & (deplicate_ticket.Survived.notnull())].head(7)
```

```
display(df_fri,df_fami)
```

	Name	Ticket	Fare	Cabin	Family_size	Survived
6	McCarthy, Mr. Timothy J	17463	51.8625	E46	1	0.0
20	Fynney, Mr. Joseph J	239865	26.0000	NaN	1	0.0
791	Gaskell, Mr. Alfred	239865	26.0000	NaN	1	0.0
195	Lurette, Miss. Elise	PC 17569	146.5208	B80	1	1.0
681	Hassab, Mr. Hammad	PC 17572	76.7292	D49	1	1.0
61	Icard, Miss. Amelie	113572	80.0000	B28	1	1.0
829	Stone, Mrs. George Nelson (Martha Evelyn)	113572	80.0000	B28	1	1.0

	Name	Ticket	Fare	Cabin	Family_size	Survived
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	PC 17599	71.2833	C85	2	1.0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	113803	53.1000	C123	2	1.0
137	Futrelle, Mr. Jacques Heath	113803	53.1000	C123	2	0.0
7	Palsson, Master. Gosta Leonard	349909	21.0750	NaN	5	0.0
24	Palsson, Miss. Torborg Danira	349909	21.0750	NaN	5	0.0
374	Palsson, Miss. Stina Viola	349909	21.0750	NaN	5	0.0
567	Palsson, Mrs. Nils (Alma Cornelia Berglund)	349909	21.0750	NaN	5	0.0

```
print('people keep the same ticket: %.0f' %len(deplicate_ticket))
print('friends: %.0f'
    %len(deplicate_ticket[deplicate_ticket.Family_size == 1]))
print('familes: %.0f'
    %len(deplicate_ticket[deplicate_ticket.Family_size > 1]))
```

```
people keep the same ticket: 596
friends: 127
familes: 469
```

有約莫600位乘客和他人持有相同票根，其中大概有75%為家庭出遊

7.2.4. 新建特徵：Connected_Survival 生存連結

因為持有同樣票根的人可能會有相互連結的生死關係，
因此希望上表中票根 PC 17599 中的乘客建立Connected_Survival = 1
以及票根 17463 349909 中的乘客建立Connected_Survival = 0
但還須要多考慮沒有生還資訊的乘客(同個票根中Survived都是NaN 在測試集中)，
將Connected_Survival = 0.5

```
# 沒有生還資訊者Connected_Survival設為0.5
df_data['Connected_Survival'] = 0.5

for _, df_grp in df_data.groupby('Ticket'):
    if(len(df_grp) > 1): # 有某種票根的人數大於一人
        for ind, row in df_grp.iterrows():

            smax = df_grp.drop(ind)['Survived'].max()
            smin = df_grp.drop(ind)['Survived'].min()

            passID = row['PassengerId']
            if(smax == 1.0):
                df_data.loc[df_data['PassengerId'] == passID,
                            'Connected_Survival'] = 1
            elif(smin==0.0):
                df_data.loc[df_data['PassengerId'] == passID,
                            'Connected_Survival'] = 0

print('people keep the same ticket: %.0f'%len(deplicate_ticket))
print('people have connected information: %.0f'
    %(df_data[df_data['Connected_Survival'] != 0.5].shape[0]))
df_data.groupby('Connected_Survival')[['Survived']].mean().round(3)
```

```
people keep the same ticket: 596
people have connected information: 496
```

[119]:

Connected_Survival	Survived
0.0	0.225
0.5	0.298
1.0	0.728

8. 第三次 - 建立模型

8.1. 切成訓練/測試集

```
df_train = df_data[:len(df_train)]
df_test = df_data[len(df_train):]
```

8.2. 取出label/target: Survived 生存與否

```
X = df_train.drop(labels = ['Survived','PassengerId'], axis = 1)
Y = df_train['Survived']
```

8.3. 特徵選取

性別、艙等、票價、生存連結

```
connect = ['Sex_Code','Pclass',
            'FareBin_Code_5','Connected_Survival']
```

8.4. 建立模型

```
connect_Model = RandomForestClassifier(random_state=2,
                                       n_estimators=250,
                                       min_samples_split=20,
                                       oob_score=True)
connect_Model.fit(X[connect], Y)
print('connect oob score :%.5f'%(connect_Model.oob_score_))
```

```
connect oob score :0.82043
```

8.5. 提交Kaggle

```
X_Submit = df_test.drop(labels=['PassengerId'], axis = 1)

connect_pred = connect_Model.predict(X_Submit[connect])

submit = pd.DataFrame({"PassengerId": df_test['PassengerId'],
                       "Survived": connect_pred.astype(int)})
submit.to_csv("submit_connect.csv",index=False)
```

9. 第四次 - 資料前處理

9.1. 觀察資料欄位：年齡 Age

- Age大約有20%左右的缺失值，不像先前的票價Fare只有一項缺失值
  - 可能會因此影響預測

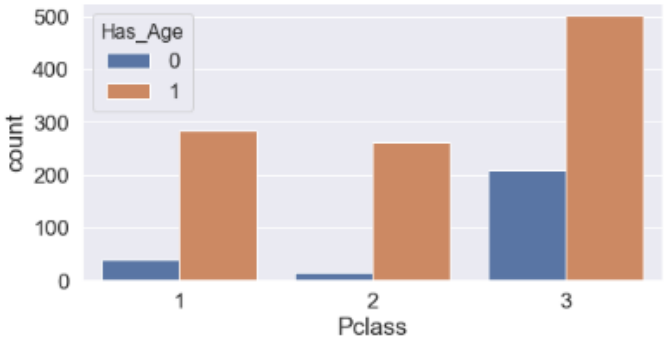
9.1.1. 觀察缺失值分布的情況：建立Has_Age欄位，有為1，沒有為0

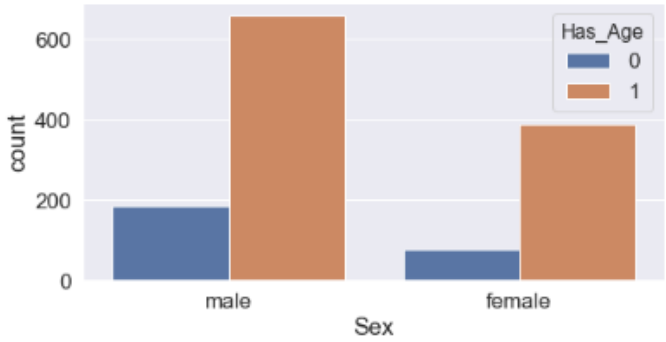
- 缺失年齡的數筆資料是否都剛好是某種性別或是艙等?
  - if so,很有可能影響預測

```
df_data['Has_Age'] =
    df_data['Age'].isnull().map(lambda x : 0 if x == True else 1)
```

9.1.2. 以性別和艙等計算缺失值數量

```
# plot
fig, [ax1, ax2] = plt.subplots(1,2)
fig.set_figwidth(18)

ax1 = sns.countplot(df_data['Pclass'],
                    hue = df_data['Has_Age'],
                    ax = ax1)
ax2 = sns.countplot(df_data['Sex'],
                    hue = df_data['Has_Age'],
                    ax = ax2)
```





9.1.3. 建立交叉表看相互關係

```
pd.crosstab(df_data['Has_Age'],
            df_data['Sex'], margins=True).round(3)
```

[134]:

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

年齡值的缺失大部分發生在三等艙
如果年齡很重要，那三等艙的觀察就會失真

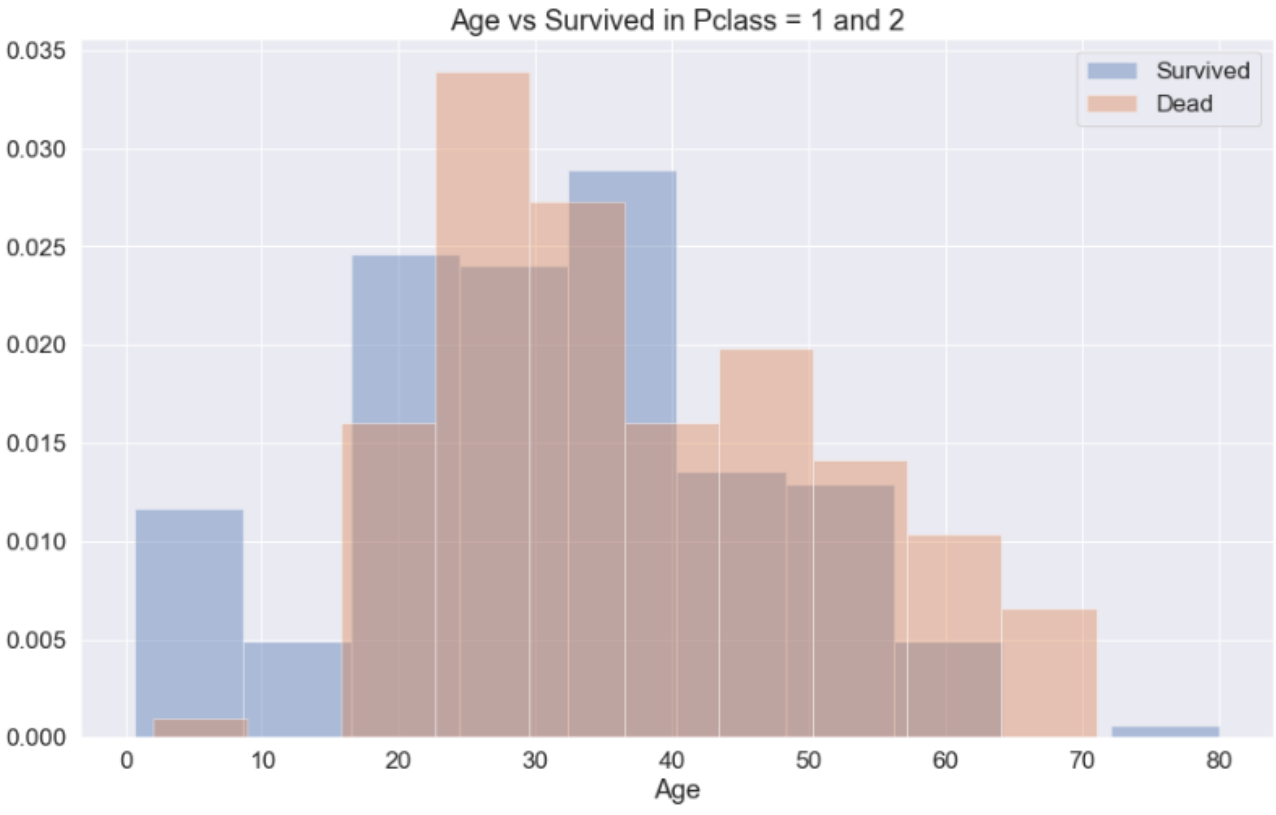
9.1.2. 先觀察1,2艙等中，年齡對生存與否的影響

```
Mask_Has_Age_P12_Survived = ((df_data.Has_Age == 1)&\
                             (df_data.Pclass != 3)&\
                             (df_data.Survived == 1))
Mask_Has_Age_P12_Dead = ((df_data.Has_Age == 1)&\
                         (df_data.Pclass != 3)&\
                         (df_data.Survived == 0))
```

方法	意義
displot	seaborn中的直方圖
subplots	同樣的資料想要用不同的視角呈現時即可使用子圖表
legend()	顯示數據的名稱

```
fig, ax = plt.subplots(figsize = (15,9))
ax = sns.distplot(df_data.loc[Mask_Has_Age_P12_Survived, 'Age'],
                  kde=False, bins=10, norm_hist=True,
                  label='Survived')
ax = sns.distplot(df_data.loc[Mask_Has_Age_P12_Dead, 'Age'],
                  kde=False, bins=10, norm_hist=True,
                  label='Dead')
ax.legend()
ax.set_title('Age vs Survived in Pclass = 1 and 2', fontsize = 20)
```

```
[139]: Text(0.5, 1.0, 'Age vs Survived in Pclass = 1 and 2')
```



- 圖中我們可以看到，左邊藍色有部分突出顯示生存率較高，大約在**小於16歲**，表示青少年以下會有比較高的生存率

- 而如果是**大於16歲**，基本上年齡不能算是顯著的特徵來判定是否生還
- 在70~80歲的這個區間，因為樣本數太少，不列入採計

1. 綜上討論，找出那些<16歲的缺失值是重要的，這會影響預測，而>16歲的部分則不採用，否則只是擬合了噪聲。
2. 因此年齡這個特徵可以抽取出<16歲及>16歲做為一個二元特徵。

9.1.3. 填入缺失值：使用姓名當中的稱謂中位數來填補

```
df_data['Title'] = df_data.Name.str.extract(' ([A-Za-z]+)\.',
                                            expand=False)
df_data['Title'] = df_data['Title'].replace(['Capt', 'Col',
                                             'Countess', 'Don',
                                             'Dr', 'Dona',
                                             'Jonkheer', 'Major',
                                             'Rev','Sir'],'Rare')
df_data['Title'] = df_data['Title'].replace(['Mlle', 'Ms','Mme'],
                                            'Miss')
df_data['Title'] = df_data['Title'].replace(['Lady'],'Mrs')
df_data['Title'] = df_data['Title'].map({"Mr":0, "Rare" : 1,
                                         "Master" : 2,"Miss" : 3,
                                         "Mrs" : 4 })
Ti = df_data.groupby('Title')['Age'].median()
Ti
```

```
[149]: Title
       0    29.0
       1    47.0
       2     4.0
       3    22.0
       4    36.0
       Name: Age, dtype: float64
```

上表顯示了年齡的中位數，
先生 - 29歲，罕見稱謂 - 47歲，小男孩 - 4歲，小姐- 22歲， 女士 - 36歲

9.1.4. 新建欄位：填滿年齡的Ti_Age，不動原始特徵Age

```
Ti_pred = df_data.groupby('Title')['Age'].median().values
df_data['Ti_Age'] = df_data['Age']
# Filling the missing age
for i in range(0,5):
 # 0 1 2 3 4 5
    df_data.loc[(df_data.Age.isnull()) &\
    (df_data.Title == i),'Ti_Age'] = Ti_pred[i]
df_data['Ti_Age'] = df_data['Ti_Age'].astype('int')
```

9.1.5. 新建欄位：Ti_Minor，分為<16歲及>16歲

```
df_data['Ti_Minor'] = ((df_data['Ti_Age']) < 16.0) * 1
```

10. 第四次 - 建立模型

10.1. 切出訓練/測試集

```
# splits again beacuse we just engineered new feature
df_train = df_data[:len(df_train)]
df_test = df_data[len(df_train):]
```

10.2. 取出label/target: Survived 生存與否

```
# Training set and labels
X = df_train.drop(labels=['Survived','PassengerId'],axis=1)
Y = df_train['Survived']
```

10.3. 建立模型

```
minor = ['Sex_Code','Pclass',
         'FareBin_Code_5','Connected_Survival','Ti_Minor']
minor_Model = RandomForestClassifier(random_state=2,
                                     n_estimators=250,
                                     min_samples_split=20,
                                     oob_score=True)
minor_Model.fit(X[minor], Y)
print('minor oob score :%.5f' %(minor_Model.oob_score_))
```

```
minor oob score :0.84175
```

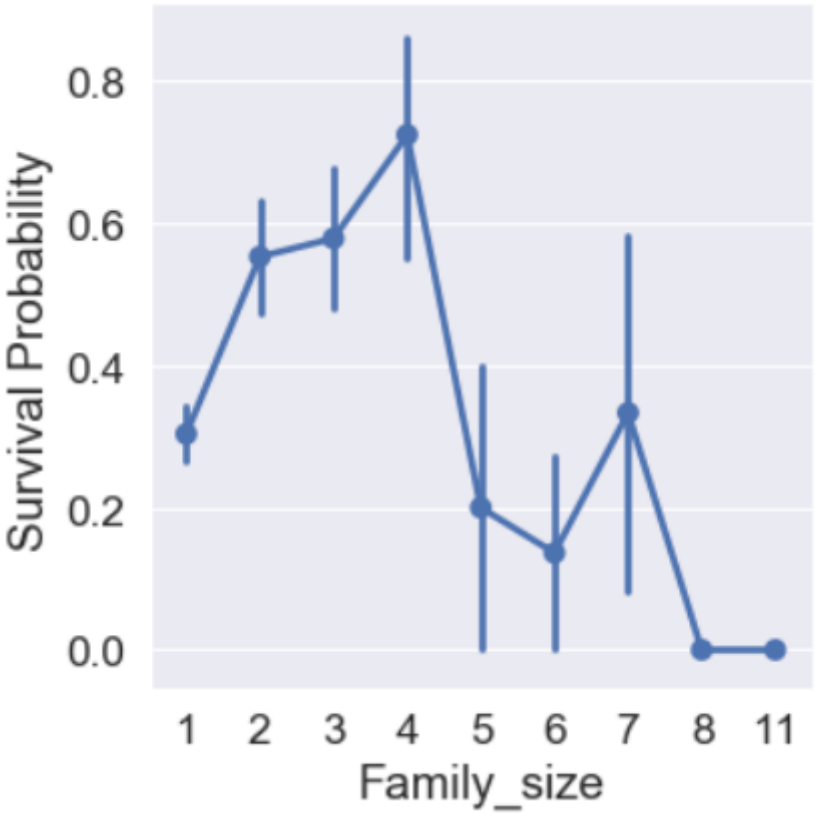
10.4. 提交Kaggle

```
# submits
X_Submit = df_test.drop(labels=['PassengerId'],axis=1)

minor_pred = minor_Model.predict(X_Submit[minor])

submit = pd.DataFrame({"PassengerId": df_test['PassengerId'],
                       "Survived":minor_pred.astype(int)})
submit.to_csv("submit_minor.csv",index=False)
```

11. 其他嘗試但沒有採用

11.1. 觀察資料欄位：家庭人數 Famliy_size

```
g = sns.factorplot(x='Family_size', y='Survived',data=df_data)
g = g.set_ylabels("Survival Probability")
```



可以看出獨自一人生存率較低，2~4人生存率較高，5人以上生存率又下降。因此猜測一人比較無法求援，2~4人能夠相互幫忙，5人以上又太多人多事雜。

11.1.1. 將家庭人數分為1人、2~4人和5人以上

```
df_data['L_Family'] = df_data['Family_size']\
                        .apply(lambda x: 0 if x<= 4 else 1)\
                        .astype(int)

# # 1人 FamilyClass=0
df_data.loc[ df_data['Family_size'] == 1, 'FamilyClass'] = 0
# 2~4人 FamilyClass = 1
df_data.loc[ (df_data['Family_size'] <= 4) &\
        (df_data['Family_size'] > 1), 'FamilyClass'] = 1
# 5人以上 FamilyClass = 2
df_data.loc[ df_data['Family_size'] >= 5, 'FamilyClass'] = 2

df_data['FamilyClass'] = df_data['FamilyClass'].astype(int)
df_data[['FamilyClass','Survived']]\
        .groupby(['FamilyClass']).mean()
```

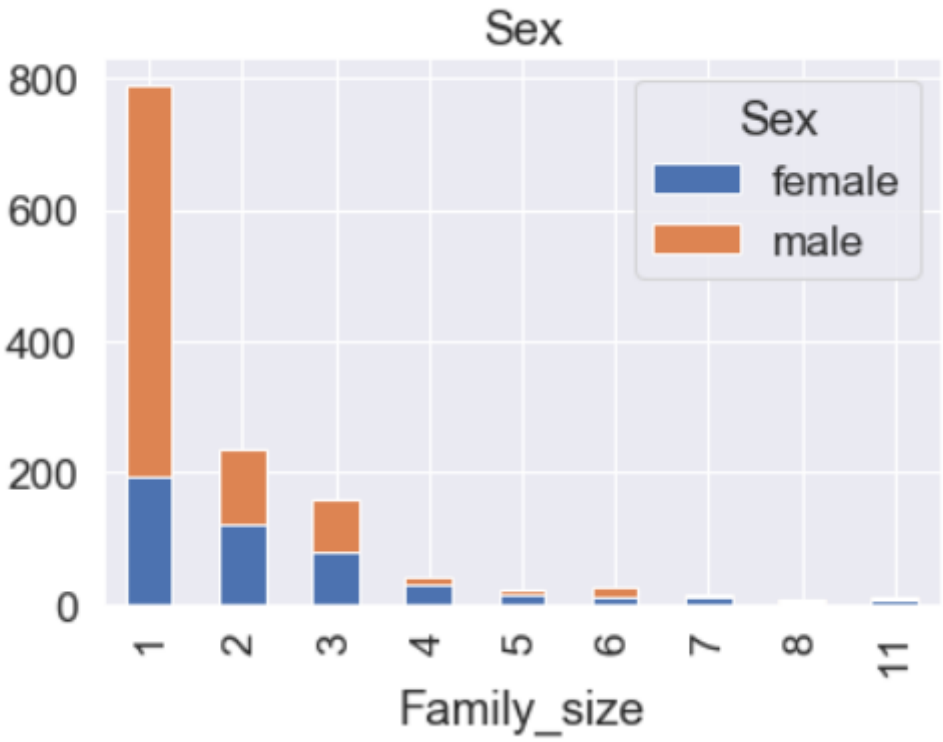
[157]:

FamilyClass	Survived
0	0.303538
1	0.578767
2	0.161290

11.1.2. 家庭人數對性別

```
1 pd.crosstab(df_data['Family_size'],df_data['Sex'])\
2             .plot(kind='bar',stacked=True,title="Sex")
```

[158]: <matplotlib.axes._subplots.AxesSubplot at 0x24ad2f30240>

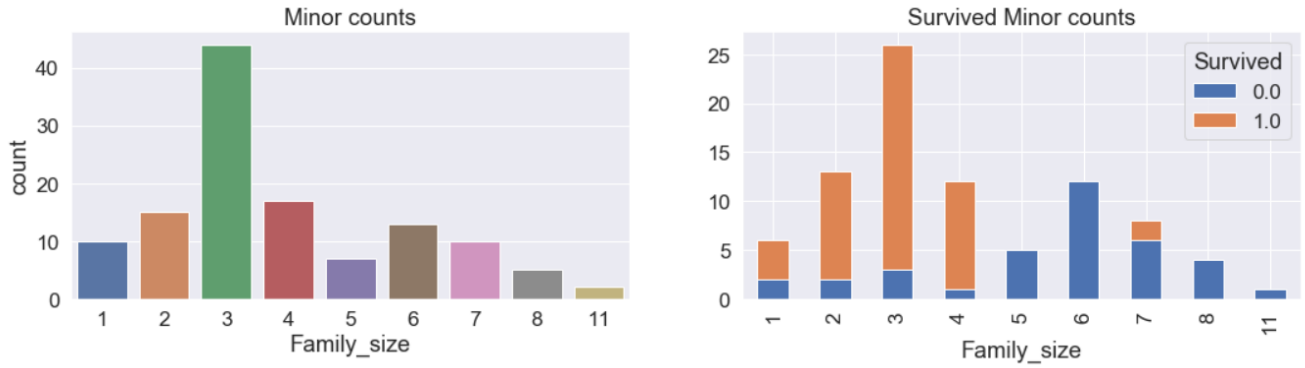


在一開始便可以透過對性別與生存率的觀察知道男性的生存率遠比女性還低，而透過這張圖表，可以發現家庭人數為1的大部分皆為男性，所以可能是因為男性普遍生存率低，造成家庭人數為1者看似生存率比較低。
因此，若我們加入家庭人數考量，有可能會重複考慮到相同特徵，造成冗餘。

11.1.3. 家庭人數對16歲以下人數

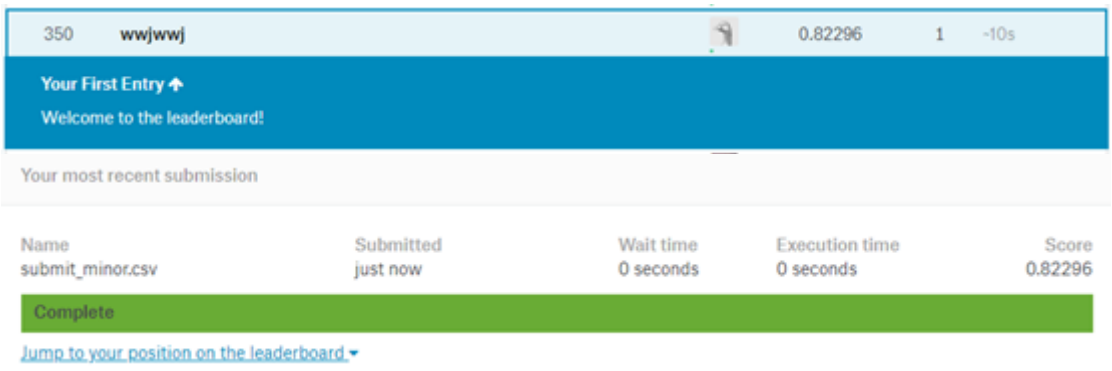
```
Minor_mask = (df_data.Ti_Minor == 1)
fig, [ax1, ax2] = plt.subplots(1, 2)
fig.set_figwidth(18)
sns.countplot(df_data[Minor_mask]['Family_size'],ax=ax1)
ax1.set_title("Minor counts")
pd.crosstab(df_data[Minor_mask]['Family_size'],\
            df_data[Minor_mask]['Survived'])\
            .plot(kind='bar',stacked=True,
            title="Survived Minor counts",ax=ax2)
```

[159]: <matplotlib.axes._subplots.AxesSubplot at 0x24ad2ef3860>



再者，可以看到2 ~ 4人的家庭中，含有16歲以下人數也是比例最高的，因此同理，有可能正因為這樣的特性，使得2 ~ 4人的家庭看似生存率比較高。
若加入這項考量，就會重複並產生冗餘。

最終上傳截圖



參考資料

[機器學習專案] Kaggle競賽-鐵達尼號生存預測(Top 3%) (https://medium.com/@yulongtsai/https-medium-com-yulongtsai-titanic-top3-8e64741cc11f)

Pandas透視表和交叉表 (https://blog.csdn.net/hustqb/article/details/78086394)

Python數據預處理中的LabelEncoder與OneHotEncoder (https://blog.csdn.net/quintind/article/details/79850455)

特徵選擇/範例三: Recursive feature elimination with cross-validation (https://machine-learning-python.kspax.io/intro-1/ex3_rfe_crossvalidation__md)