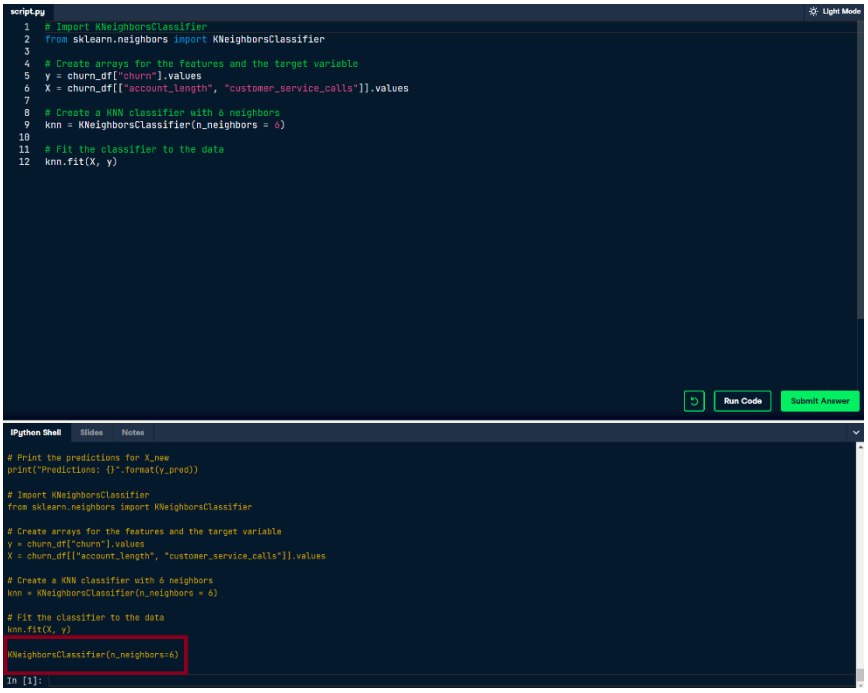
–ML 監督式學習

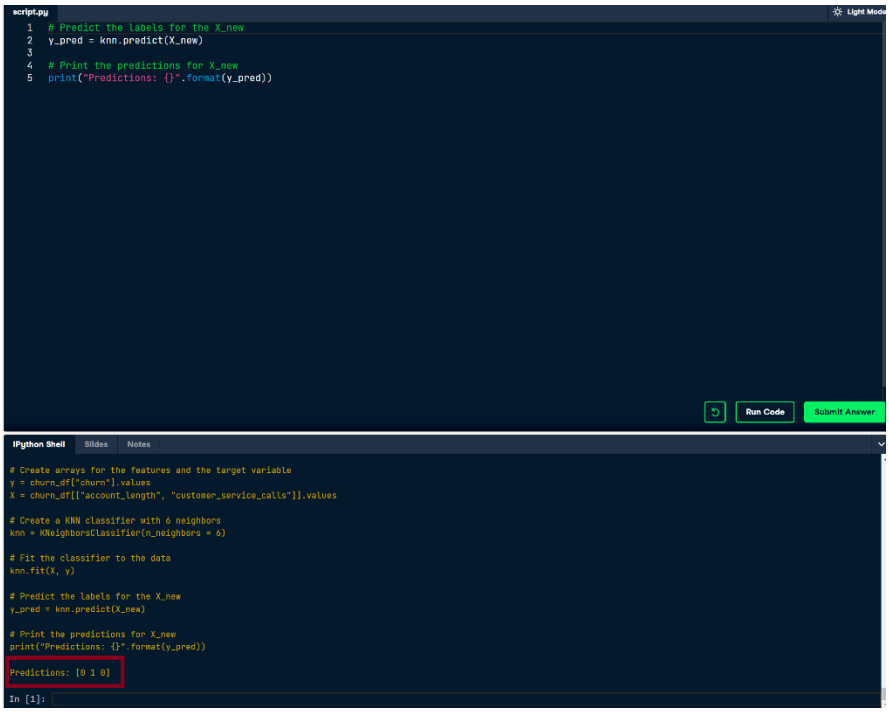
(Machine Learning, Supervised Learning, KNN)

Knn 模型設定以 k 值為 6，將模型 fit 到(X, y)



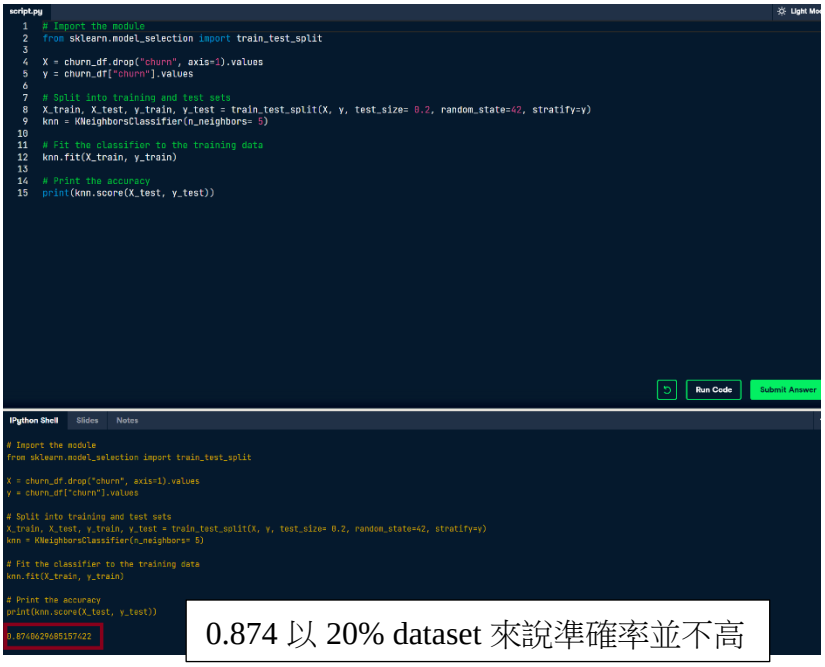
帶入 y_pred 得到預測結果

0=未流失客戶，1=流失客戶

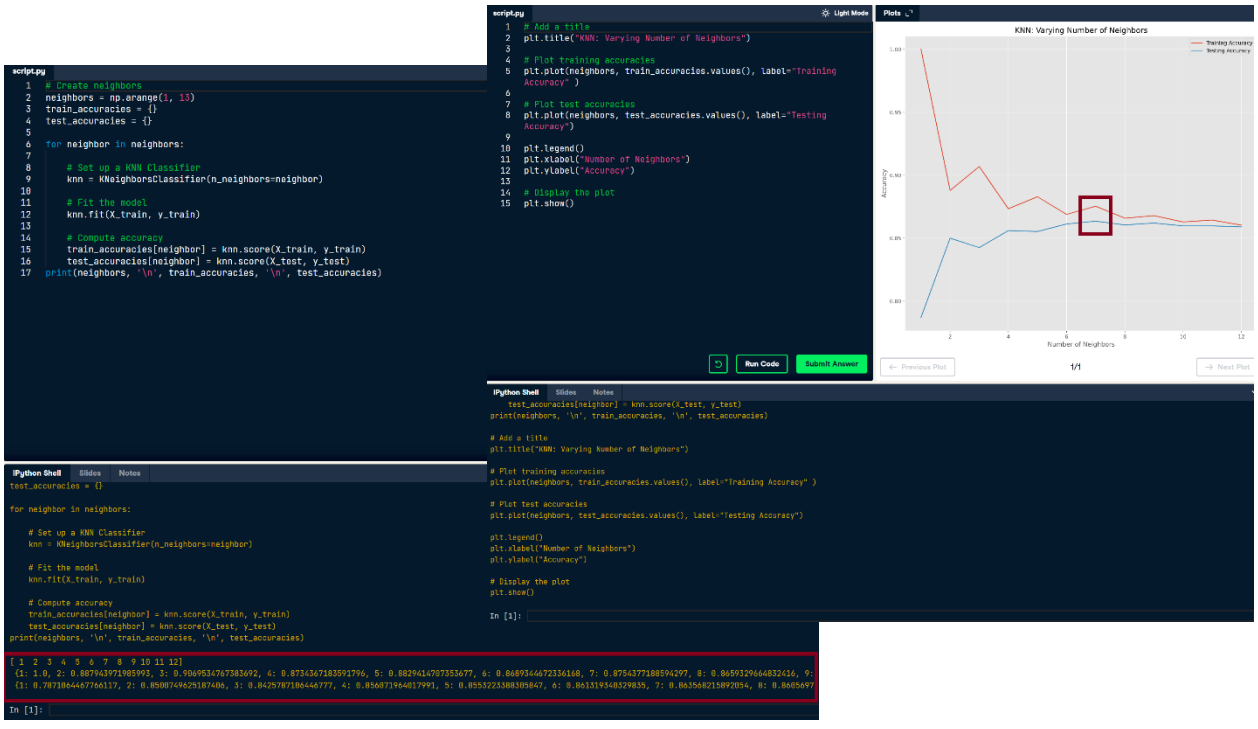


Model Measuring 模型準確度測試

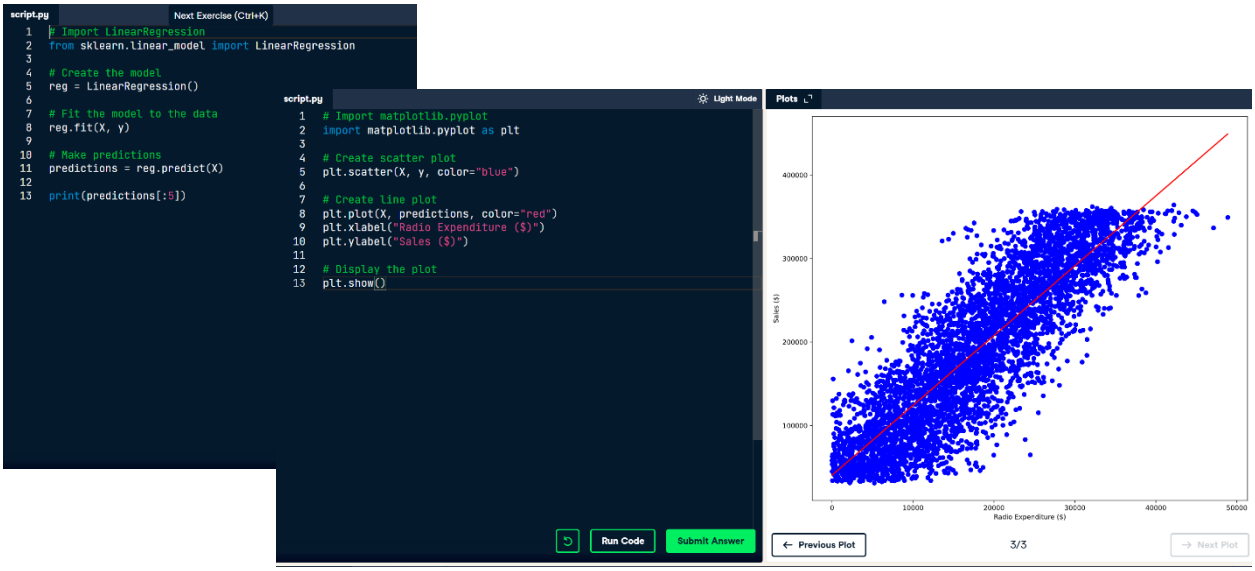
以 knn 模型測試準確度
將 test 跟 train 從 dataset 中取出 20%，種子設定 42
Stratify 穩定 test 跟 train 的比例



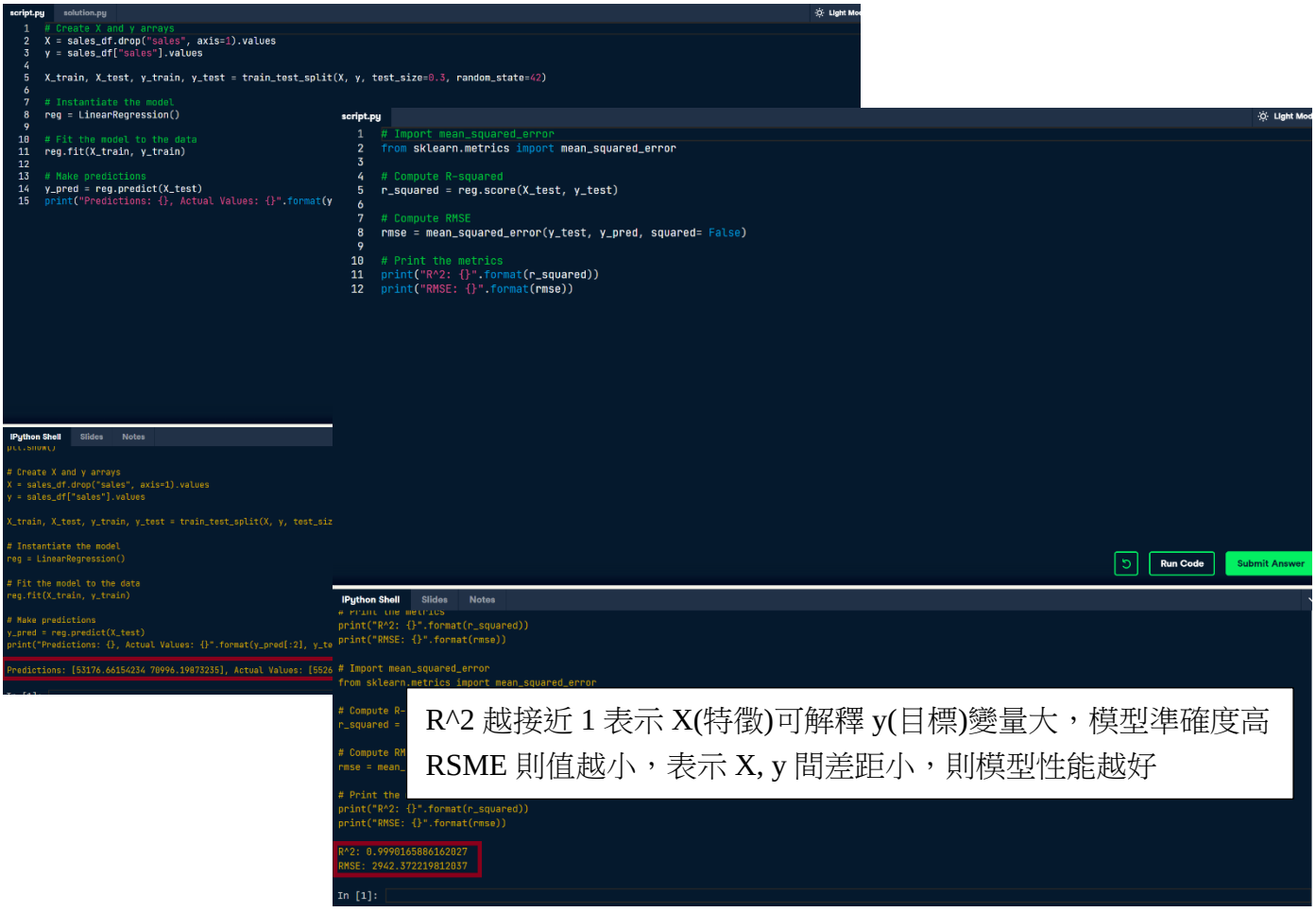
或是用 Model Complexity 和 knn 模型訓練(左圖)
以 for loop 得出 X, y 準確率
而(右圖)視覺化後可以發現，K 值越大，過度擬合；K 值越小，則未擬合
右圖中紅框即為最佳擬合點



將 X, y fit 到 LinearRegression 作為預測模型
再用 plt.plot 視覺化觀察線性(預測)與散點圖(原值)差異

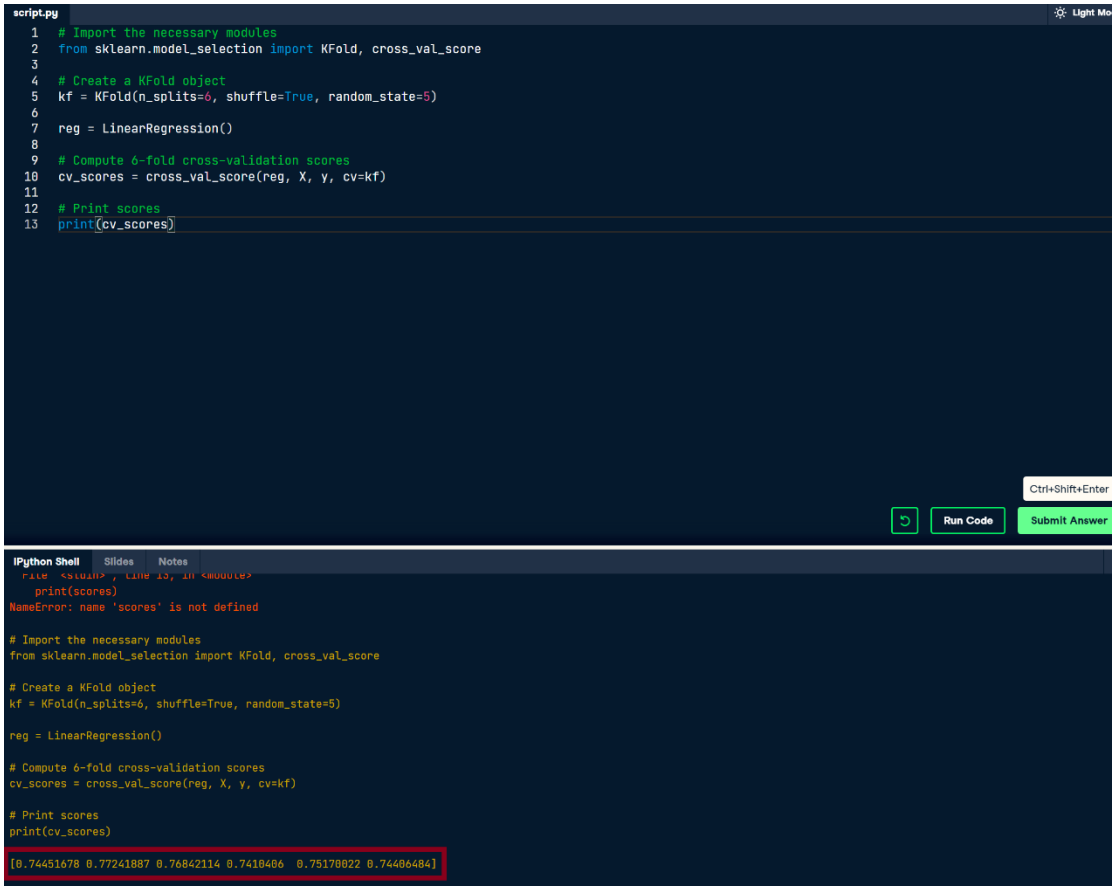


利用 .drop 分出特徵(X)及目標組(y)做預測
再計算 R^2 跟 RMSE 測試模型準確度



R^2 越接近 1 表示 X(特徵)可解釋 y(目標)變量大，模型準確度高
RSME 則值越小，表示 X, y 間差距小，則模型性能越好

利用 KFold,　cross_val_score 做交叉驗證 (cross validation)

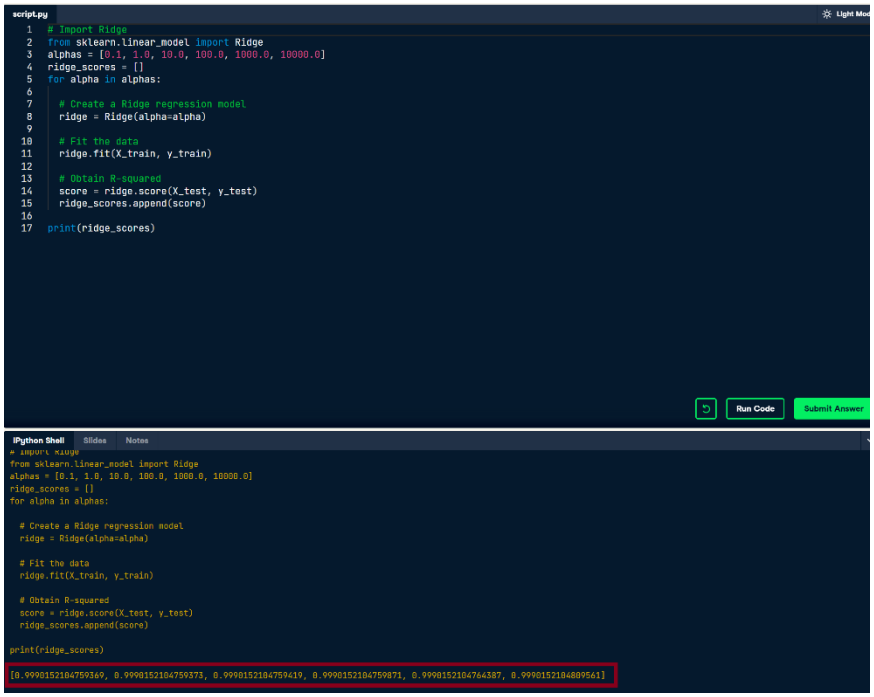


Regularized Regression 正規化模型

(Ridge, Lasso)

Ridge:

當 alphas 值越大，模型過度擬合；反之，則未擬合 (alphas 類似 K 值)

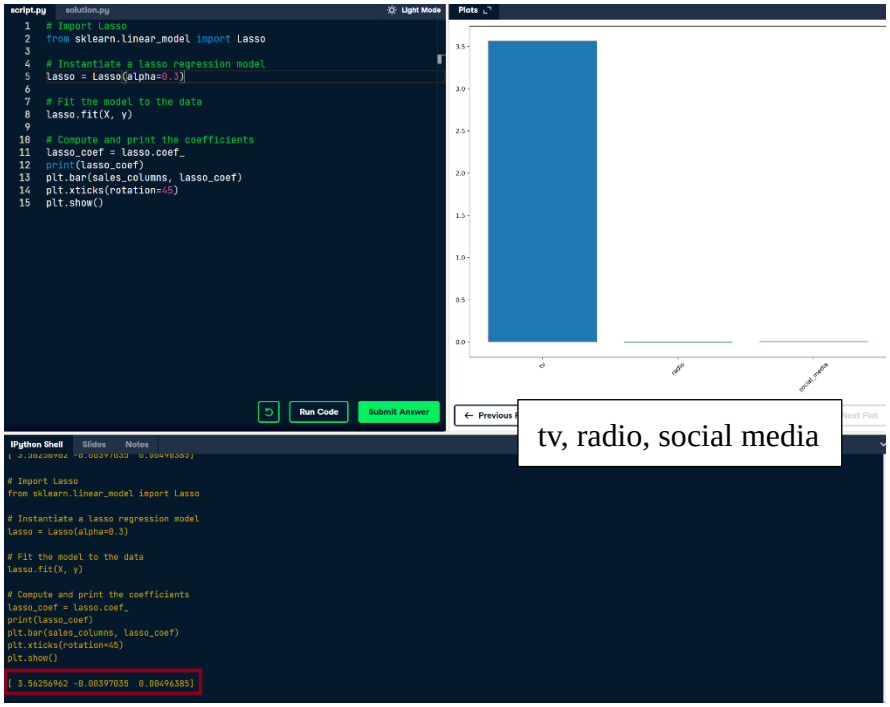


Lasso:

Lasso 傾向把不重要的特徵係數縮小至 0，可用來驗證特徵 X 的重要性

圖中，電視的係數最高，電台與社群媒體則近於 0

得出電視為主要商品銷售平台

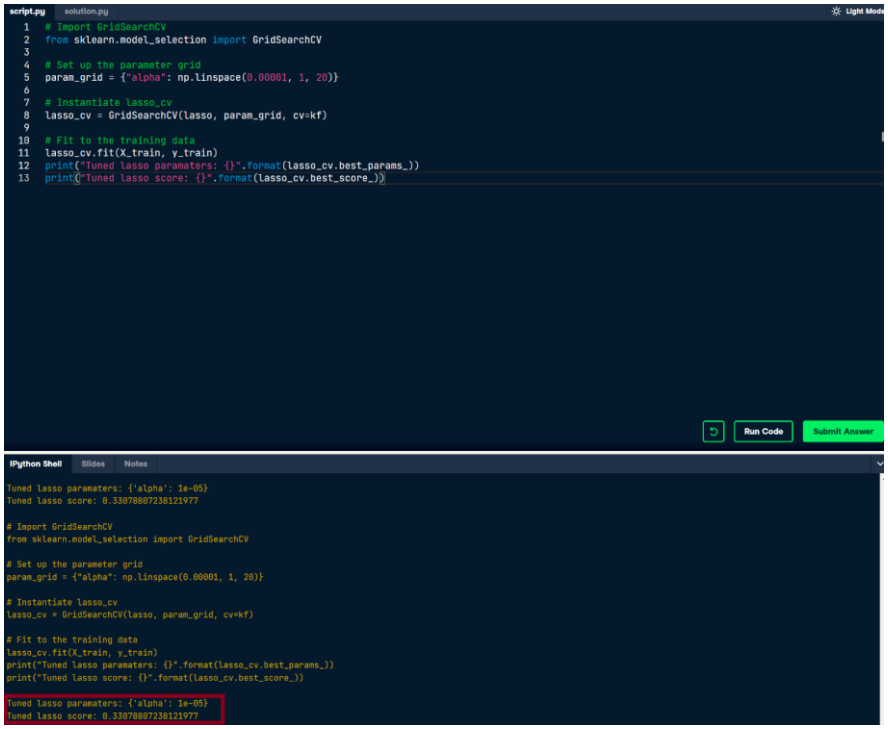


ROC 模型可視化

可看出該模型的表現比原本隨機觀測的結果好很多
(ROC 曲線在虛線上方)

GridSearch & RandomizedSearch 網格搜尋&隨機搜尋

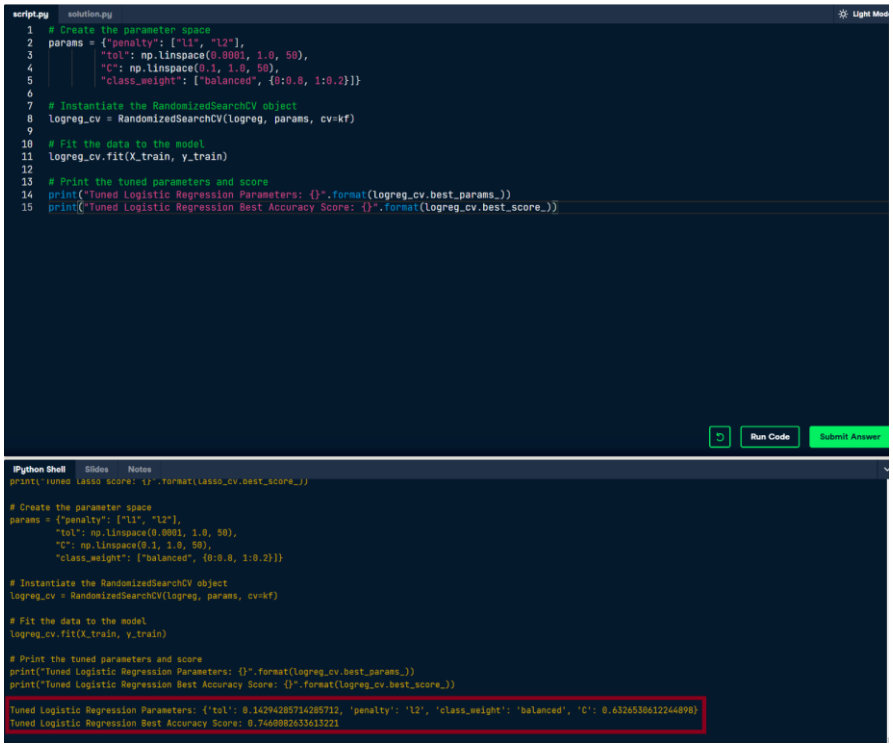
利用 GridSearch 調整超參數



```
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Set up the parameter grid
param_grid = {"alpha": np.linspace(0.00001, 1, 20)}

# Instantiate lasso_cv
lasso_cv = GridSearchCV(lasso, param_grid, cv=kf)

# Fit to the training data
lasso_cv.fit(X_train, y_train)
print("Tuned lasso paramaters: {}".format(lasso_cv.best_params_))
print("Tuned lasso score: {}".format(lasso_cv.best_score_))
```

```
Tuned lasso paramaters: {'alpha': 1e-05}
Tuned lasso score: 0.3307888723812197
```

如果資料量龐大，用 RandomizeSearch 調整超參數
縮短學習時間

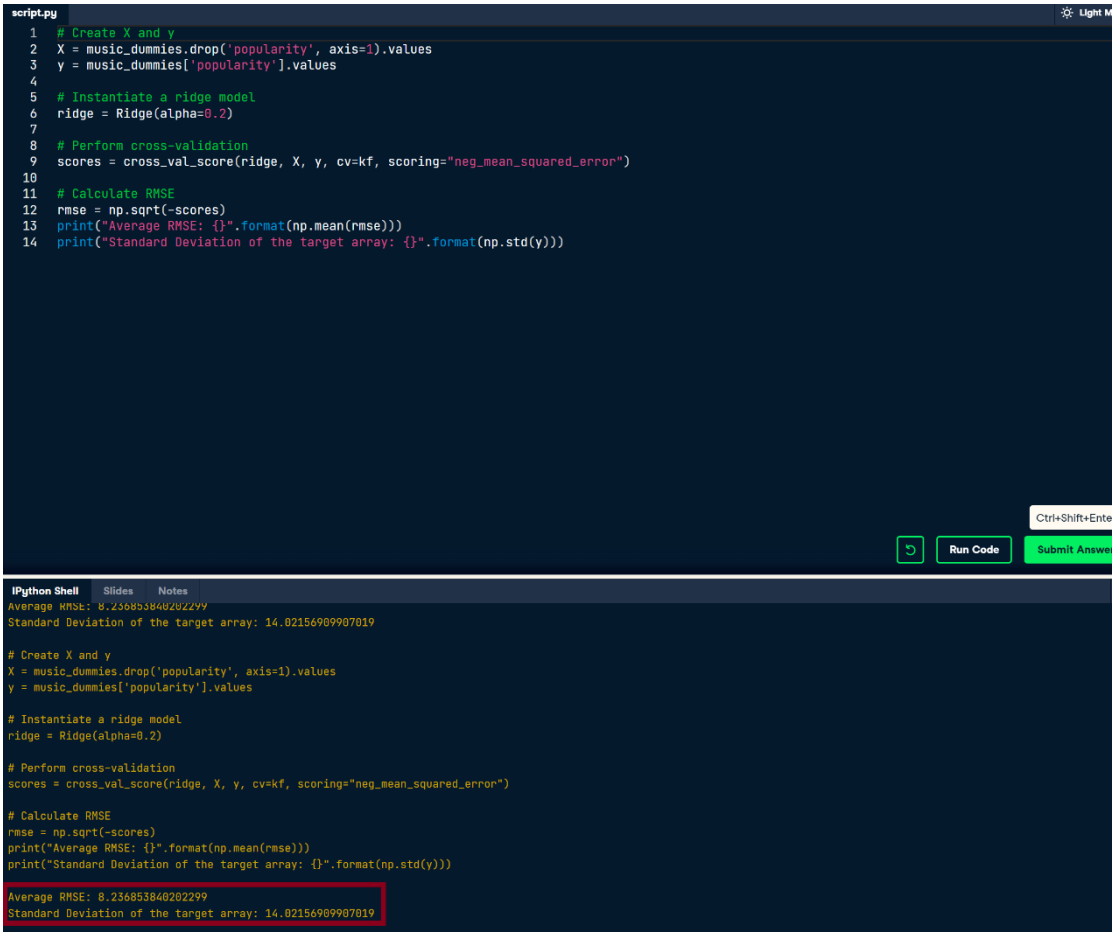


```
# Create the parameter space
params = {"penalty": ["l1", "l2"],
          "tol": np.linspace(0.0001, 1.0, 50),
          "C": np.linspace(0.1, 1.0, 50),
          "class_weight": ["balanced", {0:0.8, 1:0.2}]}

# Instantiate the RandomizedSearchCV object
logreg_cv = RandomizedSearchCV(logreg, params, cv=kf)

# Fit the data to the model
logreg_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Best Accuracy Score: {}".format(logreg_cv.best_score_))
```

```
Tuned Logistic Regression Parameters: {'tol': 0.14294285714285712, 'penalty': 'l2', 'class_weight': 'balanced', 'C': 0.6326530612244898}
Tuned Logistic Regression Best Accuracy Score: 0.7460082633613221
```

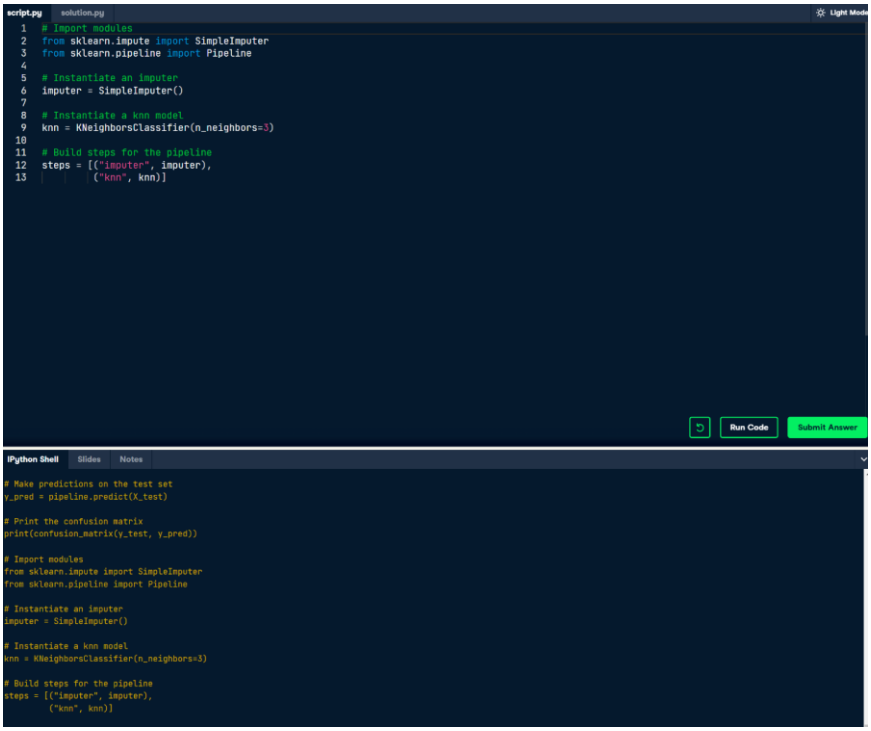
Dummy 轉換

Dummy 將資料轉換為 0 跟 1 (原為音樂種類)，以利於模型應用
算出 RMSE & STD 評估模型



Pipeline 管道模型

先設置 steps (knn & imputer)



```
# Import modules
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Instantiate an imputer
imputer = SimpleImputer()

# Instantiate a knn model
knn = KNeighborsClassifier(n_neighbors=3)

# Build steps for the pipeline
steps = [("imputer", imputer),
         ("knn", knn)]
```

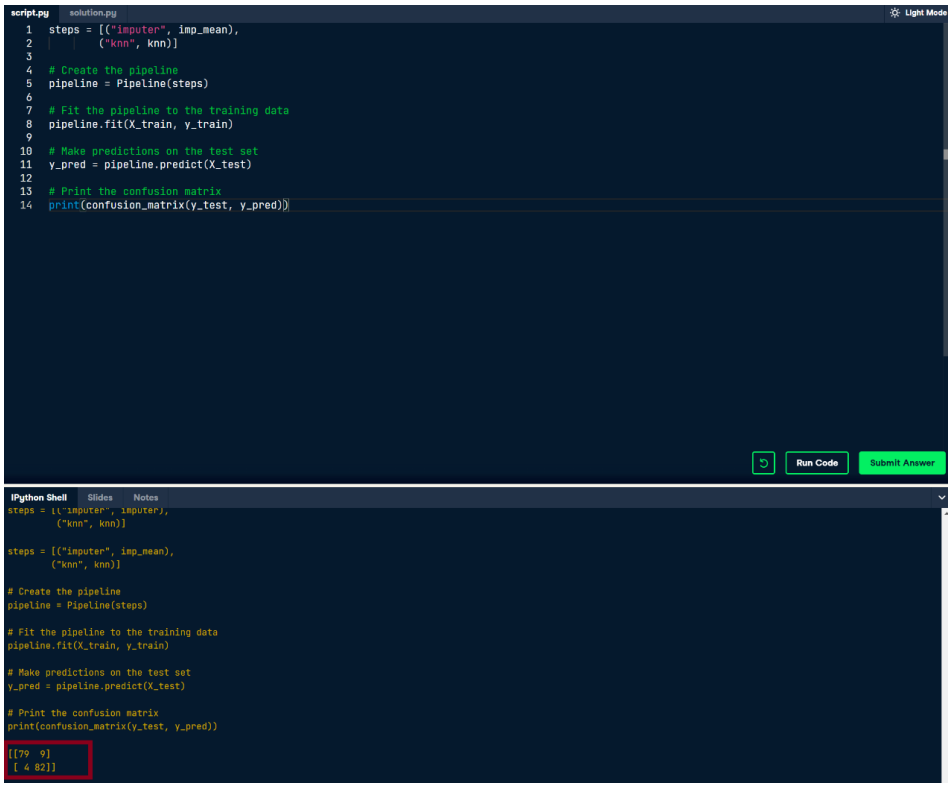
Run Code | Submit Answer

IPython Shell | Slides | Notes

```
# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))

# Import modules
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Instantiate an imputer
imputer = SimpleImputer()

# Instantiate a knn model
knn = KNeighborsClassifier(n_neighbors=3)

# Build steps for the pipeline
steps = [("imputer", imputer),
         ("knn", knn)]
```

steps 放到 Pipeline 裡

Confusion Matrix 自動處理(即 Pipeline)後，得出結果



```
steps = [("imputer", imp_mean),
         ("knn", knn)]

# Create the pipeline
pipeline = Pipeline(steps)

# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
```

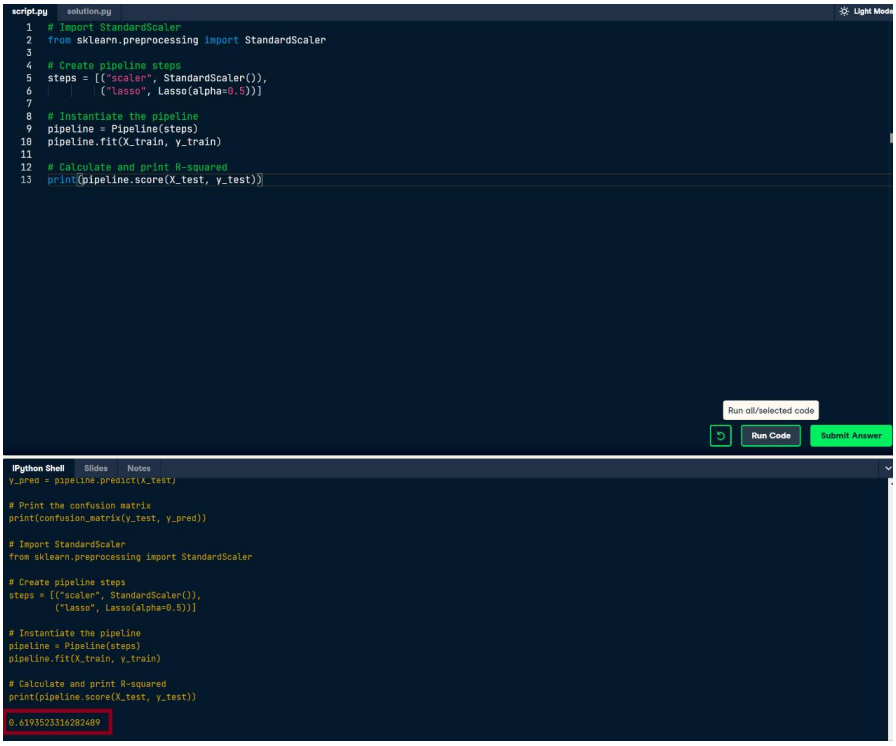
Run Code | Submit Answer

IPython Shell | Slides | Notes

```
steps = [("imputer", imputer),
         ("knn", knn)]

steps = [("imputer", imp_mean),
         ("knn", knn)]

# Create the pipeline
pipeline = Pipeline(steps)

# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))

[[79  9]
 [ 4 82]]
```

Scale 模型縮放

在模型縮放前，score 為 0.619



```
# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Create pipeline steps
steps = [("scaler", StandardScaler()),
         ("lasso", Lasso(alpha=0.5))]

# Instantiate the pipeline
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)

# Calculate and print R-squared
print(pipeline.score(X_test, y_test))
```

Run all/selected code

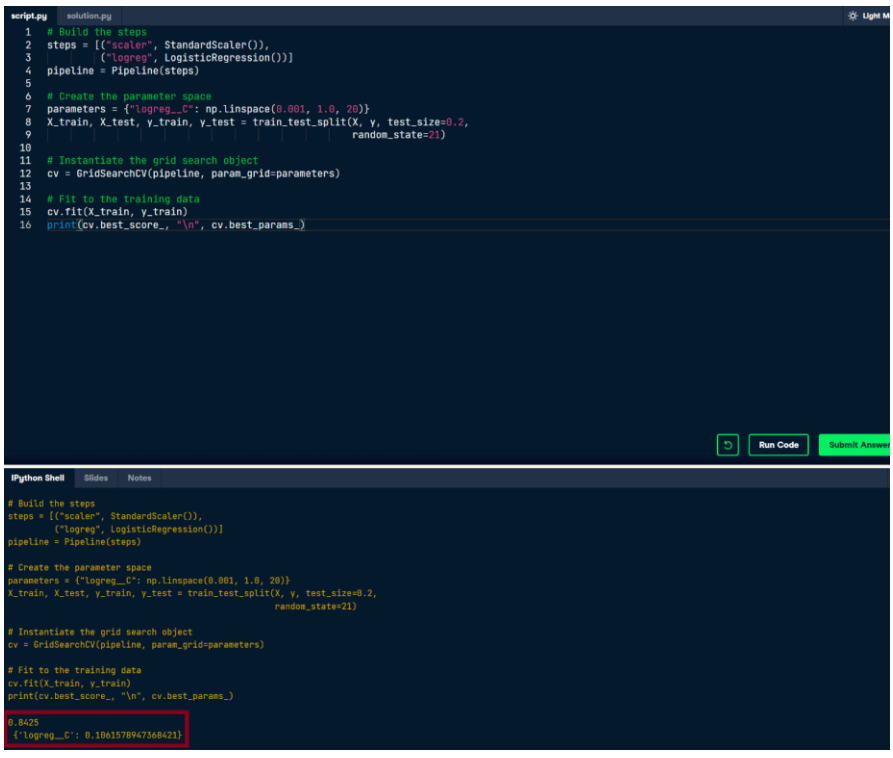
Run Code

Submit Answer

IPython Shell Slides Notes

```
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))

# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Create pipeline steps
steps = [("scaler", StandardScaler()),
         ("lasso", Lasso(alpha=0.5))]

# Instantiate the pipeline
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)

# Calculate and print R-squared
print(pipeline.score(X_test, y_test))

0.6193523316282489
```

縮放後為 0.8425，準確率上升



```
# Build the steps
steps = [("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]
pipeline = Pipeline(steps)

# Create the parameter space
parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)

# Instantiate the grid search object
cv = GridSearchCV(pipeline, param_grid=parameters)

# Fit to the training data
cv.fit(X_train, y_train)
print(cv.best_score_, "\n", cv.best_params_)
```

Run Code

Submit Answer

IPython Shell Slides Notes

```
# Build the steps
steps = [("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]
pipeline = Pipeline(steps)

# Create the parameter space
parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)

# Instantiate the grid search object
cv = GridSearchCV(pipeline, param_grid=parameters)

# Fit to the training data
cv.fit(X_train, y_train)
print(cv.best_score_, "\n", cv.best_params_)

0.8425
 {'logreg__C': 0.10615789473684211}
```

Pipeline & Scale 最終組合