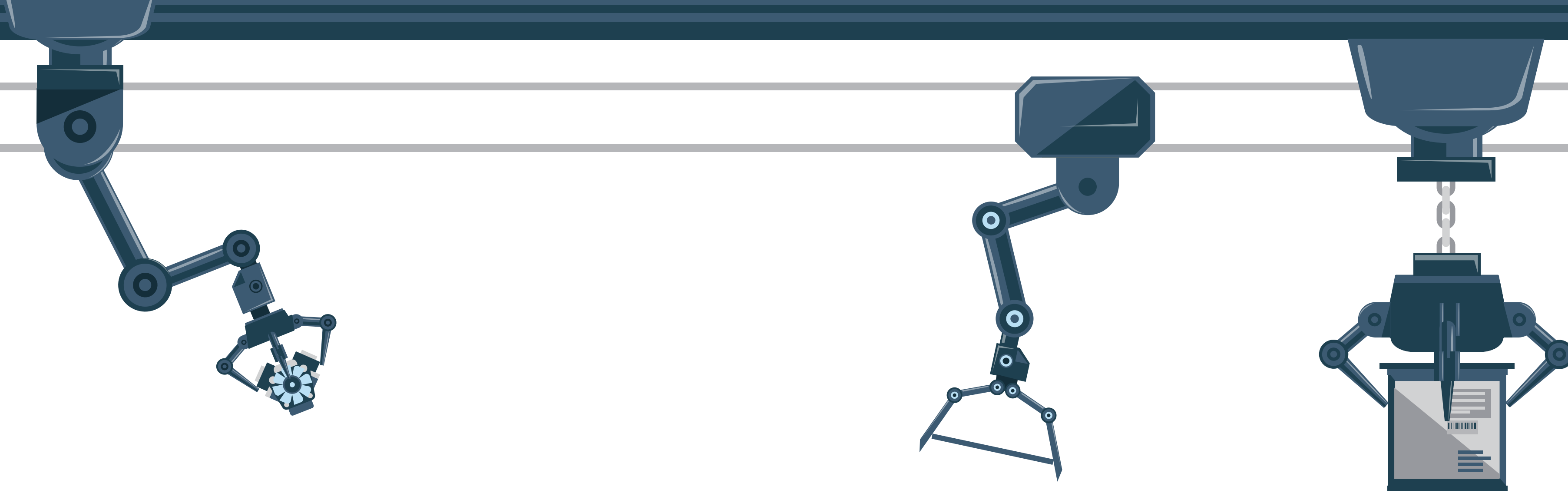


機器學習期末報告
Kaggle競賽-鐵達尼號生存預測

第3組 11024113侯亮羽 11024333邱寶樟 11024144李奕呈

Kaggle競賽-鐵達尼號生存預測

- 根據鐵達尼號乘客資料預測生還者，使用train.csv進行生還者預測。
- 特徵名稱說明

PassengerId: 乘客代碼

Survived: 生存

Pclass: 票價等級

Name: 姓名

Sex: 性別

Age: 年齡

SibSp: 配偶、親屬

Parch: 家長、小孩

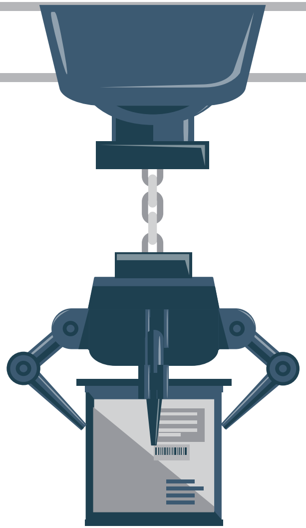
Ticket: 票號

Fare: 票價

Cabin:座艙號碼

Embarked:登船點

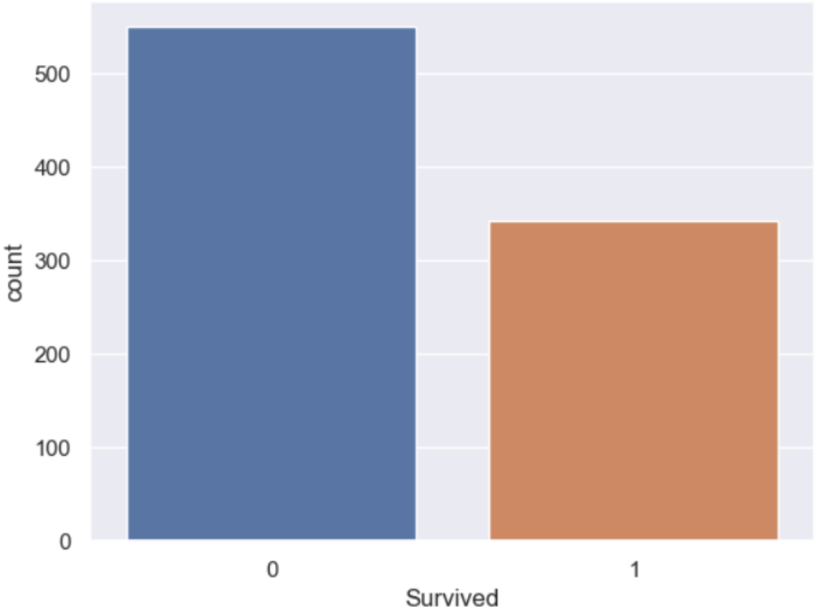
引入資料、觀察數值



檢查存活率—存活、死亡人數

```
print(data['Survived'].value_counts())
sns.set_theme()
sns.countplot(data, x='Survived')
```

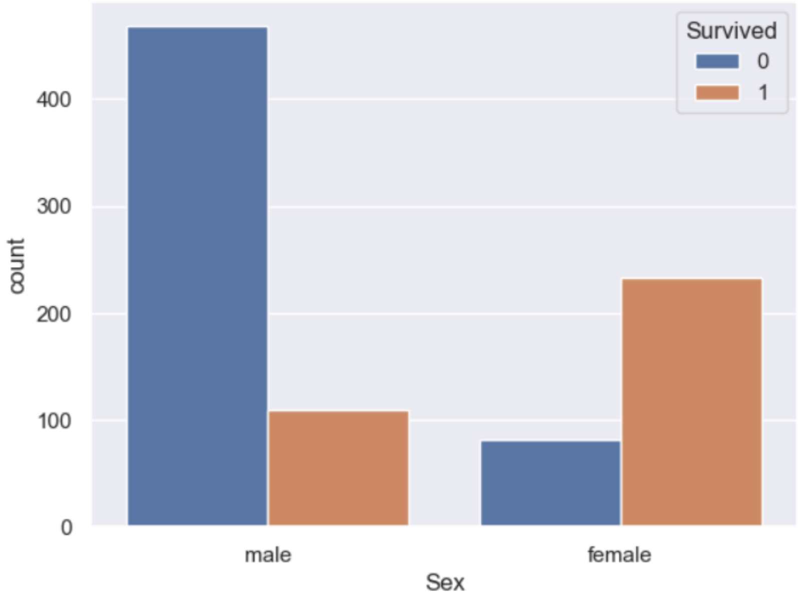
```
0      549
1      342
```



檢查存活率—性別

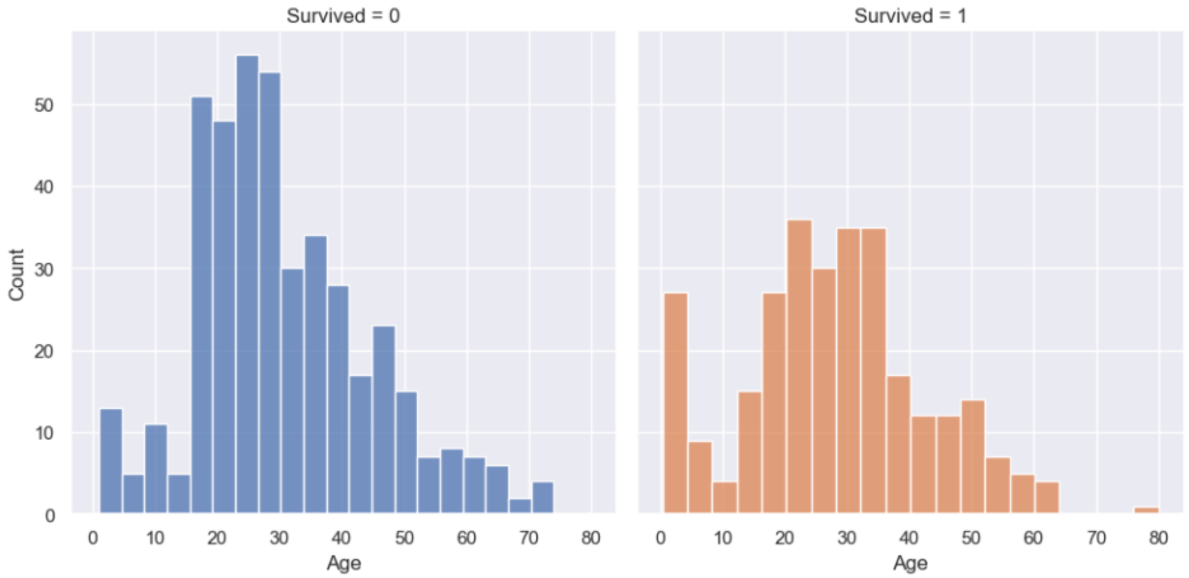
```
sns.countplot(data, x='Sex', hue='Survived')
display(data[['Sex', 'Survived']].groupby(['Sex'], as_index=False).mean().round(3))
```

	Sex	Survived
0	female	0.742
1	male	0.189

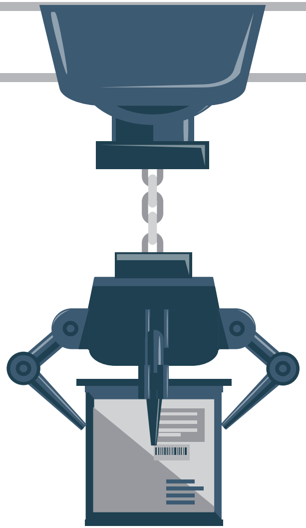


檢查存活率—年齡

```
age_survied = sns.FacetGrid(data, col='Survived', hue='Survived', height=5, aspect=1)
age_survied.map(sns.histplot, 'Age', bins=20, kde=False)
```



資料預處理



丟棄不必要的資料

```
dropped_data = data.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1)
dropped_data.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

初始資料vs丟棄後的資料

初始資料

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

丟棄後的資料

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Survived   891 non-null    int64
 1   Pclass     891 non-null    int64
 2   Sex        891 non-null    object
 3   Age        714 non-null    float64
 4   SibSp      891 non-null    int64
 5   Parch      891 non-null    int64
 6   Fare       891 non-null    float64
 7   Embarked   889 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

填補資料

- 數值資料：以「中位數」取代
- 類別資料：以「眾數」取代

```
dropped_data['Age'] =
dropped_data['Age'].fillna(dropped_data['Age'].median())

dropped_data['Embarked'] =
dropped_data['Embarked'].fillna(dropped_data['Embarked']
.mode().iloc[0])

dropped_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    object
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  891 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

處理類別資料

- 「類別」資料進行 One Hot Encoder

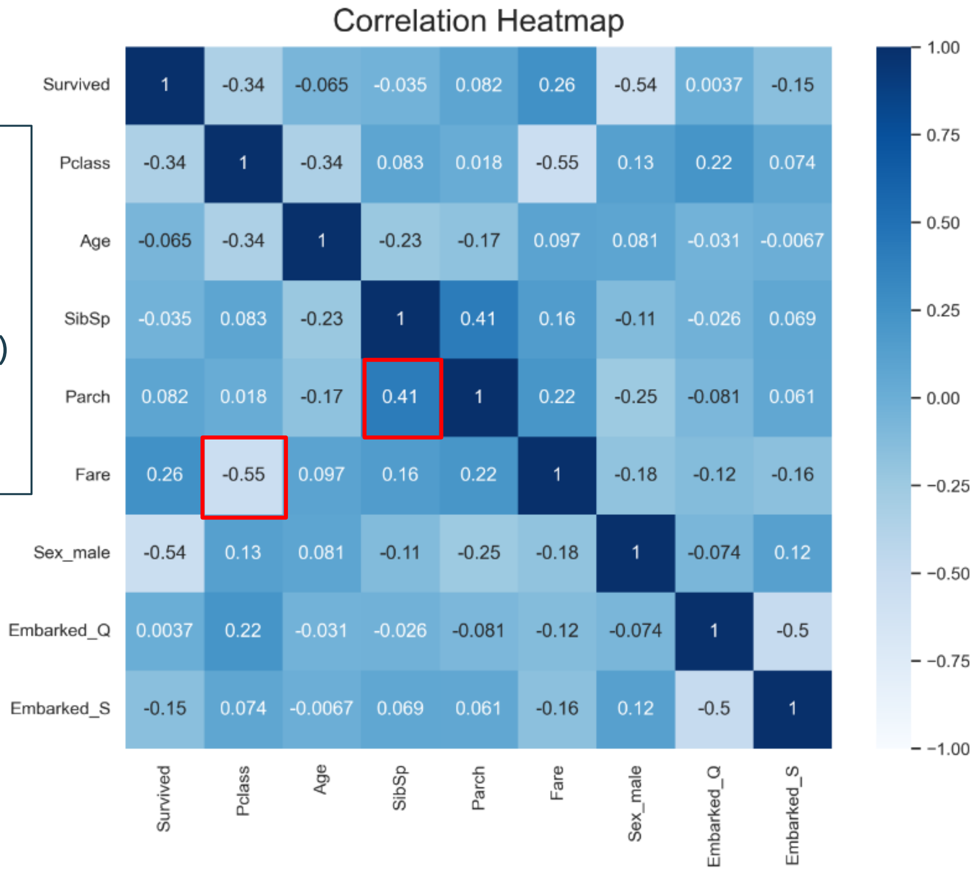
```
dropped_data = pd.get_dummies(dropped_data, drop_first=True).astype('float64')
dropped_data.head(2)
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	0.0	3.0	22.0	1.0	0.0	7.2500	1.0	0.0	1.0
1	1.0	1.0	38.0	1.0	0.0	71.2833	0.0	0.0	0.0

尋找相關性

```
dropped_data.corr()

plt.figure(figsize=(10, 8), dpi=300)

heatmap = sns.heatmap(dropped_data.corr(), vmin=-1, vmax=1, annot=True,cmap='Blues’ )

heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':20}, pad=10)
```

```
Positive Max Correlation:
Survived  Survived    1.0
dtype: float64

Negative Max Correlation:
Pclass  Fare   -0.5495
dtype: float64
```



資料標準化

```
from sklearn.preprocessing import MinMaxScaler
ss = MinMaxScaler()
X_scaled = ss.fit_transform(dropped_data.iloc[:, 1:])
pd.DataFrame(X_scaled, columns = dropped_data.columns[ 1 :])
```

	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	1.0	0.271174	0.125	0.000000	0.014151	1.0	0.0	1.0
1	0.0	0.472229	0.125	0.000000	0.139136	0.0	0.0	0.0
2	1.0	0.321438	0.000	0.000000	0.015469	0.0	0.0	1.0
3	0.0	0.434531	0.125	0.000000	0.103644	0.0	0.0	1.0
4	1.0	0.434531	0.000	0.000000	0.015713	1.0	0.0	1.0
...	...	...	...	...	...	...	...	...
886	0.5	0.334004	0.000	0.000000	0.025374	1.0	0.0	1.0
887	0.0	0.233476	0.000	0.000000	0.058556	0.0	0.0	1.0
888	1.0	0.346569	0.125	0.333333	0.045771	0.0	0.0	1.0
889	0.0	0.321438	0.000	0.000000	0.058556	1.0	0.0	0.0
890	1.0	0.396833	0.000	0.000000	0.015127	1.0	1.0	0.0

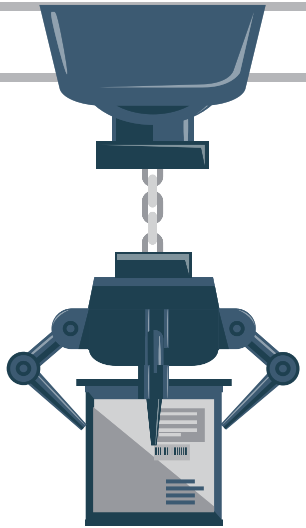
891 rows × 8 columns

分層取樣訓練與測試集

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, dropped_data.iloc[:, 0], test_size=0.2,
                                                    random_state=42, stratify=dropped_data.iloc[:, 0])

print('y_train data: ', y_train.value_counts(normalize=True))
print('\n\ny_test data: ', y_test.value_counts(normalize=True))
```

```
y_train data:  Survived
0.0    0.616573
1.0    0.383427
Name: proportion, dtype: float64


y_test data:  Survived
0.0    0.614525
1.0    0.385475
Name: proportion, dtype: float64
```

訓練與預測模型



Logistic regression

```
from sklearn.linear_model import LogisticRegression

params = {
    ‘C’  : [0.01, 0.1, 1]
}

log_reg = LogisticRegression(random_state=42)
log_grid_search_cv = GridSearchCV(log_reg, params, n_jobs=-1, verbose=1, cv=10)
log_grid_search_cv.fit(X_train, y_train)
print(log_grid_search_cv.best_params_)
print('Logistic regression training score: %.3f' % log_grid_search_cv.score(X_train, y_train))
print('Logistic regression testing score: %.3f' % log_grid_search_cv.score(X_test, y_test))
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits
{'C': 1}
Logistic regression training score: 0.801
Logistic regression testing score: 0.793
```

Decision tree

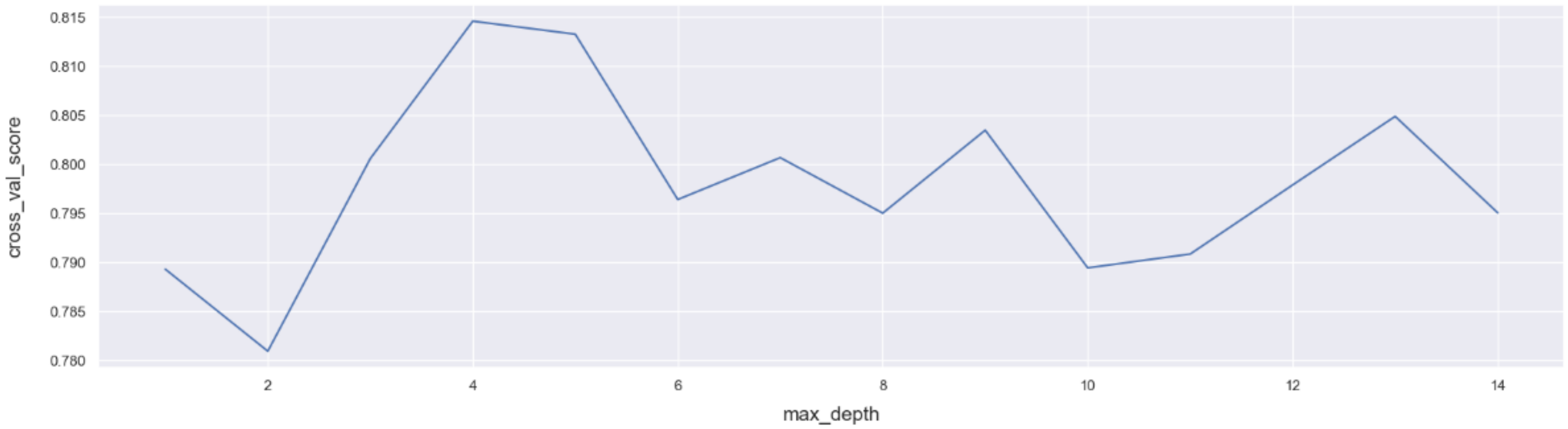
- find the optimal 'max_depth'

```
all_score = []
for i in range(1, 15, 1):
    tree_clf = DecisionTreeClassifier(max_depth=i, random_state=42)
    score = cross_val_score(tree_clf, X_train, y_train, cv=10).mean()
    all_score.append([i,score])
all_score = np.array(all_score)

max_score = np.where(all_score==np.max(all_score[:,1]))[0][0]
print("max_depth, cross_val_score:",all_score[max_score])
plt.figure(figsize=[20,5])
plt.plot(all_score[:,0],all_score[:,1])
plt.xlabel('max_depth', fontsize=15, labelpad=10)
plt.ylabel('cross_val_score', fontsize=15, labelpad=20)
plt.show()
```

Decision tree

- find the optimal 'max_depth'

```
max_depth, cross_val_score: [4.          0.81459311]
```



Decision tree

- find the optimal hyperparameters

```
params = {
    'max_depth':np.arange(3, 6),
    'min_samples_split':np.arange(2, 5),
    'min_samples_leaf':np.arange(2, 6),
}

tree_clf = DecisionTreeClassifier(random_state=42)
tree_grid_search_cv = GridSearchCV(tree_clf, params, n_jobs=-1, verbose=1, cv=10, 
scoring='roc_auc')
tree_grid_search_cv.fit(X_train, y_train)
print(tree_grid_search_cv.best_params_)
print('Decision Tree training score: %.3f' % tree_grid_search_cv.score(X_train, y_train))
print('Decision Tree testing score: %.3f' % tree_grid_search_cv.score(X_test, y_test))
```

```
Fitting 10 folds for each of 36 candidates, totalling 360 fits
{'max_depth': 4, 'min_samples_leaf': 4, 'min_samples_split': 2}
Decision Tree training score: 0.888
Decision Tree testing score: 0.818
```

Random Forest

```
params = {
   'max_depth':np.arange(3, 6),
   'min_samples_split':np.arange(2, 5),
   'min_samples_leaf':np.arange(1, 3),
}

rnd_clf = RandomForestClassifier(random_state=42)
rnd_grid_search_cv = GridSearchCV(rnd_clf, params, n_jobs=-1, verbose=1, cv=10,
scoring='roc_auc')
rnd_grid_search_cv.fit(X_train, y_train)
print(rnd_grid_search_cv.best_params_)
print('Random Forest training score: %.3f' % rnd_grid_search_cv.score(X_train, y_train))
print('Random Forest testing score: %.3f' % rnd_grid_search_cv.score(X_test, y_test))
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
{'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 3}
Random Forest training score: 0.912
Random Forest testing score: 0.841
```

XGboost

```
params = {
    'n_estimators': range(2, 11),
    'learning_rate': [.1, .2, .3],
    'colsample_bytree': [.6, .7, .8],
    'max_depth':range(2, 5),
    'subsample': [.7, .8, .9],
    'min_child_weight':range(1,3)
}

xgb_clf = xgb.XGBClassifier(random_state=42)
xgb_grid_search_cv = GridSearchCV(xgb_clf, params, n_jobs=-1, verbose=1, cv=10, scoring='roc_auc')
xgb_grid_search_cv.fit(X_train, y_train)
print(xgb_grid_search_cv.best_params_)
print('XGBoost training score: %.3f' % xgb_grid_search_cv.score(X_train, y_train))
print('XGBoost testing score: %.3f' % xgb_grid_search_cv.score(X_test, y_test))
```

```
Fitting 10 folds for each of 1458 candidates, totalling 14580 fits
{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 4, 'min_child_weight': 1, 'n_estimators': 8, 'subsample': 0.7}
XGBoost training score: 0.902
XGBoost testing score: 0.833
```

Adaboost

```
params = {
    'n_estimators': range(80, 100),
    'learning_rate': [0.1, 0.2, 0.3]
}

ada_clf = AdaBoostClassifier(random_state=42)
ada_grid_search_cv = GridSearchCV(ada_clf, params, n_jobs=-1, verbose=1, cv=10, scoring='roc_auc')
ada_grid_search_cv.fit(X_train, y_train)
print(ada_grid_search_cv.best_params_)
print('AdaBoost training score: %.3f' % ada_grid_search_cv.score(X_train, y_train))
print('AdaBoost Tree testing score: %.3f' % ada_grid_search_cv.score(X_test, y_test))
```

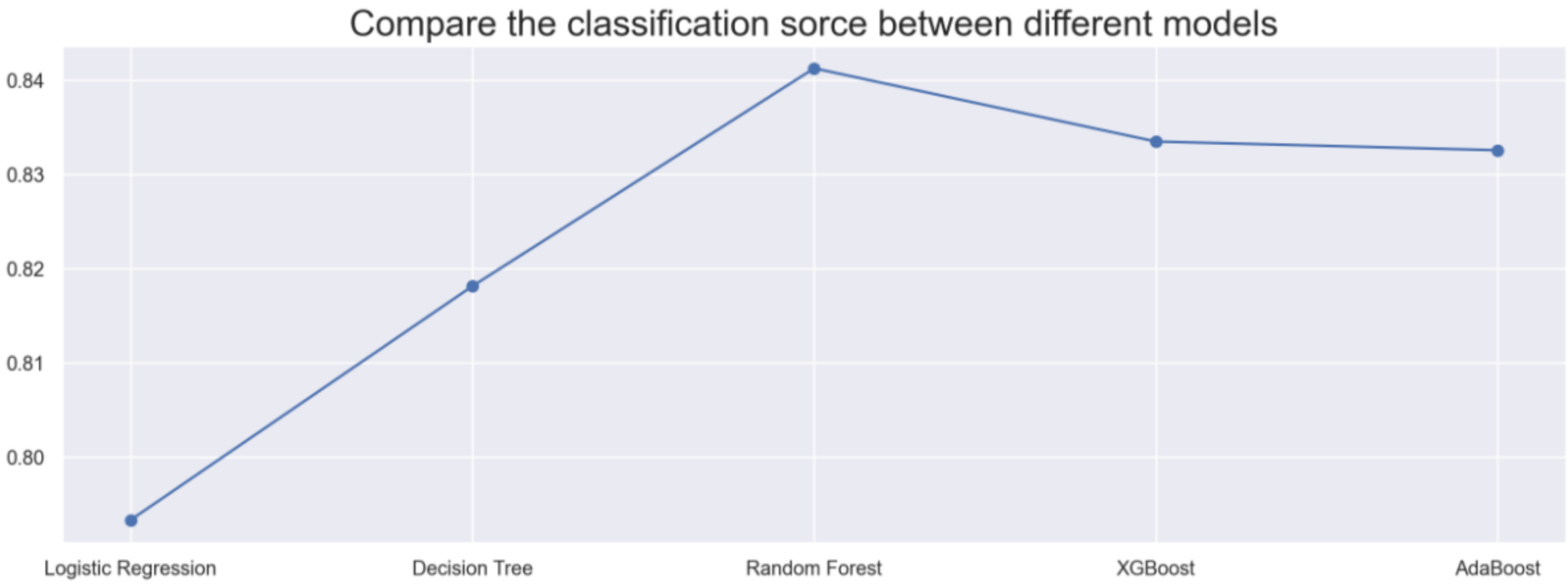
```
Fitting 10 folds for each of 60 candidates, totalling 600 fits
{'learning_rate': 0.1, 'n_estimators': 87}
AdaBoost training score: 0.885
AdaBoost Tree testing score: 0.833
```

統整與結論

比對不同模型 testing score 的數值

```
acc_log = log_grid_search_cv.score(X_test, y_test)
acc_tree = tree_grid_search_cv.score(X_test, y_test)
acc_rnd = rnd_grid_search_cv.score(X_test, y_test)
acc_xgb = xgb_grid_search_cv.score(X_test, y_test)
acc_ada = ada_grid_search_cv.score(X_test, y_test)

model = pd.DataFrame({
    'Model':['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'AdaBoost'],
    'Score':[acc_log, acc_tree, acc_rnd, acc_xgb, acc_ada]
})
model.sort_values(by='Score', ascending=False)
```

	Model	Score
2	Random Forest	0.841238
3	XGBoost	0.833465
4	AdaBoost	0.832543
1	Decision Tree	0.818116
0	Logistic Regression	0.793296

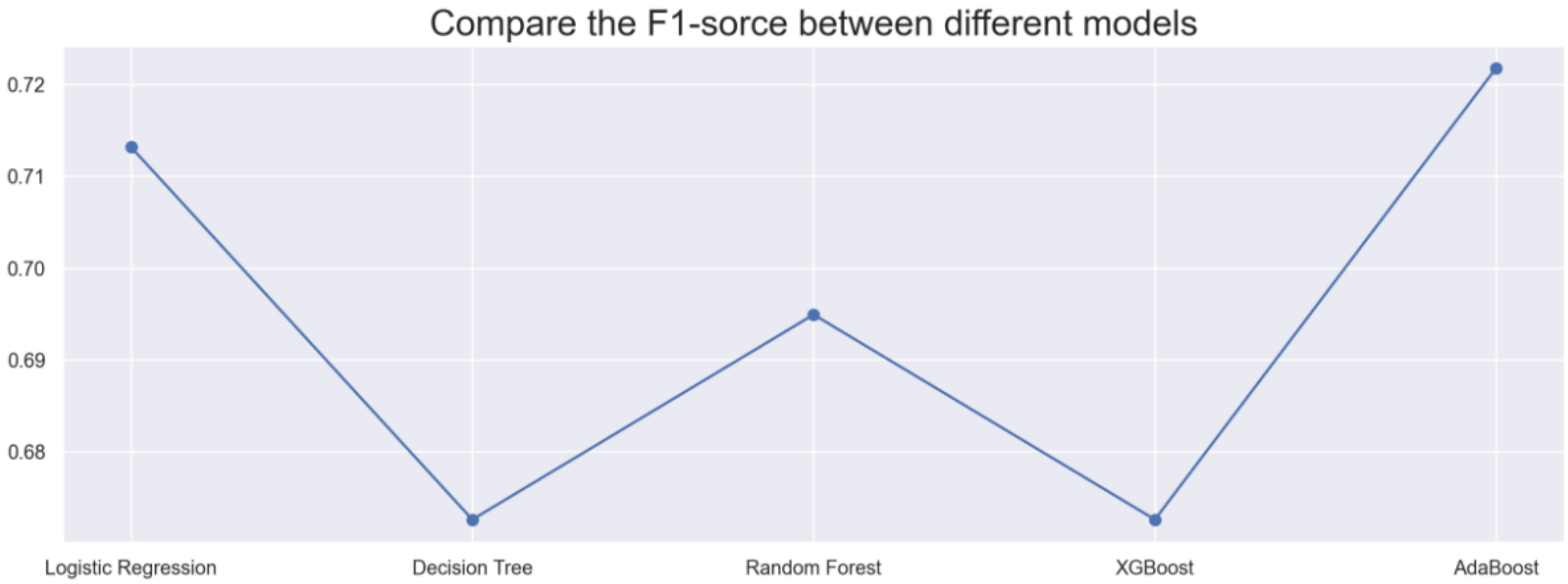

Compare the classification sorce between different models
0.84
0.83
0.82
0.81
0.80
Logistic Regression
Decision Tree
Random Forest
XGBoost
AdaBoost

比對不同模型的 F1-score

```
from sklearn.metrics import f1_score

y_pred_log = log_reg.predict(X_test)
f1_log = f1_score(y_test, y_pred_log)

y_pred_tree = tree_grid_search_cv.predict(X_test)
f1_tree = f1_score(y_test, y_pred_tree)

y_pred_rnd = rnd_grid_search_cv.predict(X_test)
f1_rnd = f1_score(y_test, y_pred_rnd)

y_pred_xgb = xgb_grid_search_cv.predict(X_test)
f1_xgb = f1_score(y_test, y_pred_xgb)

y_pred_ada = ada _grid_search_cv.predict(X_test)
f1_ada = f1_score(y_test, y_pred_ada)
```

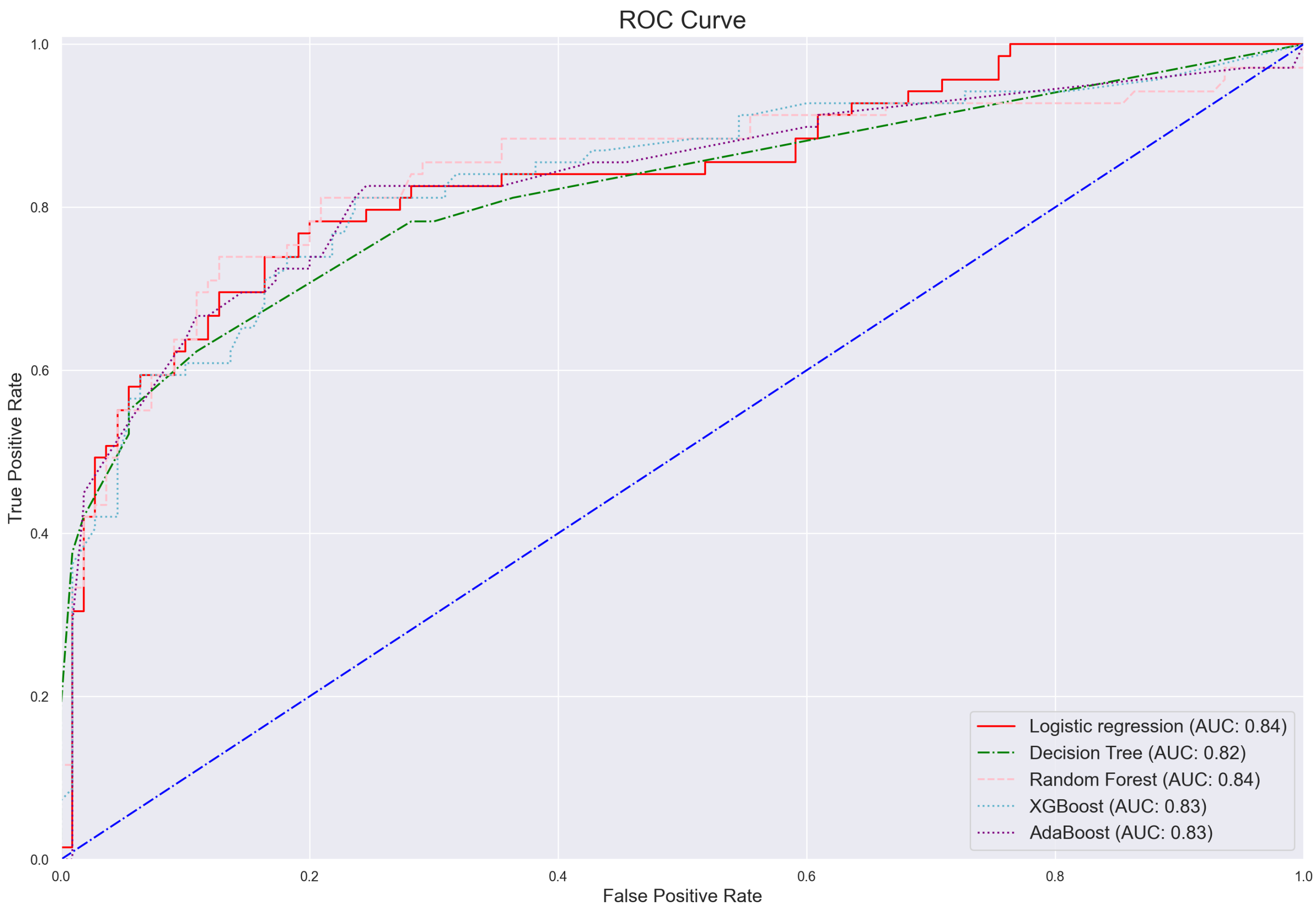
```
model2 = pd.DataFrame({
    'Model':['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'AdaBoost'],
    'F1-score':[f1_log, f1_tree, f1_rnd, f1_xgb, f1_ada]
})
model2.sort_values(by='F1-score', ascending=False)
```

	Model	F1-score
4	AdaBoost	0.721805
0	Logistic Regression	0.713178
2	Random Forest	0.694915
1	Decision Tree	0.672566
3	XGBoost	0.672566

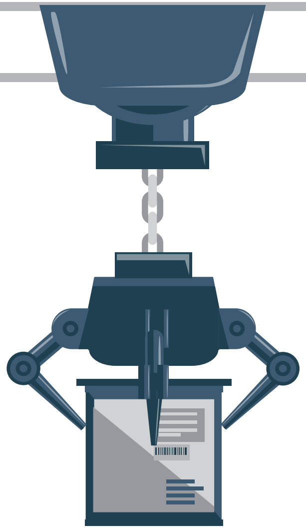

Compare the F1-sorce between different models
0.72
0.71
0.70
0.69
0.68
Logistic Regression
Decision Tree
Random Forest
XGBoost
AdaBoost

比對不同模型的 ROC 與 AUC

```
from sklearn.metrics import roc_curve, auc

log_y_score = log_grid_search_cv.predict_proba(X_test)[:, 1]
log_fpr, log_tpr, thresholds = roc_curve(y_test, log_y_score)

tree_y_score = tree_grid_search_cv.predict_proba(X_test)[:, 1]
tree_fpr, tree_tpr, thresholds = roc_curve(y_test, tree_y_score)

rnd_y_score = rnd_grid_search_cv.predict_proba(X_test)[:, 1]
rnd_fpr, rnd_tpr, thresholds = roc_curve(y_test, rnd_y_score)

xgb_y_score = xgb_grid_search_cv.predict_proba(X_test)[:, 1]
xgb_fpr, xgb_tpr, thresholds = roc_curve(y_test, xgb_y_score)

ada_y_score = ada_grid_search_cv.predict_proba(X_test)[:, 1]
ada_fpr, ada_tpr, thresholds = roc_curve(y_test, ada_y_score)
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic regression (AUC: 0.84)
Decision Tree (AUC: 0.82)
Random Forest (AUC: 0.84)
XGBoost (AUC: 0.83)
AdaBoost (AUC: 0.83)

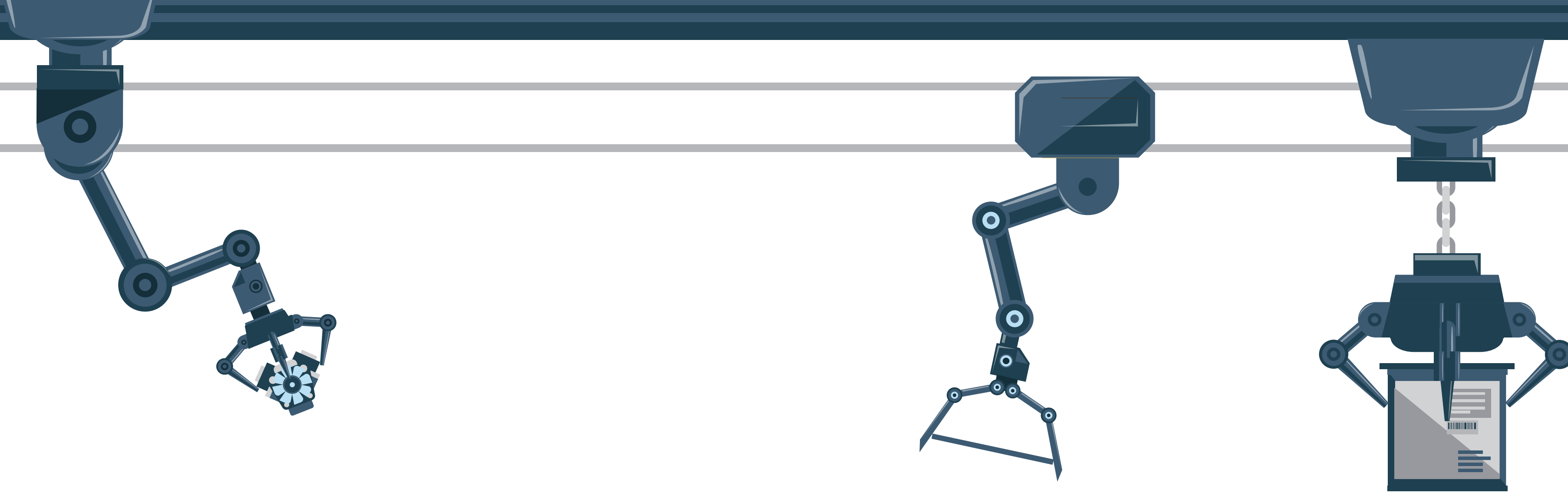
参考資料



- ［資料分析&機器學習］第4.1講:Kaggle競賽-鐵達尼號生存預測 (前 16%排名)，2017
  https://medium.com/jameslearningnote/資料分析-機器學習-第4-1講-kaggle競賽-鐵達尼號生存預測-前16-排名-a8842fea7077

- ［機器學習專案］Kaggle競賽-鐵達尼號生存預測(Top 3%)，2018
  https://yulongtsai.medium.com/https-medium-com-yulongtsai-titanic-top3-8e64741cc11f

- Titanic-Machine Learning from Disaster 鐵達尼號生存預測資料分析篇［資料分析與機器學習系列］，2023
  https://hackmd.io/@Go3PyC86QhypSl7kh5nA2Q/Hk4nXFYkK



Thanks for listening !