嘉義市政府
Chiayi City Government

107年度物聯網應用試辦暨資料收集分析案 教育訓練

機器學習基石

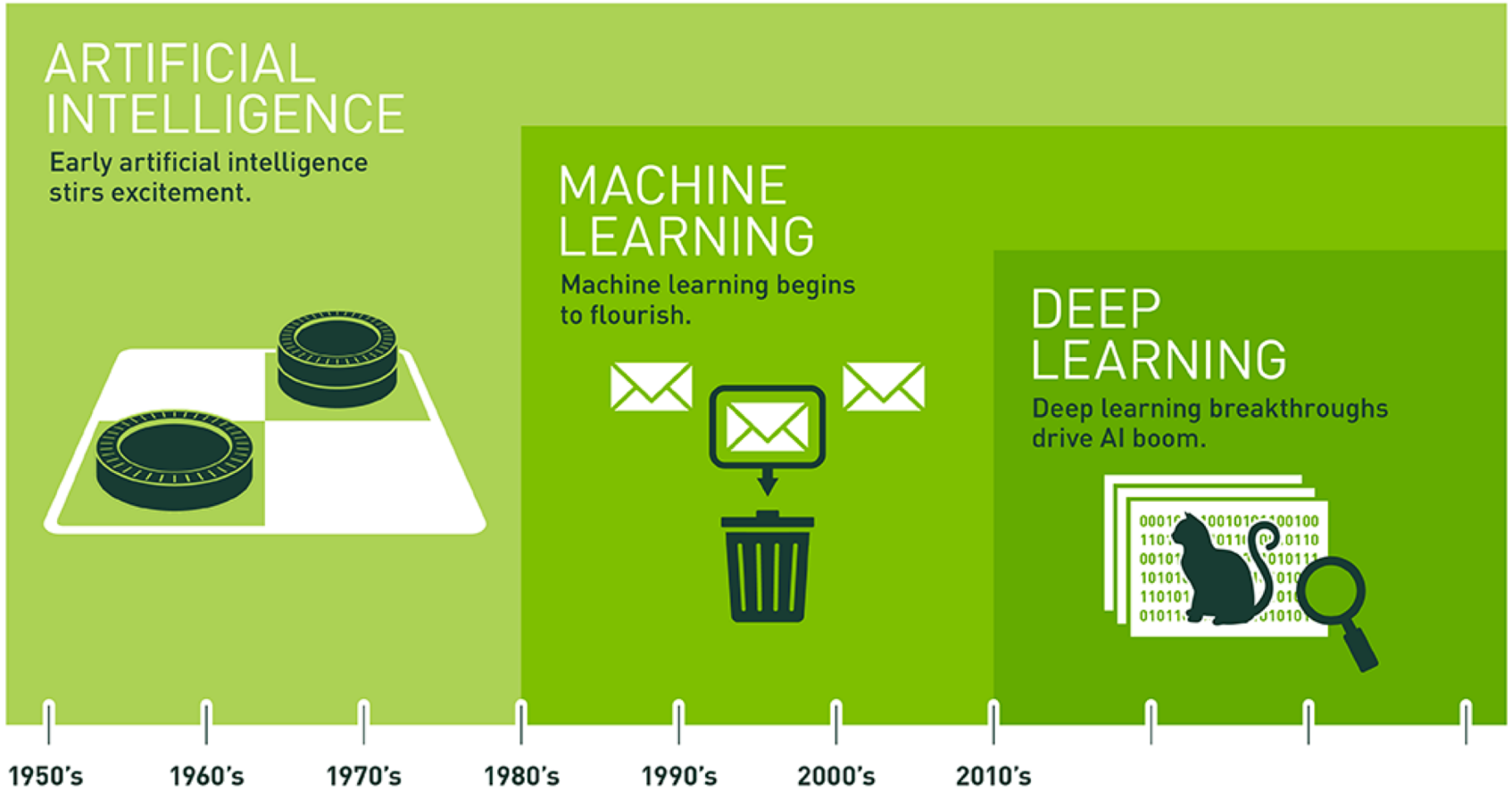
授課講師 孟璇
授課日期 108/5/2

北祥股份有限公司
PERSHING SYSTEMS CORPORATION

大綱

- 初探機器學習
- 機器學習實作

什麼是人工智慧



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

圖片來源: NVIDIA’s blog

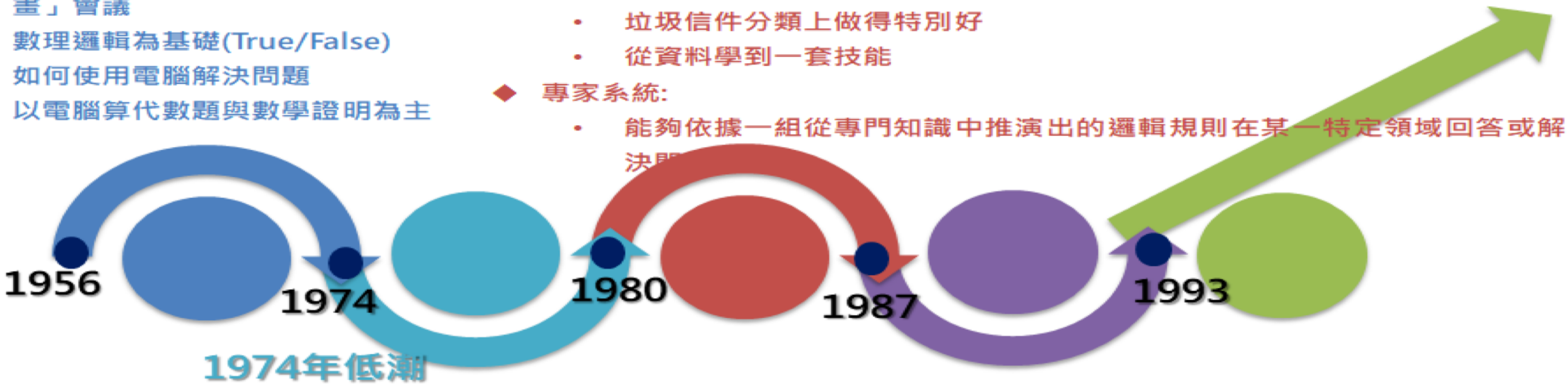
人工智慧大歷史

1956:人工智慧誕生

- 「達特矛斯夏季人工智慧研究計畫」會議
- 數理邏輯為基礎(True/False)
- 如何使用電腦解決問題
- 以電腦算代數題與數學證明為主

1980起 以機器學習帶起AI第二波

- 邏輯(0/1)→機率統計(量化)
  - 我們可以多確定這件事會發生
- 多層類神經網路失敗:
  - 1986，Hinton 等學者提出了反向傳播算法（Back Propagation），然而此方法受到梯度消失的問題，因此多層類神經網路熱潮消退。
- 淺層深度學習(SVM、決策樹等)興起:
  - 垃圾信件分類上做得特別好
  - 從資料學到一套技能
- 專家系統:
  - 能夠依據一組從專門知識中推演出的邏輯規則在某一特定領域回答或解決問



1974年低潮

- 人工智慧遇到瓶頸
- 計算機有限內存、處理速度低
  - 1965:電腦硬體指數成長(摩爾定律)
  - 1987-2017電腦成長100萬倍
- 無法回答人類不知道的問題

1987 第二次低潮

- Apple和IBM生產的台式機性能不斷提升
- 專家系統維護費用居高不下。它們難以升級，難以使用

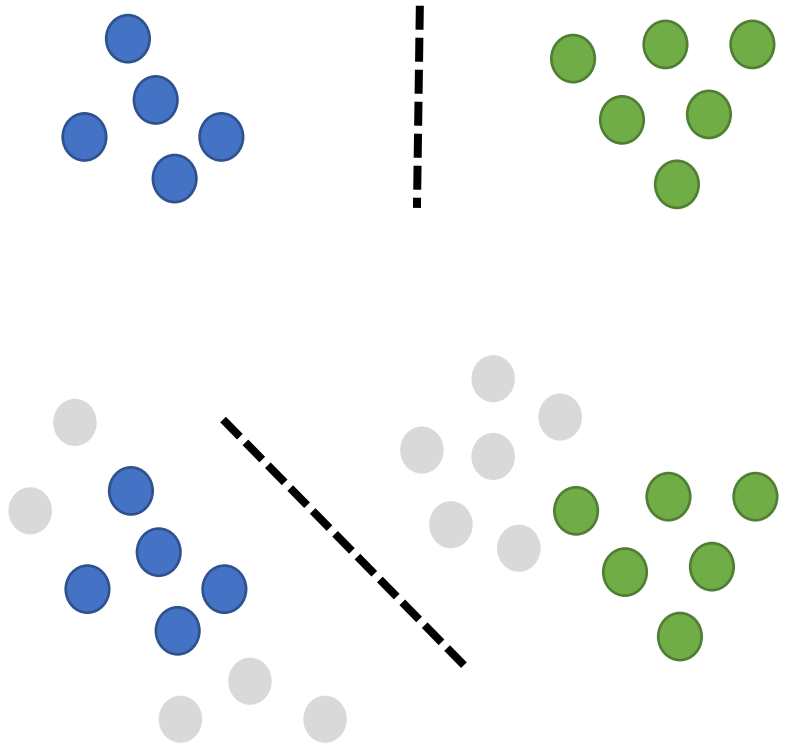
圖片來源:
https://medium.com/@suipichen/%E4%BA%BA%E5%B7%A5%E6%99%BA%E6%85%A7%E5%A4%A7%E6%AD%B7%E5%8F%B2-ffe46a350543

什麼是機器學習

- Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959).

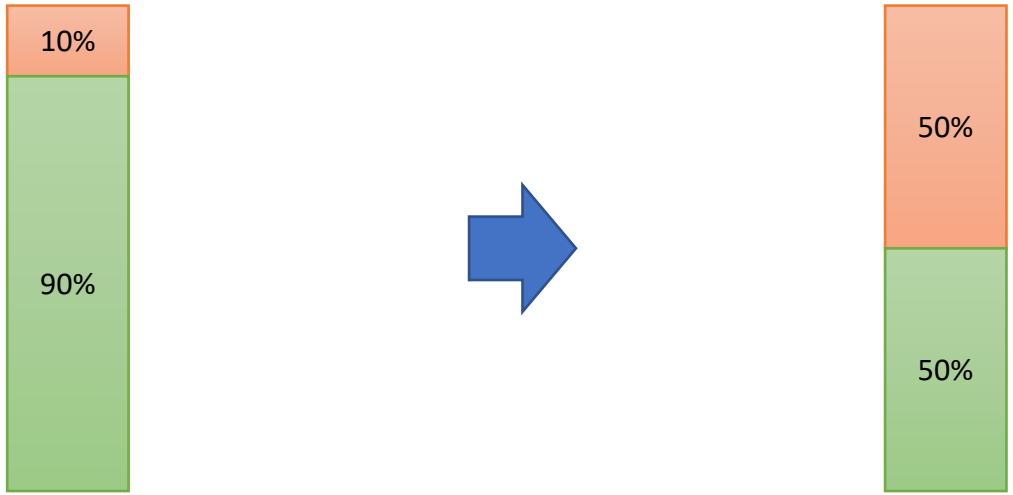
監督式 / 半監督式 / 非監督式

- 監督者的詞意
- 監督式 / 非監督式邊界模糊，常綜合使用



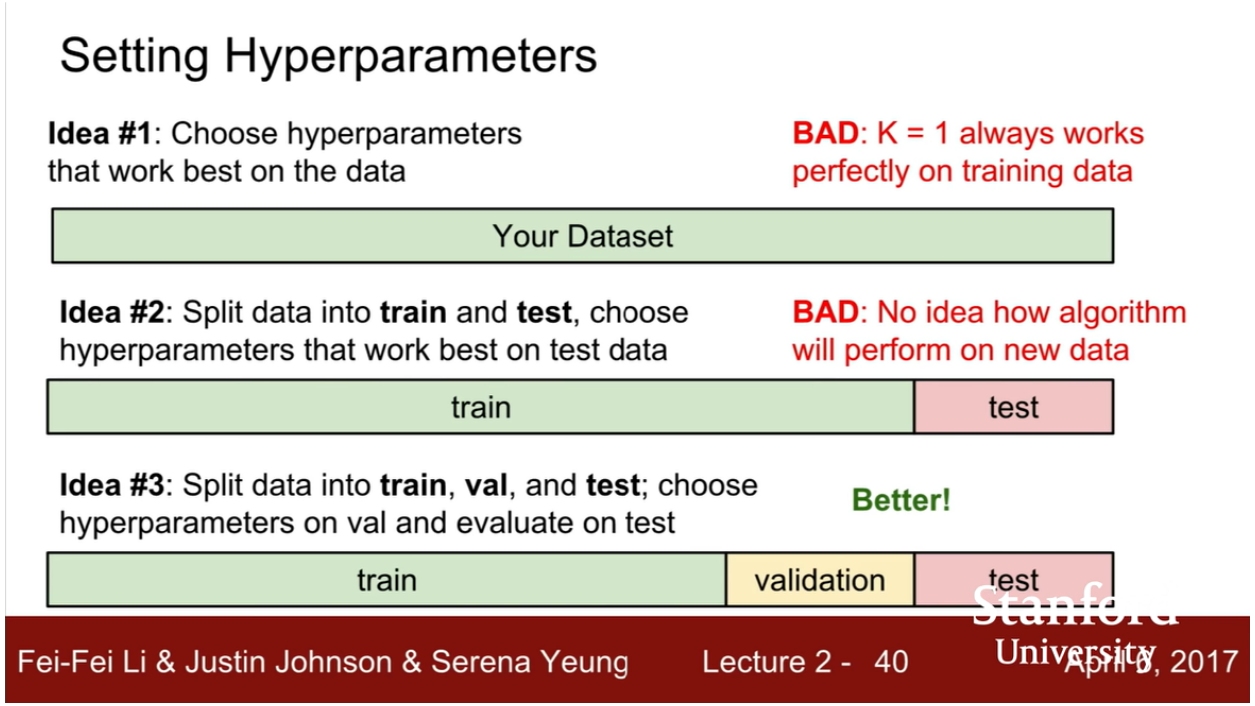
資料不平均

- 類別比例不均（如：90%為正常，10%為異常）
- 讓少數資料特徵顯現（抽樣、複製）



超參數 / 驗證集

- 超參數為控制學習演算法的設定
- 「validation set」為參數調整的參照

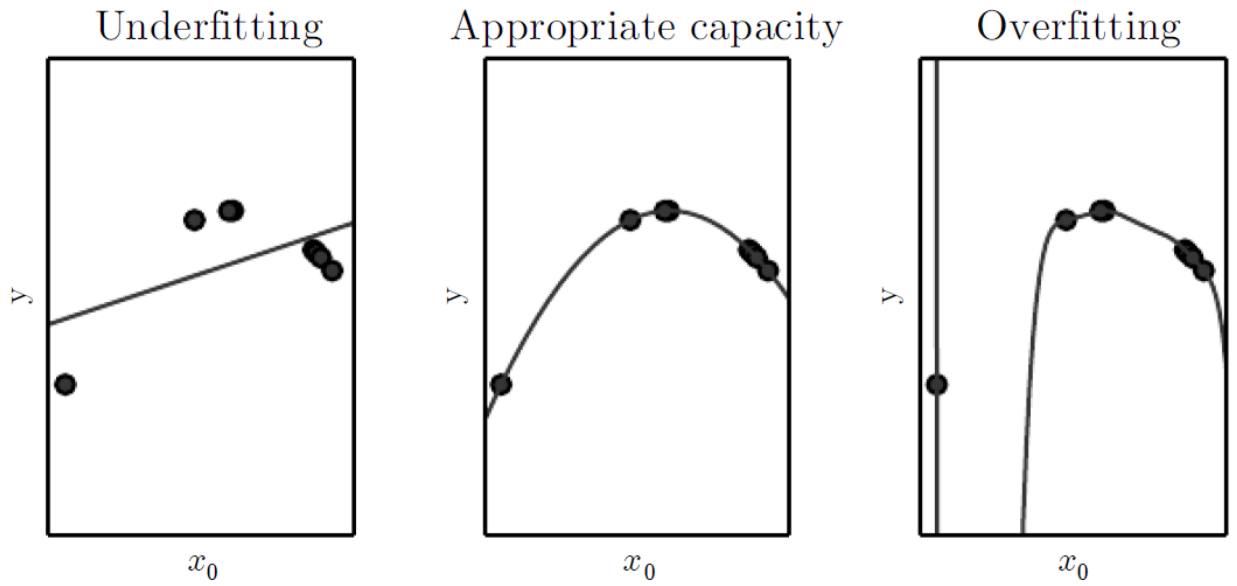


(圖片來源： Stanford的cs231n課程內容)

過度擬合 / 擬合不足

- 什麼是「capacity」
- 「training error」及「generalization error」

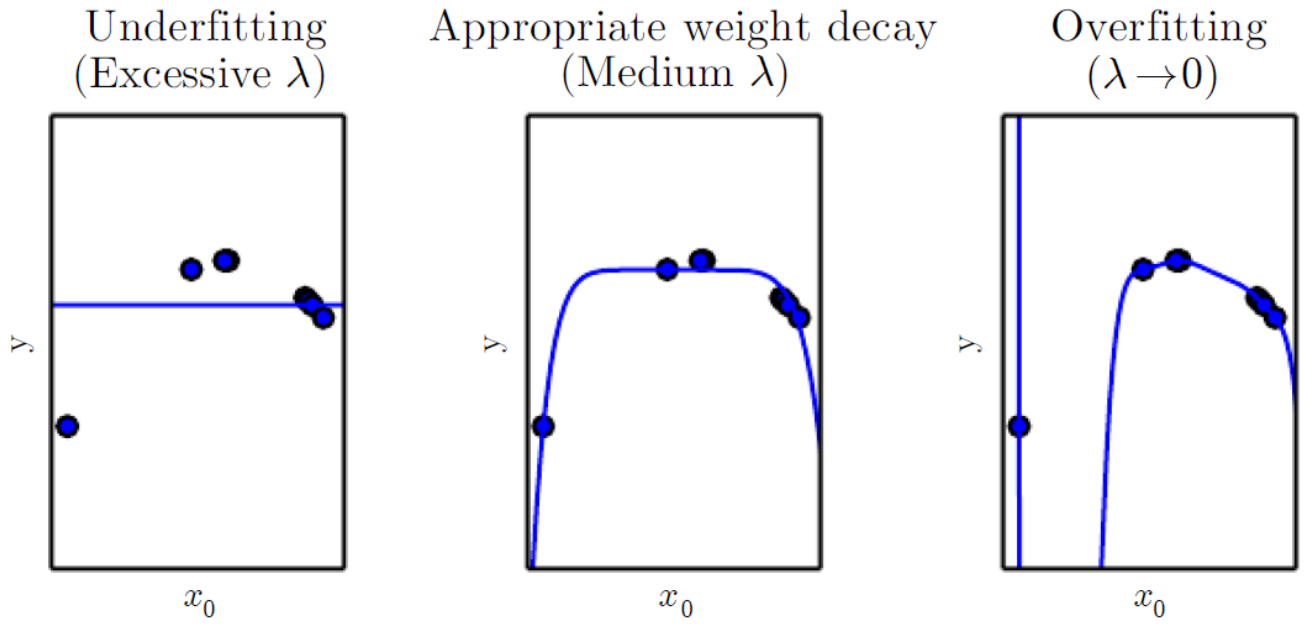
EX: 線性迴歸



(圖片來源： Ian Goodfellow的Deep Learning書籍)

正規化(Regularization)

- 減少「generalization error」
- 舉例：迴歸的weight decay



(圖片來源： Ian Goodfellow的Deep Learning書籍)

機器學習實作

資料背景&目的

- 背景
  - 1912年4月15日鐵達尼號在與冰山相撞後沈沒，在2224名乘客和機組人員中，造成1502人死亡。
- 目的
  - 預測乘客是否能存活
- 資料來源
  - Kaggle

欄位名稱

- Survival：生存與否
- Pclass：艙等
- Sex：性別
- Age：年齡
- Sibsp：父母＋小孩的數量
- Parch：兄弟姊妹＋丈夫妻子的數量
- Ticket：船票編號
- Fare：船票價格
- Cabin：艙船號碼
- Embarked：登船港口

資料預處理1

Import 所需函式庫

```
# pandas
import pandas as pd
from pandas import Series,DataFrame

# numpy, matplotlib, seaborn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

讀入資料檔

```
# get titanic & test csv files as a DataFrame
titanic_df = pd.read_csv("./titanic/train.csv")
test_df    = pd.read_csv("./titanic/test.csv")

titanic_df.head()
```

	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

觀察資料是否有空值

```
print (titanic_df.info())
print("------------------------------")
print (test_df.info())
```

```
<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)
memory usage: 36.0+ KB
```

丟掉不要的欄位

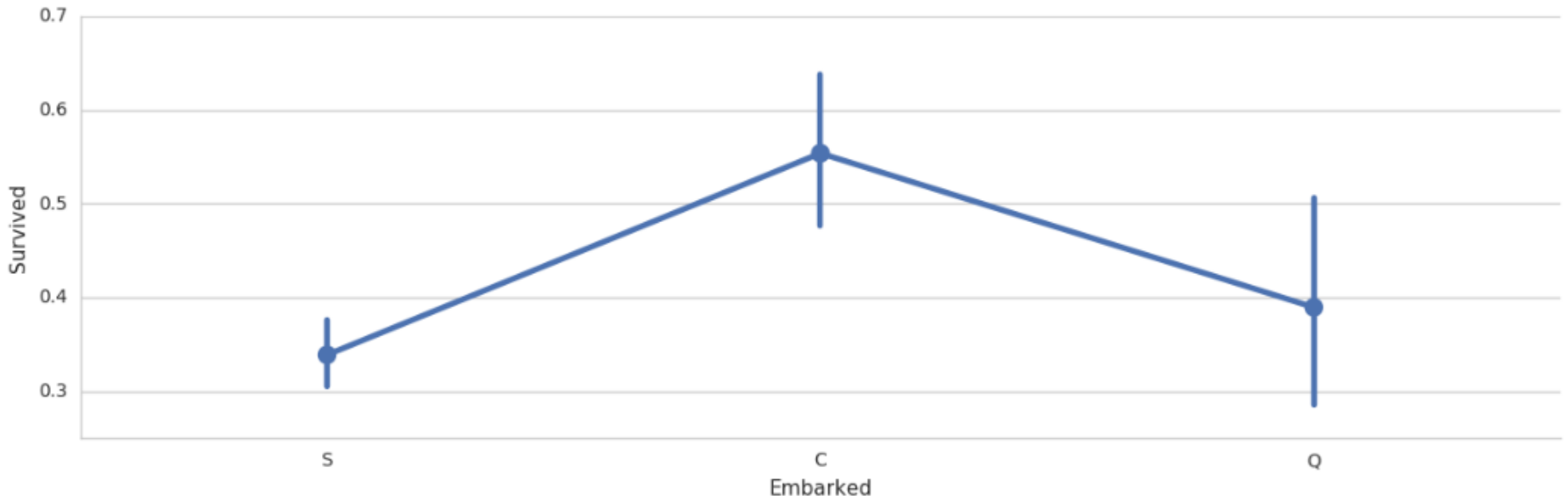
```
# drop unnecessary columns, these columns won't be useful in analysis and prediction
titanic_df = titanic_df.drop(['PassengerId','Name','Ticket'], axis=1)
test_df    = test_df.drop(['Name','Ticket'], axis=1)
```

```
print (titanic_df.columns)
print ('------------------------------')
print (test_df.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Fare', 'Cabin', 'Embarked'],
      dtype='object')
------------------------------
Index(['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Cabin', 'Embarked'],
      dtype='object')
```

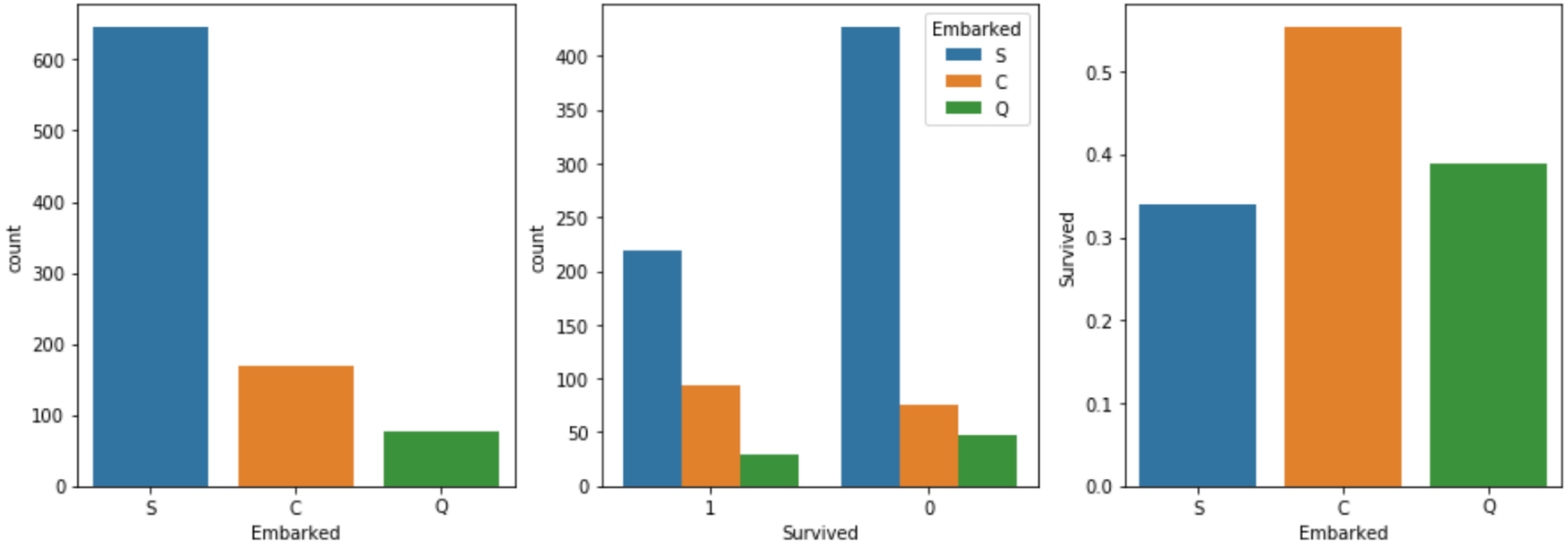
觀察資料 – Embarked

```
# Embarked

# only in titanic_df, fill the two missing values with the most occurred value, which is "S".
titanic_df["Embarked"] = titanic_df["Embarked"].fillna("S")

# plot
sns.factorplot('Embarked','Survived', data=titanic_df,size=4,aspect=3)
```



觀察資料 – Embarked

```
fig, (axis1,axis2,axis3) = plt.subplots(1,3,figsize=(15,5))
sns.countplot(x='Embarked', data=titanic_df, ax=axis1)
sns.countplot(x='Survived', hue="Embarked", data=titanic_df, order=[1,0], ax=axis2)

# group by embarked, and get the mean for survived passengers for each value in Embarked
embark_perc = titanic_df[["Embarked", "Survived"]].groupby(['Embarked'],as_index=False).mean()
sns.barplot(x='Embarked', y='Survived', data=embark_perc,order=['S','C','Q'],ax=axis3)
```



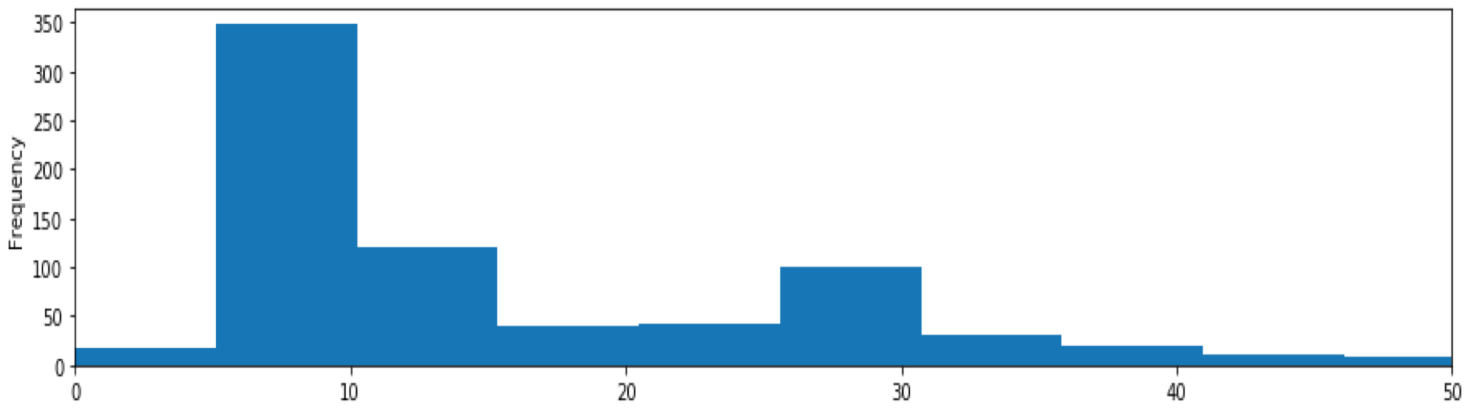
觀察資料 – Fare

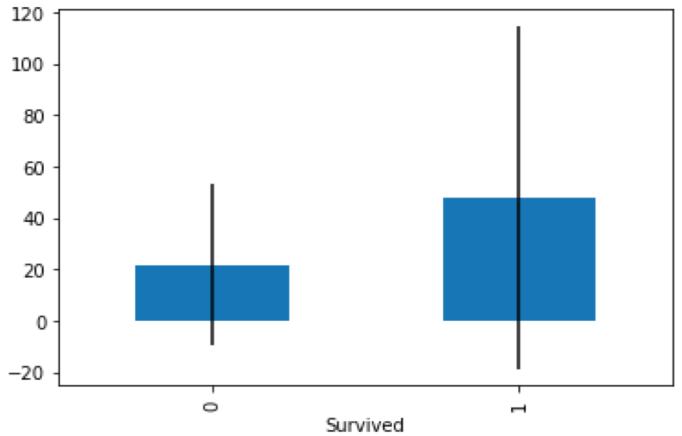
- 補缺失值
- 將值轉成整數

```
# only for test_df, since there is a missing "Fare" values
test_df["Fare"].fillna(test_df["Fare"].median(), inplace=True)

# convert from float to int
titanic_df['Fare'] = titanic_df['Fare'].astype(int)
test_df['Fare']    = test_df['Fare'].astype(int)
```

觀察資料 – Fare

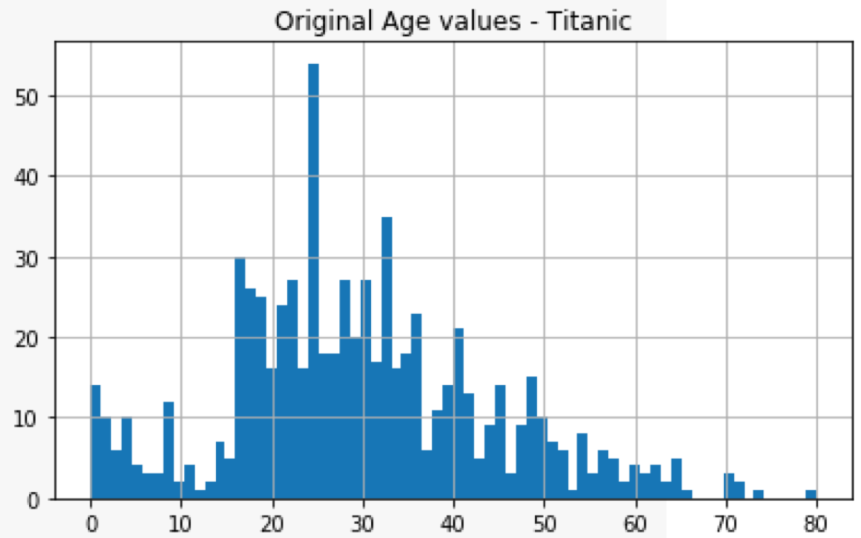
```
# get fare for survived & didn't survive passengers
fare_not_survived = titanic_df["Fare"][titanic_df["Survived"] == 0]
fare_survived     = titanic_df["Fare"][titanic_df["Survived"] == 1]

# get average and std for fare of survived/not survived passengers
avgerage_fare = pd.DataFrame([fare_not_survived.mean(), fare_survived.mean()])
std_fare      = pd.DataFrame([fare_not_survived.std(), fare_survived.std()])

# plot
titanic_df['Fare'].plot(kind='hist', figsize=(15,3),bins=100, xlim=(0,50))

avgerage_fare.index.names = std_fare.index.names = ["Survived"]
avgerage_fare.plot(yerr=std_fare,kind='bar',legend=False)
```





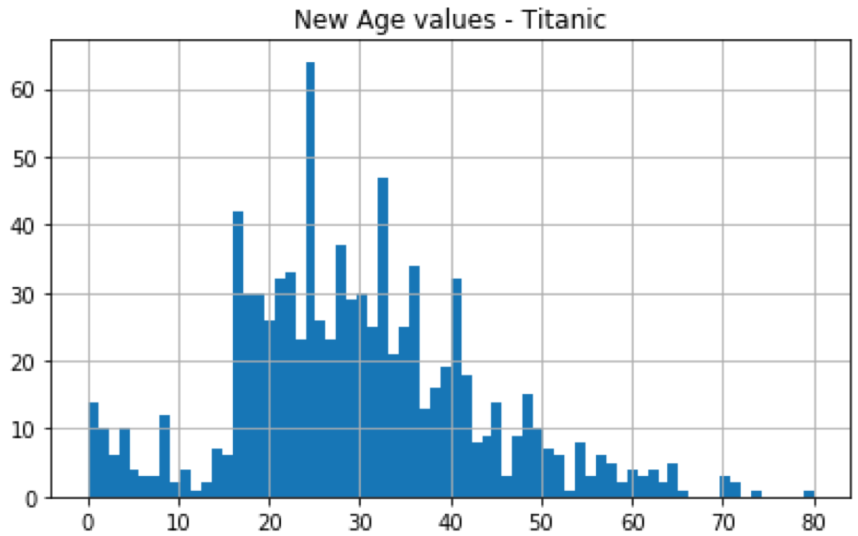
觀察資料 – Age

```
fig, (axis1,axis2) = plt.subplots(1,2,figsize=(15,4))
axis1.set_title('Original Age values - Titanic')
axis2.set_title('New Age values - Titanic')

# get average, std, and number of NaN values in titanic_df
average_age_titanic   = titanic_df["Age"].mean()
std_age_titanic       = titanic_df["Age"].std()
count_nan_age_titanic = titanic_df["Age"].isnull().sum()

# get average, std, and number of NaN values in test_df
average_age_test   = test_df["Age"].mean()
std_age_test       = test_df["Age"].std()
count_nan_age_test = test_df["Age"].isnull().sum()

# generate random numbers between (mean - std) & (mean + std)
rand_1 = np.random.randint(average_age_titanic - std_age_titanic, average_age_titanic + std_age_titanic,
                           size = count_nan_age_titanic)
rand_2 = np.random.randint(average_age_test - std_age_test, average_age_test + std_age_test,
                           size = count_nan_age_test)

# plot original Age values
# NOTE: drop all null values, and convert to int
titanic_df['Age'].dropna().astype(int).hist(bins=70, ax=axis1)
```



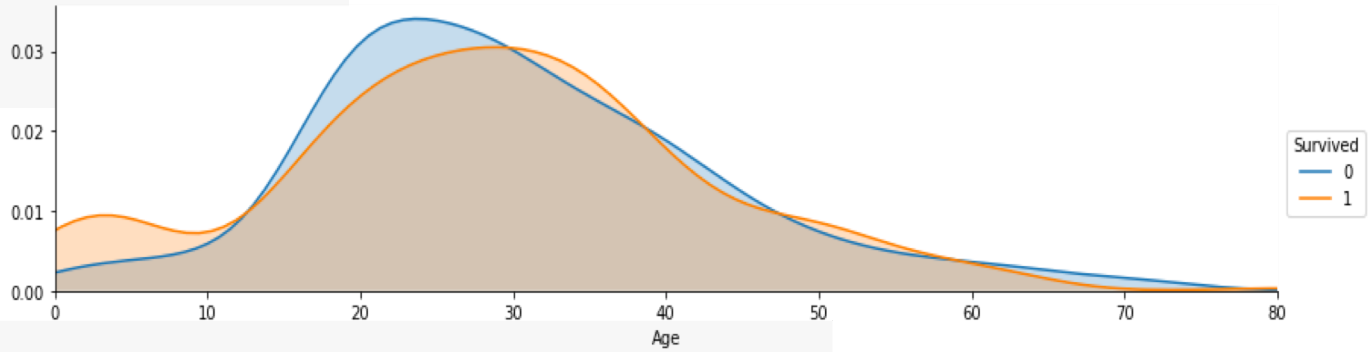
觀察資料 – Age

```
# fill NaN values in Age column with random values generated
titanic_df["Age"][np.isnan(titanic_df["Age"])] = rand_1
test_df["Age"][np.isnan(test_df["Age"])] = rand_2

# convert from float to int
titanic_df['Age'] = titanic_df['Age'].astype(int)
test_df['Age']    = test_df['Age'].astype(int)

# plot new Age Values
titanic_df['Age'].hist(bins=70, ax=axis2)
```

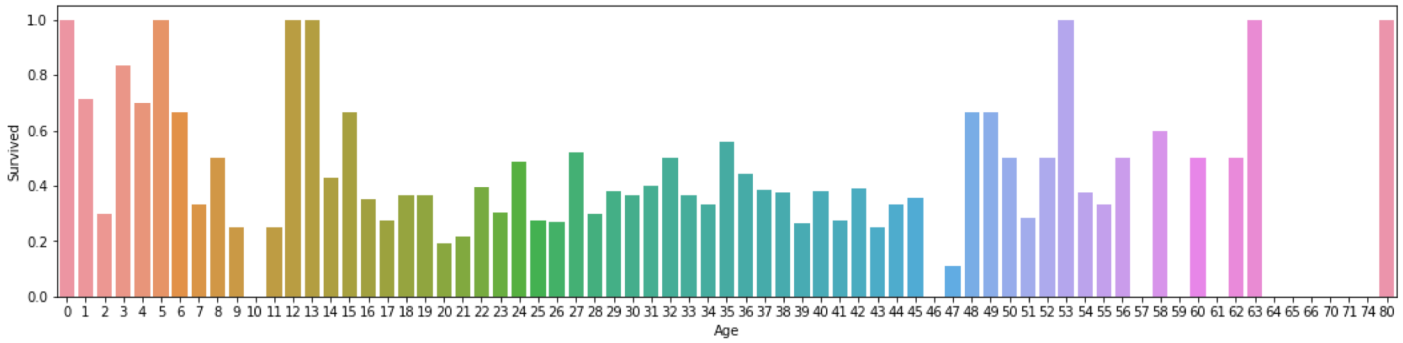


觀察資料 – Age

```
# peaks for survived/not survived passengers by their age
facet = sns.FacetGrid(titanic_df, hue="Survived",aspect=4)
facet.map(sns.kdeplot,'Age',shade= True)
facet.set(xlim=(0, titanic_df['Age'].max()))
facet.add_legend()
```




```
# average survived passengers by age
fig, axis1 = plt.subplots(1,1,figsize=(18,4))
average_age = titanic_df[["Age", "Survived"]].groupby(['Age'],as_index=False).mean()
sns.barplot(x='Age', y='Survived', data=average_age)
```



More information:
https://seaborn.pydata.org/generated/seaborn.FacetGrid.html

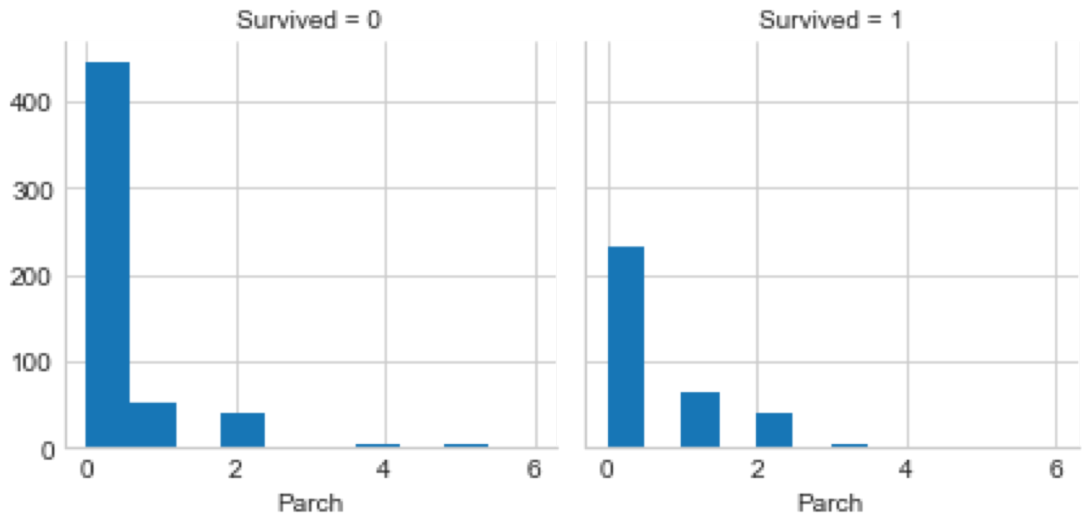
觀察資料 – Cabin

- 遺失值太多，考慮不納入分析
- 丟掉Cabin欄位

```
# Cabin
# It has a lot of NaN values, so it won't cause a remarkable impact on prediction
titanic_df.drop("Cabin",axis=1,inplace=True)
test_df.drop("Cabin",axis=1,inplace=True)
```

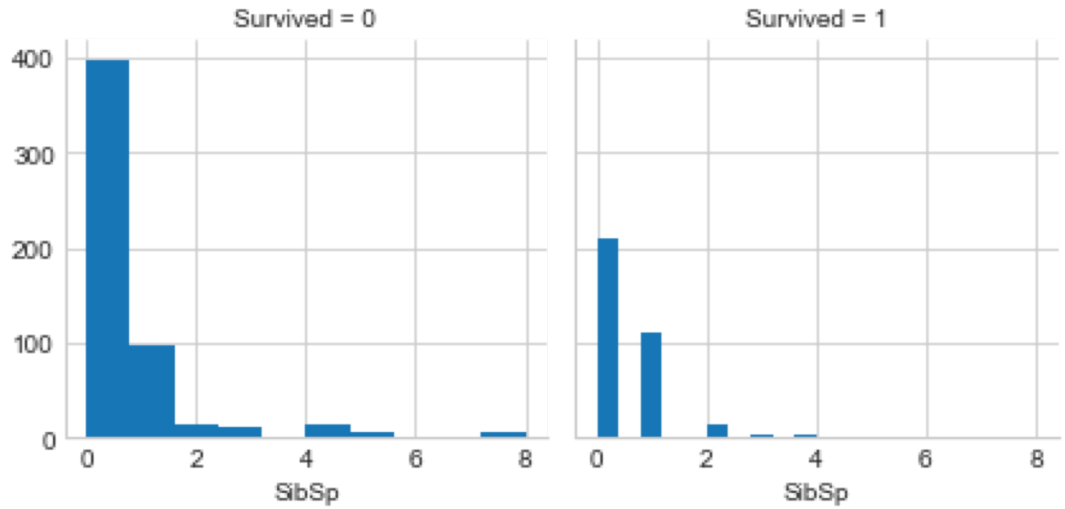
觀察資料 – Parch

```
g = sns.FacetGrid(titanic_df, col='Survived')
g.map(plt.hist, 'Parch')
```



觀察資料 – SibSp

```
g = sns.FacetGrid(titanic_df, col='Survived')
g.map(plt.hist, 'SibSp')
```



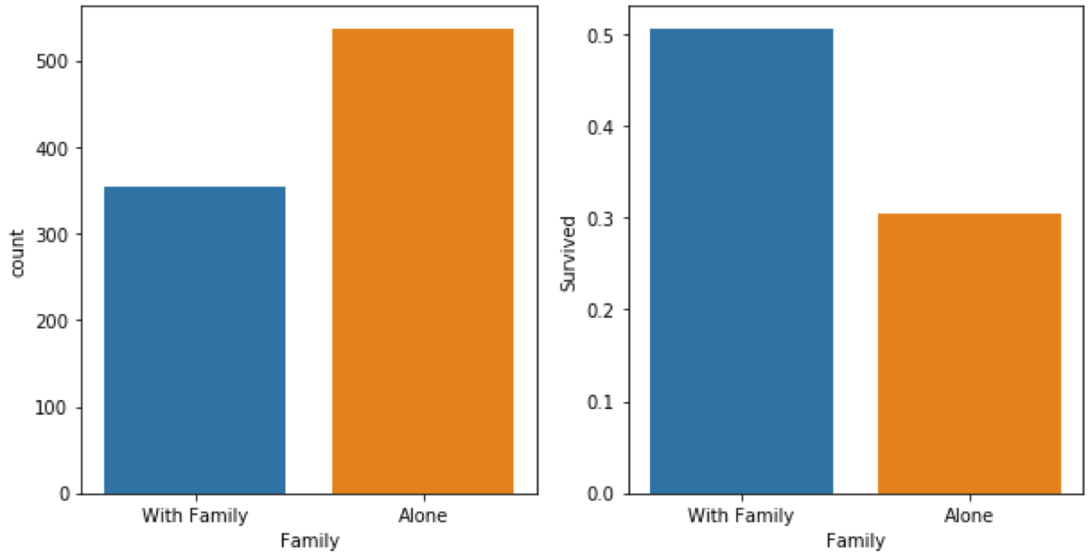
特徵工程 – Family

- 用一個欄位來表示是否為家庭

```
# Instead of having two columns Parch & SibSp,
# we can have only one column represent if the passenger had any family member aboard or not,
# Meaning, if having any family member(whether parent, brother, ...etc) will increase chances of Survival or not.
titanic_df['Family'] =  titanic_df["Parch"] + titanic_df["SibSp"]
titanic_df['Family'].loc[titanic_df['Family'] > 0] = 1
titanic_df['Family'].loc[titanic_df['Family'] == 0] = 0

test_df['Family'] =  test_df["Parch"] + test_df["SibSp"]
test_df['Family'].loc[test_df['Family'] > 0] = 1
test_df['Family'].loc[test_df['Family'] == 0] = 0

# drop Parch & SibSp
titanic_df = titanic_df.drop(['SibSp','Parch'], axis=1)
test_df    = test_df.drop(['SibSp','Parch'], axis=1)
```

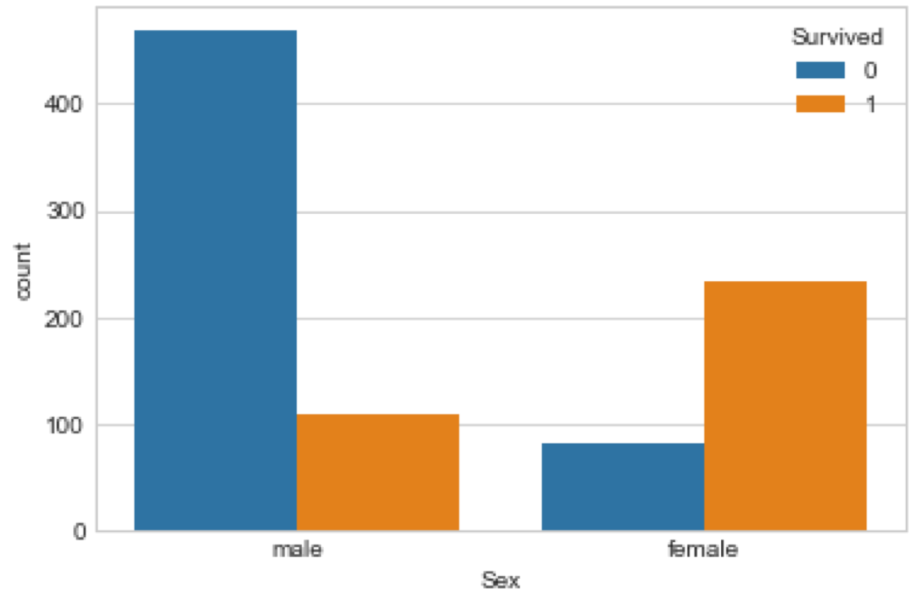
觀察資料 – Family

```
# plot
fig, (axis1,axis2) = plt.subplots(1,2,sharex=True,figsize=(10,5))

# sns.factorplot('Family',data=titanic_df,kind='count',ax=axis1)
sns.countplot(x='Family', data=titanic_df, order=[1,0], ax=axis1)

# average of survived for those who had/didn't have any family member
family_perc = titanic_df[["Family", "Survived"]].groupby(['Family'],as_index=False).mean()
sns.barplot(x='Family', y='Survived', data=family_perc, order=[1,0], ax=axis2)

axis1.set_xticklabels(["With Family","Alone"], rotation=0)
```



觀察資料 – Sex

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

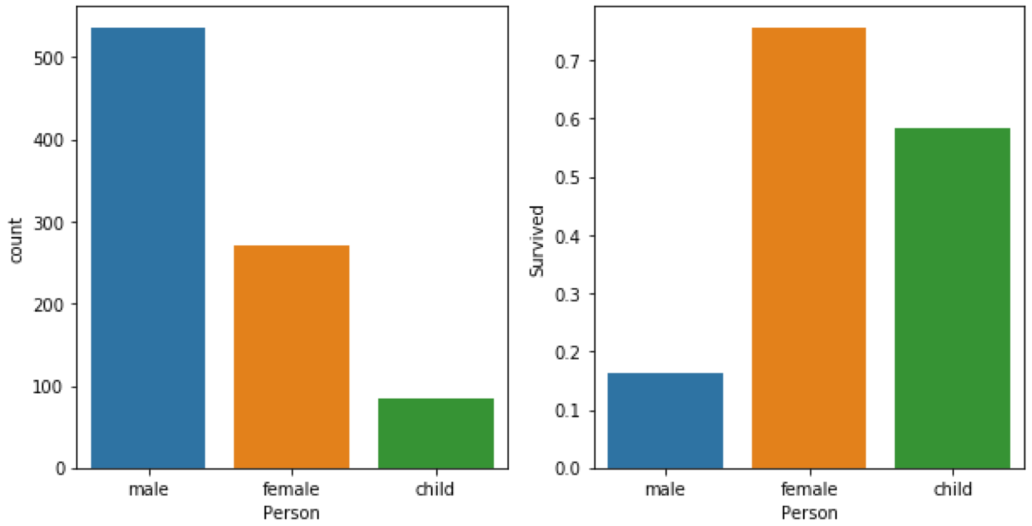


觀察資料 – Sex

```
# As we see, children(age < ~16) on aboard seem to have a high chances for Survival.
# So, we can classify passengers as males, females, and child
def get_person(passenger):
    age,sex = passenger
    return 'child' if age < 16 else sex

titanic_df['Person'] = titanic_df[['Age','Sex']].apply(get_person,axis=1)
test_df['Person']    = test_df[['Age','Sex']].apply(get_person,axis=1)

# No need to use Sex column since we created Person column
titanic_df.drop(['Sex'],axis=1,inplace=True)
test_df.drop(['Sex'],axis=1,inplace=True)

# create dummy variables for Person column, & drop Male as it has the lowest average of survived passengers
person_dummies_titanic  = pd.get_dummies(titanic_df['Person'])
person_dummies_titanic.columns = ['Child','Female','Male']
person_dummies_titanic.drop(['Male'], axis=1, inplace=True)

person_dummies_test  = pd.get_dummies(test_df['Person'])
person_dummies_test.columns = ['Child','Female','Male']
person_dummies_test.drop(['Male'], axis=1, inplace=True)

titanic_df = titanic_df.join(person_dummies_titanic)
test_df    = test_df.join(person_dummies_test)
```

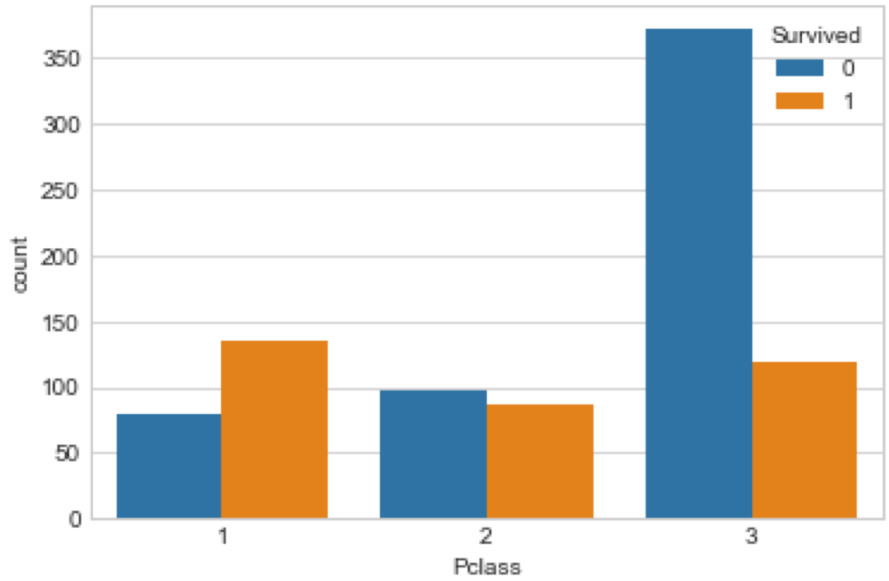
觀察資料 – Sex

```
fig, (axis1,axis2) = plt.subplots(1,2,figsize=(10,5))

# sns.factorplot('Person',data=titanic_df,kind='count',ax=axis1)
sns.countplot(x='Person', data=titanic_df, ax=axis1)

# average of survived for each Person(male, female, or child)
person_perc = titanic_df[["Person", "Survived"]].groupby(['Person'],as_index=False).mean()
sns.barplot(x='Person', y='Survived', data=person_perc, ax=axis2, order=['male','female','child'])

titanic_df.drop(['Person'],axis=1,inplace=True)
test_df.drop(['Person'],axis=1,inplace=True)
```



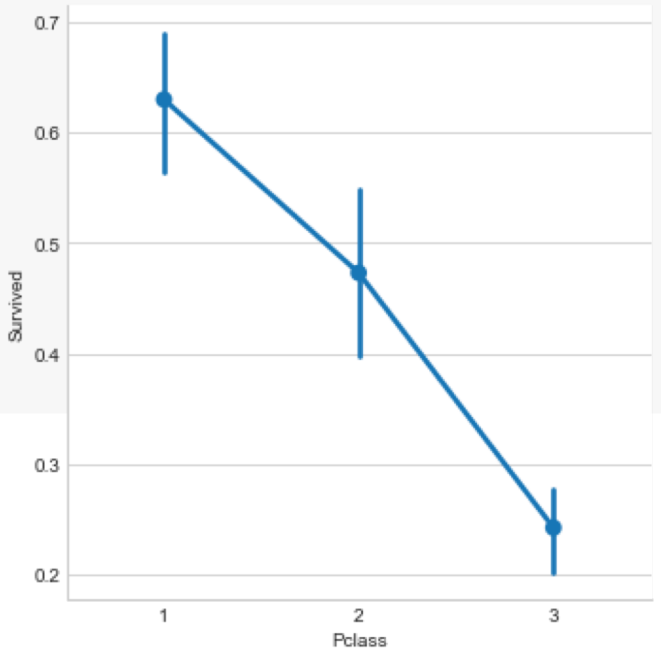
觀察資料 — Pclass

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



觀察資料 — Pclass

```
# sns.factorplot('Pclass',data=titanic_df,kind='count',order=[1,2,3])
sns.factorplot('Pclass','Survived',order=[1,2,3], data=titanic_df,size=5)

# create dummy variables for Pclass column, & drop 3rd class as it has the lowest average of survived passengers
pclass_dummies_titanic  = pd.get_dummies(titanic_df['Pclass'])
pclass_dummies_titanic.columns = ['Class_1','Class_2','Class_3']
pclass_dummies_titanic.drop(['Class_3'], axis=1, inplace=True)

pclass_dummies_test  = pd.get_dummies(test_df['Pclass'])
pclass_dummies_test.columns = ['Class_1','Class_2','Class_3']
pclass_dummies_test.drop(['Class_3'], axis=1, inplace=True)

titanic_df.drop(['Pclass'],axis=1,inplace=True)
test_df.drop(['Pclass'],axis=1,inplace=True)

titanic_df = titanic_df.join(pclass_dummies_titanic)
test_df    = test_df.join(pclass_dummies_test)
```



建立模型1

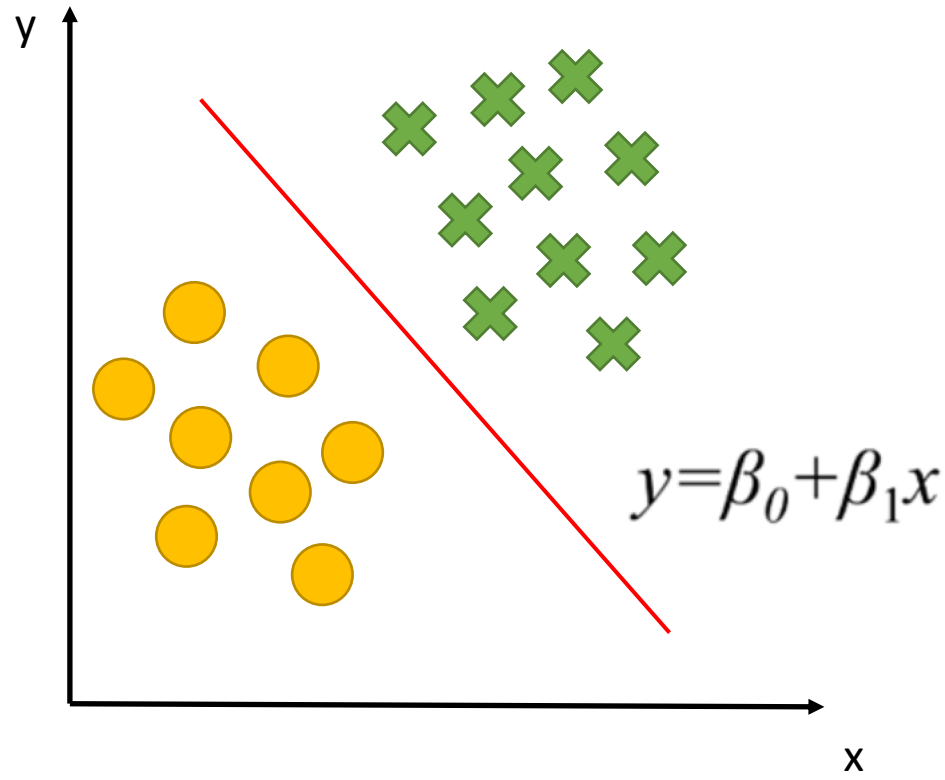
定義訓練跟測試集

- X 是什麼？
- Y 是什麼？

```
# define training and testing sets

X_train = titanic_df.drop("Survived",axis=1)
Y_train = titanic_df["Survived"]
X_test  = test_df.drop("PassengerId",axis=1).copy()
```

分類模型

- 處裡線性和二元分類的問題



邏輯斯回歸

```
# Logistic Regression

logreg = LogisticRegression()
```
呼叫 LogisticRegression()

```
logreg.fit(X_train, Y_train)
```
根據給定的訓練數據擬合模型

```
Y_pred = logreg.predict(X_test)
```
預測X數據的分類結果

```
logreg.score(X_train, Y_train)
```
評估模型

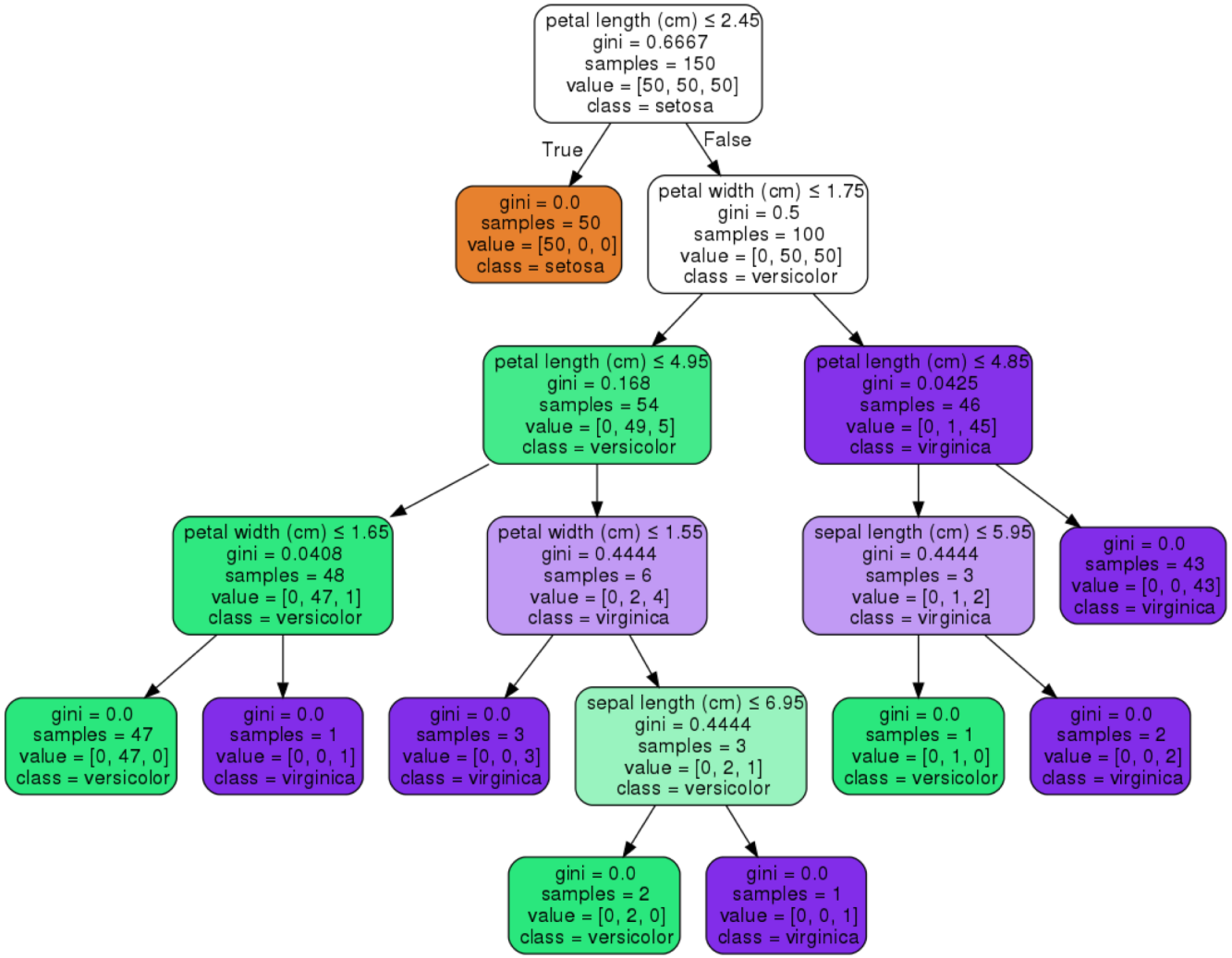
```
0.8058361391694725
```

Support Vector Classification (SVC)

```
# Support Vector Machines

svc = SVC()

svc.fit(X_train, Y_train)

Y_pred = svc.predict(X_test)

svc.score(X_train, Y_train)
```

```
0.8731762065095399
```

Decision tree



Random Forests

```
# Random Forests

random_forest = RandomForestClassifier(n_estimators=100)

random_forest.fit(X_train, Y_train)

Y_pred = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
```

0.9674523007856342

獲取迴歸係數

```
# get Correlation Coefficient for each feature using Logistic Regression
coeff_df = pd.DataFrame(titanic_df.columns.delete([0,1]))
coeff_df.columns = ['Features']
coeff_df["Coefficient Estimate"] = pd.Series(logreg.coef_[0])

# preview
coeff_df
```

	Features	Coefficient Estimate
0	Age	-0.023379
1	Fare	0.000835
2	C	0.594886
3	Q	0.280388
4	Family	-0.217172
5	Child	1.753677
6	Female	2.760550
7	Class_1	1.998658
8	Class_2	1.118209

資料預處理2

Import 所需函式庫

```
import pandas as pd
import numpy as np
import re

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls

import sklearn
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier)
from sklearn.svm import SVC
import xgboost as xgb

from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

讀入資料檔

```
train = pd.read_csv('./titanic/train.csv')
test = pd.read_csv('./titanic/test.csv')

combine = train.append(test)
combine
```

為了整理資料方便，先把訓練資料及測試資料合併

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket
0	22.0	NaN	S	7.2500	Braund, Mr. Owen Harris	0	1	3	male	1	0.0	A/5 21171
1	38.0	C85	C	71.2833	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	2	1	female	1	1.0	PC 17599
2	26.0	NaN	S	7.9250	Heikkinen, Miss. Laina	0	3	3	female	0	1.0	STON/O2. 3101282
3	35.0	C123	S	53.1000	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	4	1	female	1	1.0	113803
4	35.0	NaN	S	8.0500	Allen, Mr. William Henry	0	5	3	male	0	0.0	373450
5	NaN	NaN	Q	8.4583	Moran, Mr. James	0	6	3	male	0	0.0	330877

觀察資料

```
print('總資料筆數：',len(combine))
print('缺值：')
print(combine.isna().sum())
```

```
總資料筆數： 1309
缺值：
Age             263
Cabin          1014
Embarked          2
Fare              1
Name              0
Parch             0
PassengerId       0
Pclass            0
Sex               0
SibSp             0
Survived        418
Ticket            0
dtype: int64
```

→ 船艙幾乎都缺值，補值效用不高

→ 有418筆為測試資料，沒有值合理

***要處理缺值的欄位為Age、Embarked、Fare

處理遺失值

```
combine['Embarked'] = combine['Embarked'].fillna('S')
```
→ 將遺失值令為最多的那個類別

```
combine['Fare'] = combine['Fare'].fillna(combine['Fare'].median())
```
→ 將遺失值令為中位數

```
age_avg = combine['Age'].mean()
age_std = combine['Age'].std()
age_null_count = combine['Age'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=age_null_count)
combine['Age'][np.isnan(combine['Age'])] = age_null_random_list
combine['Age'] = combine['Age'].astype(int)
```

年齡遺失值比較多，因此不直接補中位數

補值方式為：在均勻分配中隨機抽取整數（上下界為年齡平均加減一倍標準差）

檢查處理缺值後的結果

```
print('總資料筆數：',len(combine))
print('缺值：')
print(combine.isna().sum())
```

```
總資料筆數： 1309
缺值：
Age             0
Cabin        1014
Embarked        0
Fare            0
Name            0
Parch           0
PassengerId     0
Pclass          0
Sex             0
SibSp           0
Survived      418
Ticket          0
dtype: int64
```

Age、Embarked、Fare已補值完畢

特徵工程 – Has_Cabin

```
combine['Has_Cabin'] = combine["Cabin"].apply(lambda x: 0 if type(x) == float else 1)

combine[['Cabin','Has_Cabin']].head()
```

	Cabin	Has_Cabin
0	NaN	0
1	C85	1
2	NaN	0
3	C123	1
4	NaN	0

有船艙的標示1，沒有的標示0
(有船艙登記可能代表社會地位不同)

特徵工程 – Name_length

```
combine['Name_length'] = combine['Name'].apply(len)

combine[['Name','Name_length']].head()
```

	Name	Name_length
0	Braund, Mr. Owen Harris	23
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	51
2	Heikkinen, Miss. Laina	22
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	44
4	Allen, Mr. William Henry	24

將名字的長度當作一個特徵
(名字的長度可能意味著是不是古老的家族，較有權勢)

特徵工程 - FamilySize

```
combine['FamilySize'] = combine['SibSp'] + combine['Parch'] + 1

combine[['SibSp','Parch', 'FamilySize']].head()
```

	SibSp	Parch	FamilySize
0	1	0	2
1	1	0	2
2	0	0	1
3	1	0	2
4	0	0	1

計算船上的家族人數
(家族人數在逃生時可能為重要因素)

特徵工程 - IsAlone

```
combine['IsAlone'] = 0
combine.loc[combine['FamilySize'] == 1, 'IsAlone'] = 1

combine[['FamilySize', 'IsAlone']].head()
```

	FamilySize	IsAlone
0	2	0
1	2	0
2	1	1
3	2	0
4	1	1

計算是不是獨自登船
(理由同家族人數，在逃生時可能為重要因素)

特徵工程 - Title

```
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    if title_search:
        return title_search.group(1)
    return ""

combine['Title'] = combine['Name'].apply(get_title)
combine[['Name','Title']].head()
```

	Name	Title
0	Braund, Mr. Owen Harris	Mr
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	Mrs
2	Heikkinen, Miss. Laina	Miss
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	Mrs
4	Allen, Mr. William Henry	Mr

擷取名字中的稱謂

特徵工程 - Title

```
combine['Title'].value_counts()
```

```
Mr          757
Miss        260
Mrs         197
Master       61
Dr            8
Rev           8
Col           4
Major         2
Mlle          2
Ms            2
Capt          1
Jonkheer      1
Countess      1
Don           1
Dona          1
Sir           1
Mme           1
Lady          1
Name: Title, dtype: int64
```

觀察擷取出的稱謂

特徵工程 - Title

```
combine['Title'] = combine['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr', 'Major',
                                             'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
combine['Title'] = combine['Title'].replace('Mlle', 'Miss')
combine['Title'] = combine['Title'].replace('Ms', 'Miss')
combine['Title'] = combine['Title'].replace('Mme', 'Mrs')
```

```
combine['Title'].value_counts()
```

```
Mr        757
Miss      264
Mrs       198
Master     61
Rare       29
Name: Title, dtype: int64
```

整理完後的稱謂結果

特徵工程

將類別數字化，並將連續型數值轉換為區間

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


title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
combine['Title'] = combine['Title'].map(title_mapping)
combine['Title'] = combine['Title'].fillna(0)


combine['Embarked'] = combine['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)


combine.loc[ combine['Fare'] <= 8, 'Fare'] = 0
combine.loc[(combine['Fare'] > 8) & (combine['Fare'] <= 14), 'Fare'] = 1
combine.loc[(combine['Fare'] > 14) & (combine['Fare'] <= 31), 'Fare'] = 2
combine.loc[ combine['Fare'] > 31, 'Fare'] = 3
combine['Fare'] = combine['Fare'].astype(int)


combine.loc[ combine['Age'] <= 16, 'Age'] = 0
combine.loc[(combine['Age'] > 16) & (combine['Age'] <= 32), 'Age'] = 1
combine.loc[(combine['Age'] > 32) & (combine['Age'] <= 48), 'Age'] = 2
combine.loc[(combine['Age'] > 48) & (combine['Age'] <= 64), 'Age'] = 3
combine.loc[ combine['Age'] > 64, 'Age'] = 4

combine.head()
```

	Age	Cabin	Embarked	Fare	Name	Parch	PassengerId	Pclass	Sex	SibSp	Survived	Ticket	Has_Cabin	Name_length	FamilySize	IsAlone	Title
0	1	NaN	0	0	Braund, Mr. Owen Harris	0	1	3	1	1	0.0	A/5 21171	0	23	2	0	1
1	2	C85	1	3	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	2	1	0	1	1.0	PC 17599	1	51	2	0	3
2	1	NaN	0	0	Heikkinen, Miss. Laina	0	3	3	0	0	1.0	STON/O2. 3101282	0	22	1	1	2
3	2	C123	0	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	4	1	0	1	1.0	113803	1	44	2	0	3
4	2	NaN	0	1	Allen, Mr. William	0	5	3	1	0	0.0	373450	0	24	1	1	1

特徵工程

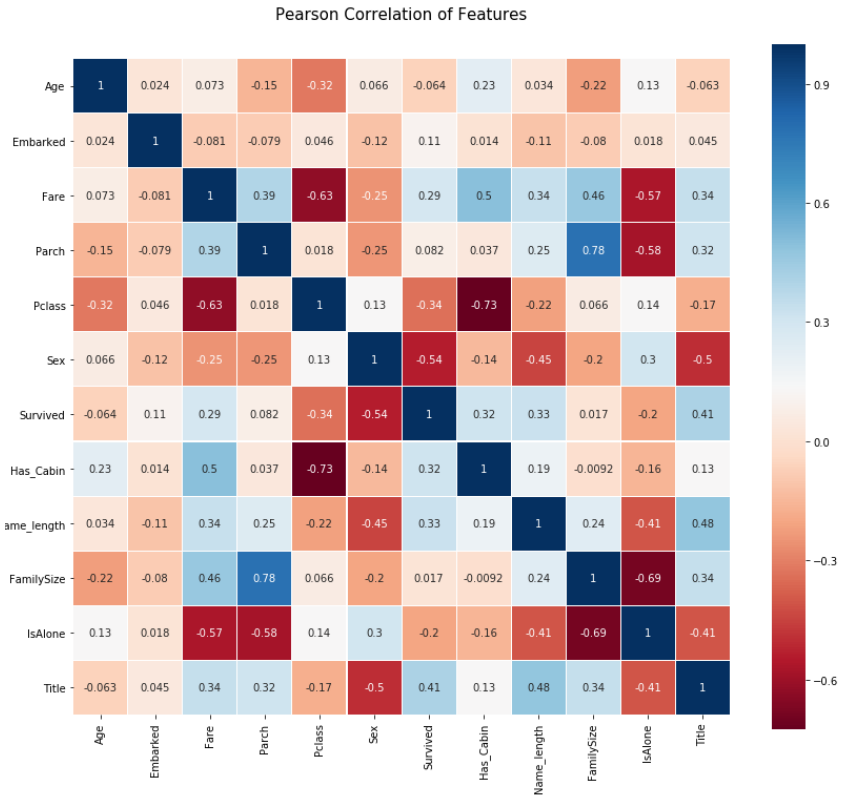
去除掉不需要的欄位，並還原為訓練資料及測試資料

```
drop_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
combine = combine.drop(drop_elements, axis = 1)

train = combine[combine['Survived'].isna()==False]
test = combine[combine['Survived'].isna()==True]

train.head()
```

	Age	Embarked	Fare	Parch	Pclass	Sex	Survived	Has_Cabin	Name_length	FamilySize	IsAlone	Title
0	1	0	0	0	3	1	0.0	0	23	2	0	1
1	2	1	3	0	1	0	1.0	1	51	2	0	3
2	1	0	0	0	3	0	1.0	0	22	1	1	2
3	2	0	3	0	1	0	1.0	1	44	2	0	3
4	2	0	1	0	3	1	0.0	0	24	1	1	1

特徵視覺化觀察

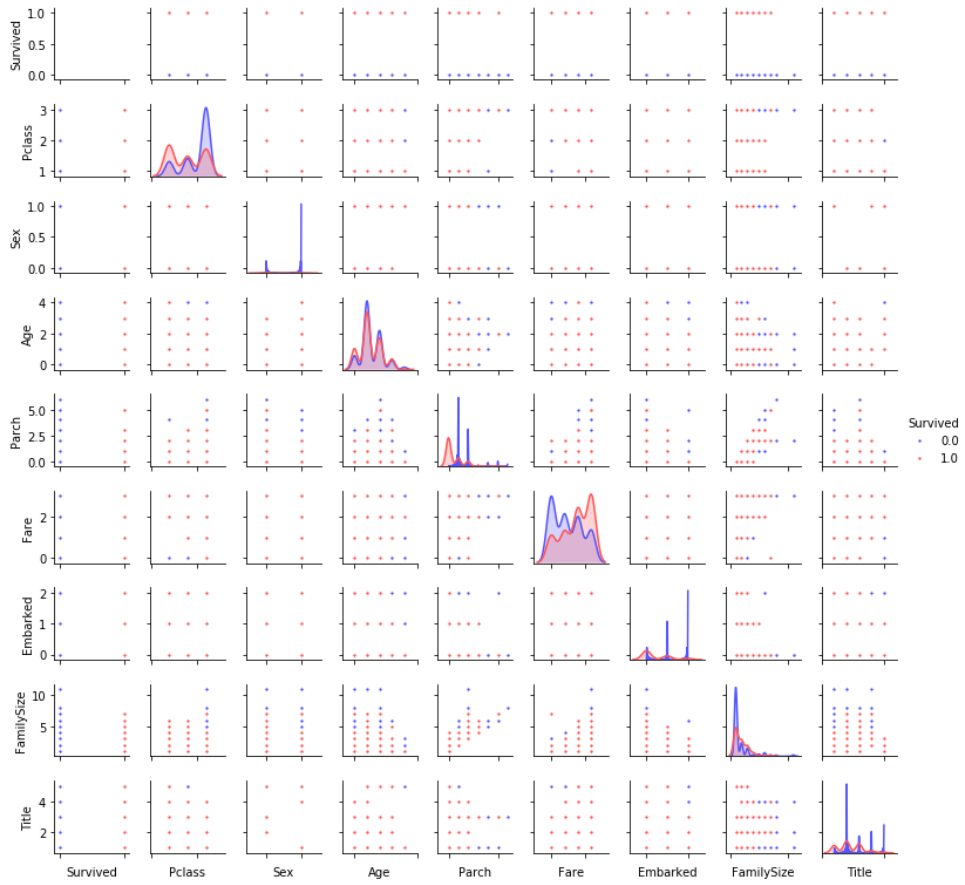
```
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
```



視覺化呈現特徵間的關係

特徵視覺化觀察

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



觀察特徵間與存活與否的關係

模型訓練前準備

```
ntrain = train.shape[0]
ntest = test.shape[0]
SEED = 0
NFOLDS = 5
kf = KFold(n_splits=NFOLDS, random_state=SEED)


class SklearnHelper(object):
    def __init__(self, clf, seed=0, params=None):
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def fit(self,x,y):
        return self.clf.fit(x,y)
```

訓練資料及測試資料的筆數

因未來模型很多但方法都一樣，為了方便管理而設立class

建立模型2

模型訓練前準備

```
def get_oof(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, y_train)):
        x_tr = x_train[train_index]
        y_tr = y_train[train_index]
        x_te = x_train[test_index]

        clf.train(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)
```

創建out-of-fold結果，避免over fitting

設置模型超參數

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

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

ada_params = {
    'n_estimators': 500,
    'learning_rate' : 0.75
}

gb_params = {
    'n_estimators': 500,
    'max_depth': 5,
    'min_samples_leaf': 2,
    'verbose': 0
}

svc_params = {
    'kernel' : 'linear',
    'C' : 0.025
    }
```

將五個模型的超參數建立成字典型式方便調用管理

模型訓練

```
rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf_params)
et = SklearnHelper(clf=ExtraTreesClassifier, seed=SEED, params=et_params)
ada = SklearnHelper(clf=AdaBoostClassifier, seed=SEED, params=ada_params)
gb = SklearnHelper(clf=GradientBoostingClassifier, seed=SEED, params=gb_params)
svc = SklearnHelper(clf=SVC, seed=SEED, params=svc_params)
```

建立模型

```
y_train = train['Survived'].ravel()
train = train.drop(['Survived'], axis=1)
test = test.drop(['Survived'], axis=1)
x_train = train.values
x_test = test.values
```

準備輸入模型的資料

```
et_oof_train, et_oof_test = get_oof(et, x_train, y_train, x_test)
rf_oof_train, rf_oof_test = get_oof(rf,x_train, y_train, x_test)
ada_oof_train, ada_oof_test = get_oof(ada, x_train, y_train, x_test)
gb_oof_train, gb_oof_test = get_oof(gb,x_train, y_train, x_test)
svc_oof_train, svc_oof_test = get_oof(svc,x_train, y_train, x_test)
```

訓練模型

特徵重要性探討

```
rf_feature = rf.fit(x_train,y_train).feature_importances_
et_feature = et.fit(x_train,y_train).feature_importances_
ada_feature = ada.fit(x_train,y_train).feature_importances_
gb_feature = gb.fit(x_train,y_train).feature_importances_
```

計算特徵重要性

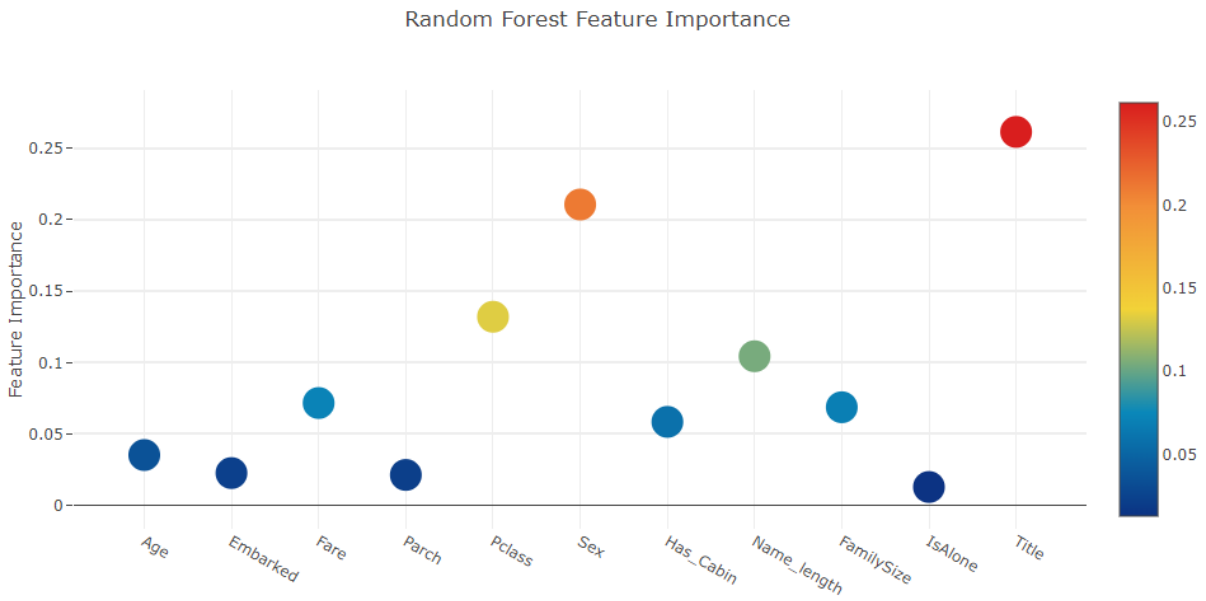
```
cols = train.columns.values
feature_dataframe = pd.DataFrame( {'features': cols,
     'Random Forest feature importances': rf_feature,
     'Extra Trees  feature importances': et_feature,
      'AdaBoost feature importances': ada_feature,
    'Gradient Boost feature importances': gb_feature
    })

feature_dataframe
```

將特徵重要性彙整成表格

	features	Random Forest feature importances	Extra Trees feature importances	AdaBoost feature importances	Gradient Boost feature importances
0	Age	0.035300	0.030184	0.012	0.046591
1	Embarked	0.022676	0.028065	0.008	0.024412
2	Fare	0.071709	0.053145	0.016	0.057525
3	Parch	0.021370	0.016403	0.064	0.010853
4	Pclass	0.132096	0.122301	0.030	0.087310
5	Sex	0.210666	0.369746	0.012	0.008652
6	Has_Cabin	0.058520	0.085952	0.012	0.037197
7	Name_length	0.104465	0.049163	0.726	0.176652
8	FamilySize	0.068760	0.046206	0.048	0.112556
9	IsAlone	0.012929	0.020858	0.004	0.007099
10	Title	0.261510	0.177977	0.068	0.431154

特徵重要性視覺化

```
trace = go.Scatter(
    y = feature_dataframe['Random Forest feature importances'].values,
    x = feature_dataframe['features'].values,
    mode='markers',
    marker=dict(
        sizemode = 'diameter',
        sizeref = 1,
        size = 25,
        color = feature_dataframe['Random Forest feature importances'].values,
        colorscale='Portland',
        showscale=True
    ),
    text = feature_dataframe['features'].values
)
data = [trace]

layout= go.Layout(
    autosize= True,
    title= 'Random Forest Feature Importance',
    hovermode= 'closest',
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig,filename='scatter2010')
```



特徵重要性探討

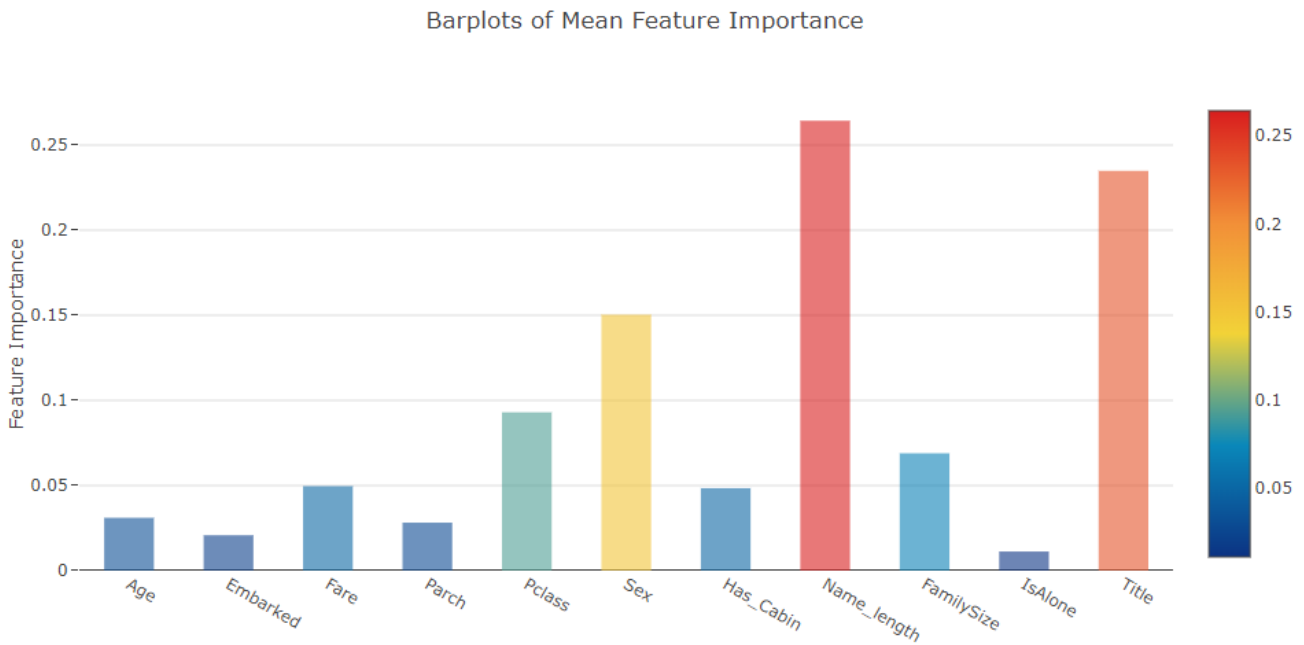
```
feature_dataframe['mean'] = feature_dataframe.mean(axis= 1)

feature_dataframe
```

將各模型的特徵重要性平均

	features	Random Forest feature importances	Extra Trees feature importances	AdaBoost feature importances	Gradient Boost feature importances	mean
0	Age	0.035300	0.030184	0.012	0.046591	0.031018
1	Embarked	0.022676	0.028065	0.008	0.024412	0.020788
2	Fare	0.071709	0.053145	0.016	0.057525	0.049595
3	Parch	0.021370	0.016403	0.064	0.010853	0.028157
4	Pclass	0.132096	0.122301	0.030	0.087310	0.092927
5	Sex	0.210666	0.369746	0.012	0.008652	0.150266
6	Has_Cabin	0.058520	0.085952	0.012	0.037197	0.048417
7	Name_length	0.104465	0.049163	0.726	0.176652	0.264070
8	FamilySize	0.068760	0.046206	0.048	0.112556	0.068880
9	IsAlone	0.012929	0.020858	0.004	0.007099	0.011221
10	Title	0.261510	0.177977	0.068	0.431154	0.234660

特徵重要性視覺化

```
y = feature_dataframe['mean'].values
x = feature_dataframe['features'].values
data = [go.Bar(
            x= x,
             y= y,
            width = 0.5,
            marker=dict(
               color = feature_dataframe['mean'].values,
            colorscale='Portland',
            showscale=True,
            reversescale = False
            ),
            opacity=0.6
        )]

layout= go.Layout(
    autosize= True,
    title= 'Barplots of Mean Feature Importance',
    hovermode= 'closest',
    yaxis=dict(
        title= 'Feature Importance',
        ticklen= 5,
        gridwidth= 2
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='bar-direct-labels')
```



模型評估

```
base_predictions_test = pd.DataFrame({
    'RandomForest': rf_oof_test.ravel(),
    'ExtraTrees': et_oof_test.ravel(),
    'AdaBoost': ada_oof_test.ravel(),
    'GradientBoost': gb_oof_test.ravel(),
    'SVM': svc_oof_test.ravel()
    })

base_predictions_test
```

將各模型預測結果列表

	RandomForest	ExtraTrees	AdaBoost	GradientBoost	SVM
0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.6	1.0	1.0	1.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	1.0	1.0	1.0	1.0	1.0

模型評估

```
base_predictions_test = base_predictions_test.replace(0.2, 0.0)
base_predictions_test = base_predictions_test.replace(0.4, 0.0)
base_predictions_test = base_predictions_test.replace(0.6, 1.0)
base_predictions_test = base_predictions_test.replace(0.8, 1.0)

base_predictions_test
```

	RandomForest	ExtraTrees	AdaBoost	GradientBoost	SVM
0	0.0	0.0	0.0	0.0	0.0
1	1.0	0.6	1.0	1.0	1.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	1.0	1.0	1.0	1.0	1.0

之前的預測為各fold平均，需轉為0 or 1

模型評估

```
ans = pd.read_csv('gender_submission.csv')

print(ans)
```

讀入標準答案

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0
10	902	0
11	903	0
12	904	1
13	905	0
14	906	1
15	907	1
16	908	0
17	909	0

模型評估

```
print('RandomForest準確率：',accuracy_score(base_predictions_test['RandomForest'], ans['Survived']))
print('ExtraTrees準確率：',accuracy_score(base_predictions_test['ExtraTrees'], ans['Survived']))
print('AdaBoost準確率：',accuracy_score(base_predictions_test['AdaBoost'], ans['Survived']))
print('GradientBoost準確率：',accuracy_score(base_predictions_test['GradientBoost'], ans['Survived']))
print('SVM準確率：',accuracy_score(base_predictions_test['SVM'], ans['Survived']))
```

```
RandomForest準確率： 0.8971291866028708
ExtraTrees準確率： 0.9019138755980861
AdaBoost準確率： 0.8827751196172249
GradientBoost準確率： 0.8253588516746412
SVM準確率： 0.937799043062201
```

由於部分模型有隨機特性，故數值不一定相同

進階技巧-ensemble

```
x_train = np.concatenate(( et_oof_train, rf_oof_train, ada_oof_train, gb_oof_train, svc_oof_train), axis=1)
x_test = np.concatenate(( et_oof_test, rf_oof_test, ada_oof_test, gb_oof_test, svc_oof_test), axis=1)
```

```
gbm = xgb.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)
```

```
print('xgboost準確率：',accuracy_score(predictions, ans['Survived']))
```

```
xgboost準確率： 0.9066985645933014
```

All you need is python!