DỰ ÁN CUỐI KHÓA

FOUNDATIONS OF MACHINE LEARNING 08

Người thực hiện: Võ Nguyễn Nhật Duy

Thành phố Hồ Chí Minh - 2025

Mục lục

1 Giới Thiệu Bài Toán

Dự án này tập trung vào việc xây dựng một mô hình học máy để dự đoán khả năng sống sót của hành khách trên tàu Titanic dựa trên các đặc trưng như độ tuổi, giới tính, hạng vé, và các thông tin khác. Vụ đắm tàu Titanic năm 1912 là một trong những thảm họa hàng hải nổi tiếng nhất trong lịch sử, và bài toán này nhằm khai thác dữ liệu lịch sử để đưa ra dự đoán chính xác về khả năng sống sót.

Mục tiêu chính của dự án là:

- Xây dựng một pipeline học máy hoàn chỉnh từ tiền xử lý dữ liệu đến huấn luyện mô hình.
- Đánh giá và so sánh hiệu suất của các mô hình học máy khác nhau.
- Triển khai mô hình dưới dạng API và ứng dụng web để sử dụng thực tế.

2 Khám Phá và Tiền Xử Lí Dữ Liệu

Bộ dữ liệu Titanic được lấy từ nguồn Kaggle, bao gồm thông tin về 891 hành khách với các đặc trưng như:

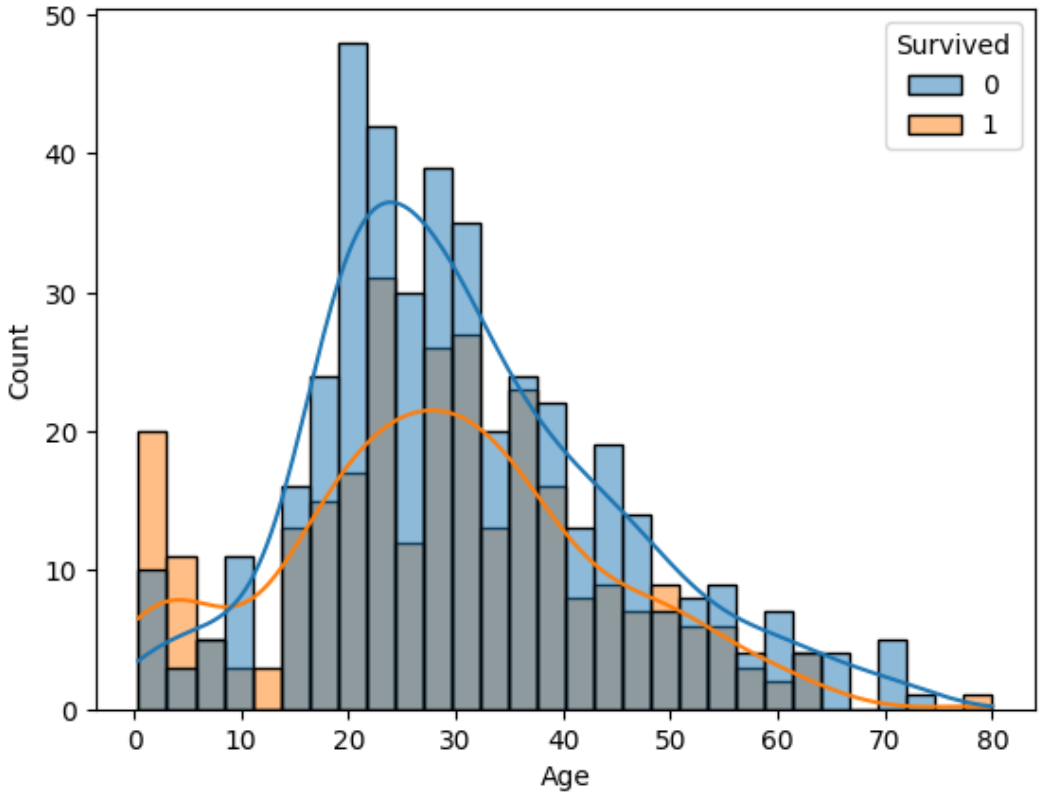
- `Pclass`: Hạng vé (1, 2, hoặc 3).
- `Sex`: Giới tính (male, female).
- `Age`: Tuổi của hành khách.
- `SibSp`: Số lượng anh chị em hoặc vợ/chồng trên tàu.
- `Parch`: Số lượng cha mẹ hoặc con cái trên tàu.
- `Fare`: Giá vé.
- `Embarked`: Cảng lên tàu (C, Q, S).
- `Survived`: Biến mục tiêu (0 = không sống sót, 1 = sống sót).

Một vài dòng đầu tiên của dữ liệu:

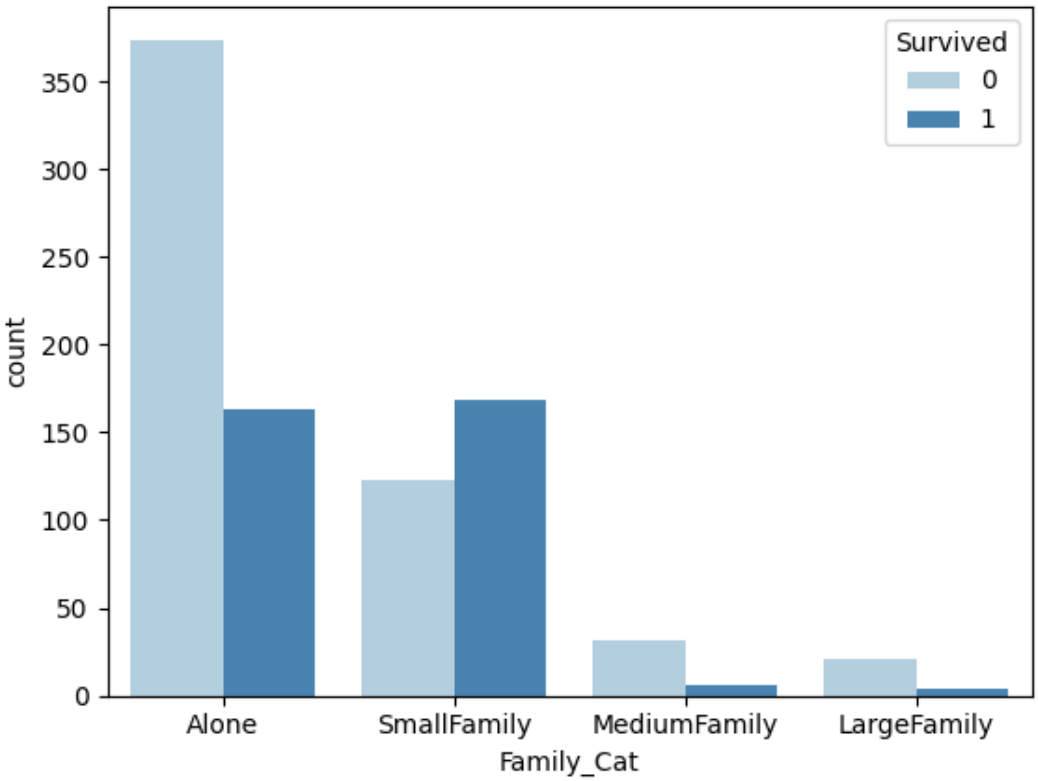
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 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

2.1 Khám Phá Dữ Liệu

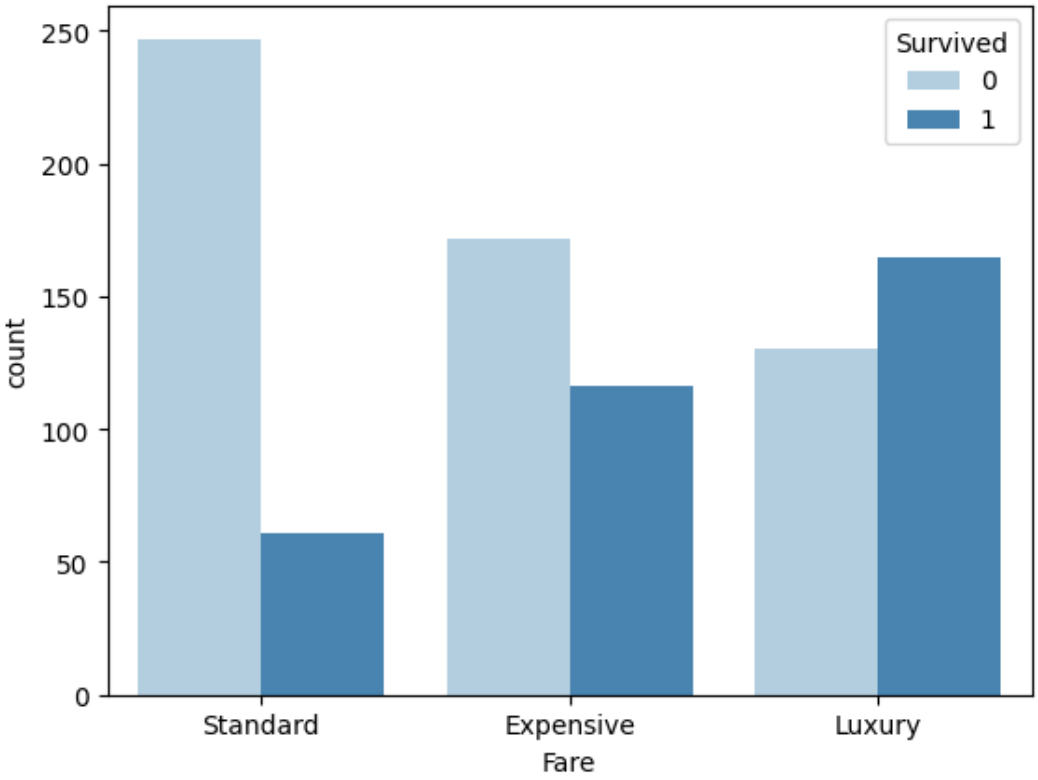
- **Phân bố sống sót**: Khoảng 38% hành khách trong tập dữ liệu sống sót.
- **Giới tính**: Tỷ lệ sống sót của nữ (74%) cao hơn nhiều so với nam (19%).
- **Hạng vé**: Hành khách ở hạng vé cao (Pclass=1) có tỷ lệ sống sót cao hơn (63%) so với hạng thấp (Pclass=3, 24%).
- **Nơi khởi hành**: Hành khách khởi hành ở ba cảng khác nhau, dữ liệu cho thấy hành khách khỏi hành tại cảng C (Cherbourg) sẽ có tỉ lệ sống sót cao nhất, tiếp đến là cảng Q (Queenstown) và cuối cùng là S (Southampton).
- **Độ tuổi**: Phân bố nhiều vào khoảng từ 20 đến 50 tuổi, trẻ em dưới 10 tuổi có tỉ lệ sống sót cao nhất).
- **Gia đình**: hai đặc trưng SibSp và Parch sẽ được gộp thành Family, chúng ta sẽ tạo 1 đặc trưng mới là Family_Cat tương ứng với 4 cấp độ Alone (đi một mình), SmallFamily, MediumFamily và LargeFamily. Dữ liệu cho thấy khi đi cùng từ 1-3 người (SmallFamily) tỉ lệ sống sót sẽ cao nhất.
- **Độ tuổi**: Phân bố nhiều vào khoảng từ 20 đến 50 tuổi, trẻ em dưới 10 tuổi có tỉ lệ sống sót cao nhất.



Tỷ lệ sống sót theo độ tuổi



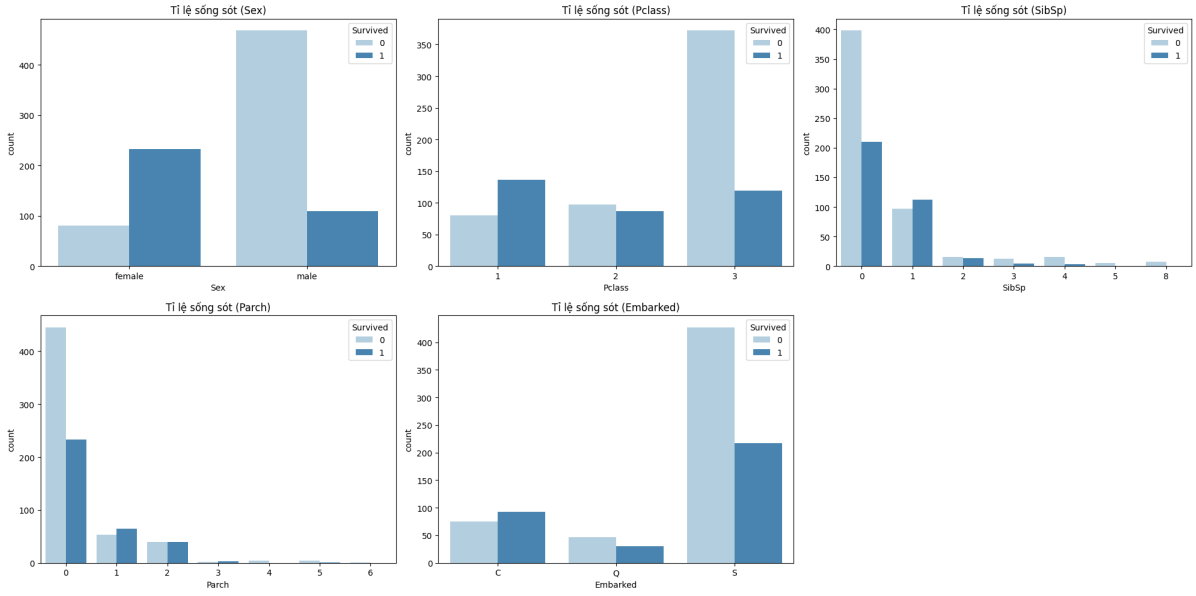
Tỷ lệ sống sót theo gia đình



Tỷ lệ sống sót theo hạng vé

Ta có thể sử dụng subplots trong thư viện matplotlib để vẽ nhiều biểu đồ cùng một lúc cho các đặc trưng

```
categories = ['Sex', 'Pclass', 'SibSp', 'Parch', 'Embarked']

n_rows = 2
n_cols = 3

fig, ax = plt.subplots(n_rows, n_cols, figsize=(20, 10))

for r in range(n_rows):
    for c in range(n_cols):
        i = r * n_cols + c
        if i < len(categories):
            sns.countplot(x=categories[i], data=train_df,
               hue='Survived', palette='Blues', ax=ax[r][c])
            ax[r][c].set_title(f"Survived rate
               ({categories[i]})")
        else:
            ax[r][c].axis('off')

plt.tight_layout()
```



2.2 Tiền Xử Lí Dữ Liệu

Quá trình tiền xử lý dữ liệu là một bước quan trọng trong phân tích dữ liệu và xây dựng mô hình học máy, đảm bảo dữ liệu đầu vào được làm sạch, chuẩn hóa và sẵn sàng cho các mô hình học máy.

Feature Engineering

Ta sẽ tạo thêm đặc trưng Title tương ứng với danh xưng của hành khách trên chuyến tàu. Ta có thể sử dụng biểu thức chính quy (regex) hoặc phương pháp tách chuỗi để lấy ra các danh xưng như "Mr", "Mrs", "Miss", "Master", "Dr", v.v. Sau khi trích xuất, có thể mã hóa chúng thành các biến số bằng OneHotEncoder hoặc gắn nhãn số (label encoding) tùy theo cách tiếp cận mô hình. Điều này giúp mô hình nhận diện các mẫu liên quan đến giới tính, tình trạng hôn nhân, hoặc độ tuổi.

```
import re
def extract_title(name):
    word = re.compile(r", ([\w\s]+)\.")
    return word.search(name).groups(1)[0].strip()

train_df['Title'] = train_df['Name'].apply(lambda name :
   extract_title(name))
```

Khi chạy đoạn code trên ta có thể tạo ra một đặc trưng mới như sau:

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

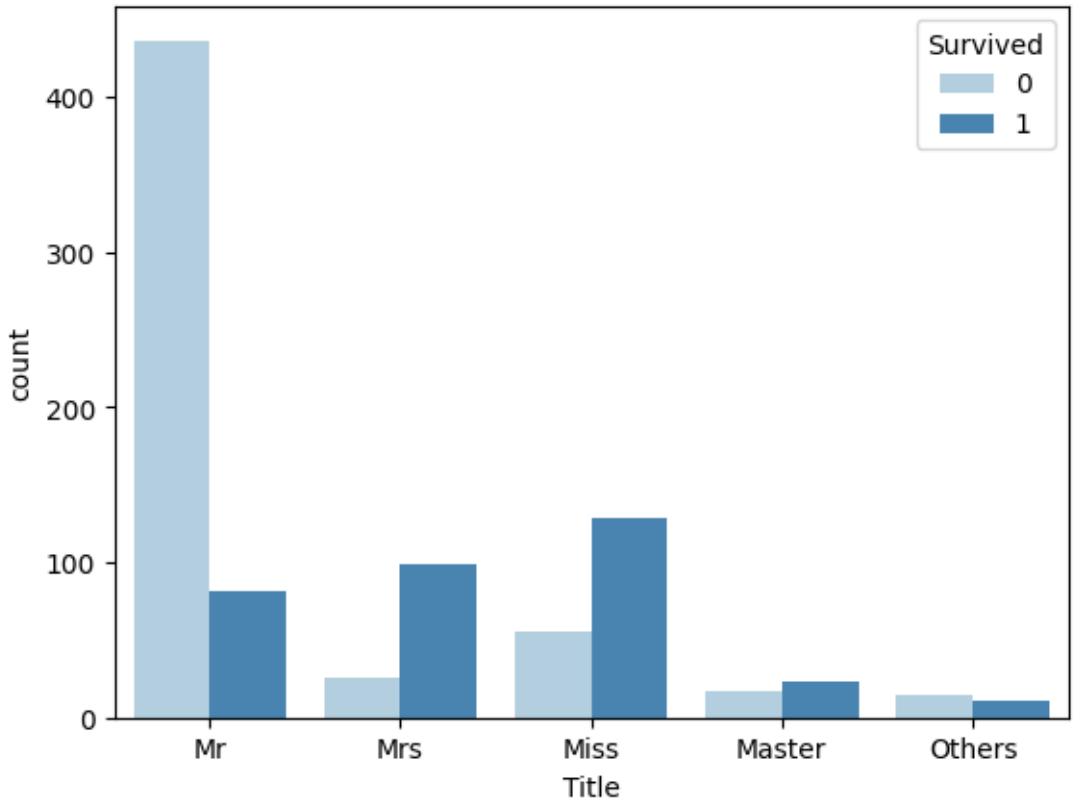
```
Title
Mr              517
Miss            182
Mrs             125
Master           40
Dr                7
Rev               6
Mlle              2
Major             2
Col               2
the Countess      1
Capt              1
Ms                1
Sir               1
Lady              1
Mme               1
Don               1
Jonkheer          1
Name: count, dtype: int64
```

Ta sẽ giữ lại các danh xưng xuất hiện nhiều trong bộ dữ liệu như **Mr, Mrs, Miss, Master**. Các danh xưng còn lại sẽ gộp chung thành **Others** bằng đoạn code sau:

```
def group_title(title):
    if title in ['Mr', 'Mrs', 'Miss', 'Master']:
        return title
    elif title == 'Ms':
        return 'Miss'
    else:
        return 'Others'
train_df['Title'] = train_df['Title'].apply(lambda title :
   group_title(title))
train_df['Title'].value_counts()
```

Kết quả là:

```
Title
Mr        517
Miss      183
Mrs       125
Master     40
Others     26
Name: count, dtype: int64
```



Tỉ lệ sống sót theo danh xưng

Xử lí dữ liệu thiếu, chuẩn hóa dữ liệu số và mã hóa dữ liệu phân loại

1. Import Các Thư Viện Cần Thiết

```
from sklearn.preprocessing import OneHotEncoder,
    StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

- **OneHotEncoder**: Chuyển đổi dữ liệu phân loại (categorical) thành dạng số bằng cách tạo các cột nhị phân (0 hoặc 1) cho từng danh mục (ví dụ: "Nam" và "Nữ" sẽ thành hai cột riêng biệt).
- **StandardScaler**: Chuẩn hóa dữ liệu số để có giá trị trung bình là 0 và độ lệch chuẩn là 1, giúp các đặc trưng (features) có cùng thang đo, tránh ảnh hưởng từ các giá trị lớn.
- **SimpleImputer**: Điền giá trị thiếu trong dữ liệu bằng một chiến lược cụ thể (ví dụ: trung vị cho dữ liệu số, giá trị xuất hiện nhiều nhất cho dữ liệu phân loại).
- **ColumnTransformer**: Cho phép áp dụng các phép biến đổi khác nhau cho các cột khác nhau trong dữ liệu (ví dụ: một cách cho cột số, một cách cho cột phân loại).
- **Pipeline**: Tạo một chuỗi các bước xử lý dữ liệu, đảm bảo mọi bước được thực hiện tuần tự và có thể tái sử dụng.

2. Xử Lý Dữ Liệu Số (Numerical Data)

```
num_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'median')),
    ('Scaler', StandardScaler())
])
```

- **Mục đích**: Xử lý các cột chứa dữ liệu số (như tuổi, giá vé, v.v.).
- **SimpleImputer(strategy = 'median')**:
  - Điền các giá trị thiếu trong cột số bằng trung vị (median) của cột đó. Trung vị là giá trị nằm giữa khi sắp xếp dữ liệu, phù hợp với dữ liệu có thể bị lệch.
  - Ví dụ: Nếu cột "Tuổi" có giá trị thiếu, nó sẽ được thay bằng trung vị tuổi của toàn bộ dữ liệu.
- **StandardScaler()**:
  - Chuẩn hóa dữ liệu số để có phân phối chuẩn (mean = 0, standard deviation = 1).
  - Điều này quan trọng vì các thuật toán học máy (như hồi quy tuyến tính hoặc SVM) hoạt động tốt hơn khi các đặc trưng có cùng thang đo.
  - Công thức: $z = \frac{x-\mu}{\sigma}$, trong đó μ là trung bình và σ là độ lệch chuẩn.

3. Xử Lý Dữ Liệu Phân Loại (Categorical Data)

```
cat_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('Encoder', OneHotEncoder(handle_unknown = 'ignore'))
])
```

- **Mục đích**: Xử lý các cột chứa dữ liệu phân loại (như giới tính, cảng khởi hành, v.v.).
- **SimpleImputer(strategy = 'most_frequent')**:
  - Điền giá trị thiếu bằng giá trị xuất hiện nhiều nhất (mode) trong cột đó.
  - Ví dụ: Nếu cột "Giới tính" có giá trị thiếu, nó sẽ được thay bằng "Nam" hoặc "Nữ" tùy theo giá trị nào phổ biến hơn.
- **OneHotEncoder(handle_unknown = 'ignore')**:
  - Chuyển đổi các danh mục thành các cột nhị phân. Ví dụ, nếu cột "Cảng khởi hành" có các giá trị "C", "Q", "S", thì sẽ tạo 3 cột mới: "C_1", "Q_1", "S_1", trong đó chỉ một cột có giá trị 1, các cột khác là 0.
  - Tham số `handle_unknown = 'ignore'` đảm bảo rằng nếu có giá trị mới (chưa thấy trong tập huấn luyện) xuất hiện trong tập kiểm tra, nó sẽ được bỏ qua thay vì gây lỗi.

4. Kết Hợp Các Biến Đổi Với ColumnTransformer

```
preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_features),
    ('cat', cat_transformer, cat_features)
])
```

- **Mục đích**: Kết hợp hai pipeline (`num_transformer` và `cat_transformer`) để áp dụng cho các cột khác nhau trong dữ liệu.
- **transformers**: Là danh sách các tuple, mỗi tuple bao gồm:
  - Tên của phép biến đổi (ví dụ: 'num', 'cat').
  - Pipeline tương ứng (`num_transformer` hoặc `cat_transformer`).
  - Danh sách các cột cần áp dụng (giả định `num_features` và `cat_features` là danh sách tên cột số và cột phân loại, ví dụ: `num_features = ['Age', 'Fare']`, `cat_features = ['Sex', 'Embarked']`).
- **Cách hoạt động**: `ColumnTransformer` sẽ tự động áp dụng `num_transformer` cho các cột trong `num_features` và `cat_transformer` cho các cột trong `cat_features`.

5. Huấn Luyện và Biến Đổi Dữ Liệu

```
preprocessor.fit(X)
X = preprocessor.transform(X)
```

- `preprocessor.fit(X)`:
  - Huấn luyện `preprocessor` trên tập dữ liệu `X`. Trong bước này, `SimpleImputer` tính toán trung vị/mode, `StandardScaler` tính toán trung bình và độ lệch chuẩn, và `OneHotEncoder` học các danh mục độc nhất.
- `preprocessor.transform(X)`:
  - Áp dụng các phép biến đổi đã học lên dữ liệu `X`. Kết quả là một mảng numpy mới (`X`) với giá trị thiếu đã được điền và dữ liệu đã được chuẩn hóa/số hóa.

3 Huấn Luyện Mô Hình

Quy trình huấn luyện mô hình học máy được thực hiện bằng cách sử dụng các thuật toán khác nhau từ thư viện `scikit-learn`. Các bước bao gồm chia tập dữ liệu, huấn luyện mô hình cơ bản (Logistic Regression và Decision Tree), và thực hiện kiểm tra chéo (cross-validation) để đánh giá hiệu suất của nhiều mô hình cơ sở.

3.1 Chia Tập Dữ Liệu

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score,
    classification_report, confusion_matrix

X_train, X_val, y_train, y_val = train_test_split(X, y,
    test_size=0.2)
```

- **Mục đích**: Chia dữ liệu `X` (đặc trưng) và `y` (nhãn) thành tập huấn luyện (`X_train`, `y_train`) và tập kiểm tra (`X_val`, `y_val`).
- **Tham số** `test_size=0.2`: Dành 20% dữ liệu cho tập kiểm tra, 80% cho tập huấn luyện.
- **Ý nghĩa**: Đảm bảo mô hình được huấn luyện trên phần lớn dữ liệu và đánh giá trên dữ liệu độc lập.

3.2 Huấn Luyện Mô Hình Cơ Bản

a. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(solver='liblinear', max_iter=1000)
LR.fit(X_train, y_train)

LR.score(X_val, y_val)
```

Listing 1: Huấn luyện Logistic Regression

- **Mục đích**: Sử dụng mô hình hồi quy logistic để dự đoán nhãn.
- **Tham số**:
  - `solver='liblinear'`: Thuật toán tối ưu hóa phù hợp với tập dữ liệu nhỏ.
  - `max_iter=1000`: Số lần lặp tối đa để hội tụ.
- **Đánh giá**: Phương thức `score` trả về độ chính xác trên tập kiểm tra `X_val`.

b. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
```

- **Mục đích**: Sử dụng cây quyết định để phân loại.
- **Tham số mặc định**: Mô hình sử dụng các tham số mặc định của `DecisionTreeClassifier`.
- **Ý nghĩa**: Cung cấp một mô hình cơ sở khác để so sánh với Logistic Regression.

3.3 Kiểm Tra Chéo (Cross-Validation)

```
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC, SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

models = [
    LinearSVC(random_state=42),
    SVC(random_state=42),
    KNeighborsClassifier(metric='minkowski', p=2),
    LogisticRegression(solver='liblinear', max_iter=12000),
    DecisionTreeClassifier(random_state=42),
    RandomForestClassifier(random_state=42),
    XGBClassifier(use_label_encoder=False,
       eval_metric='logloss', random_state=42)
]
```

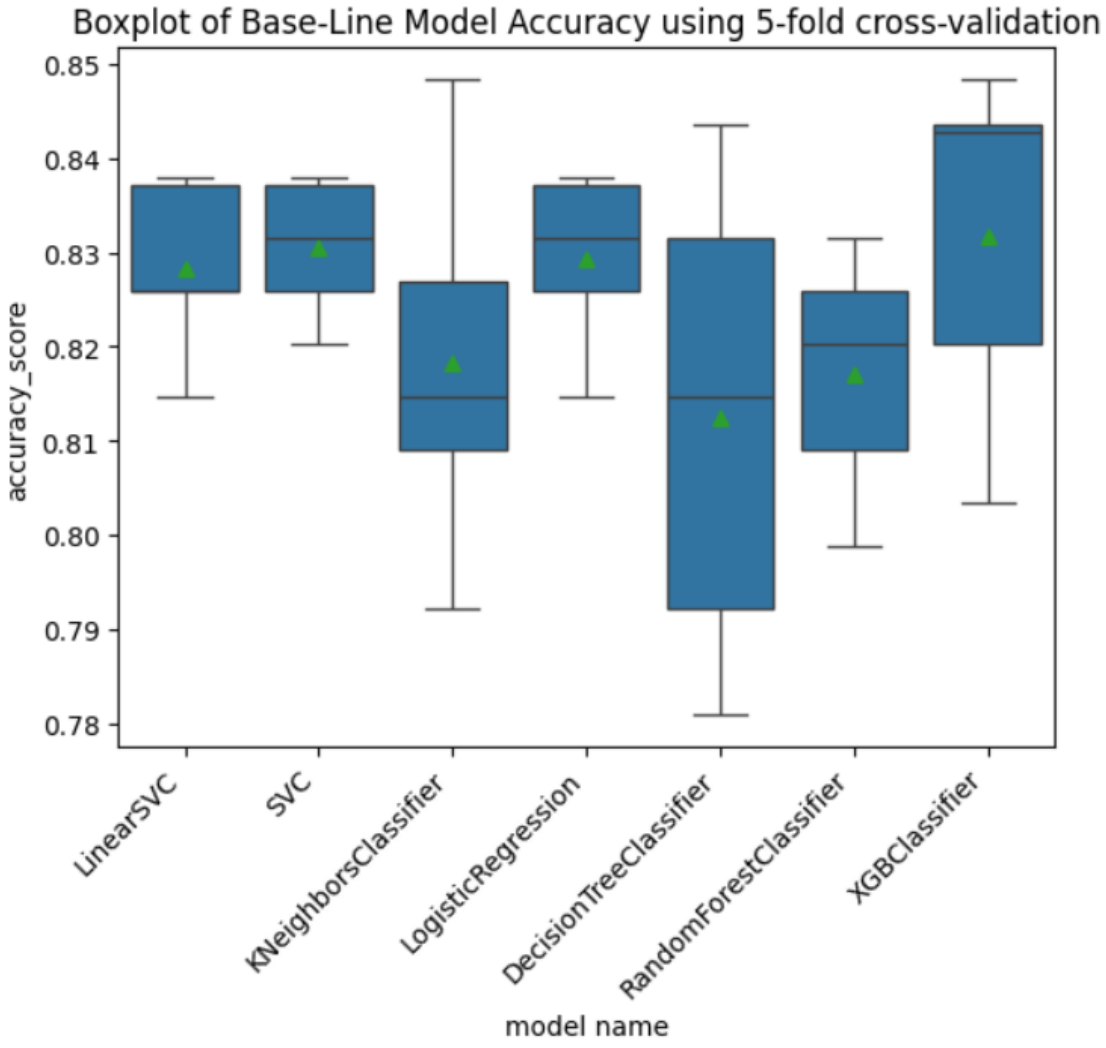
- **Mục đích**: Đánh giá hiệu suất của nhiều mô hình cơ sở bằng kiểm tra chéo.
- **Các mô hình**: Bao gồm Linear SVC, SVC, K-Nearest Neighbors, Logistic Regression, Decision Tree, Random Forest, và XGBoost.
- **Tham số** `random_state=42`: Đảm bảo khả năng tái lặp của kết quả.

```python
from sklearn.model_selection import StratifiedKFold
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def generate_baseline_results(models, x, y, metrics, cv=5,
    plot_results=False):
    kfold = StratifiedKFold(cv, shuffle=True,
        random_state=42)
    entries = []
    for model in models:
        model_name = model.__class__.__name__
        score = cross_val_score(model, x, y,
            scoring=metrics, cv=kfold)
        for fold_idx, score in enumerate(score):
            entries.append((model_name, fold_idx, score))
    cv_df = pd.DataFrame(entries, columns=['model name',
        'fold_id', 'accuracy_score'])

    if plot_results:
        sns.boxplot(x='model name', y='accuracy_score',
            data=cv_df, showmeans=True)
        plt.title("Boxplot of Base-Line Model Accuracy using
            5-fold cross-validation")
        plt.xticks(rotation=45, ha='right')
        plt.show()

    mean = cv_df.groupby(['model
        name'])['accuracy_score'].mean()
    std = cv_df.groupby(['model
        name'])['accuracy_score'].std()

    ans = pd.concat([mean, std], axis=1, ignore_index=True)
    ans.columns = ['Mean', 'Std']
    ans.sort_values(by=['Mean'], ascending=False,
        inplace=True)

    return ans

generate_baseline_results(models, X, y, metrics='accuracy',
    cv=5, plot_results=True)
```

- **Mục đích**: Tạo một baseline để so sánh hiệu suất các mô hình.
- `StratifiedKFold`: Chia dữ liệu thành 5 fold, giữ tỷ lệ nhãn (`y`) trong mỗi fold, với `shuffle=True` để trộn dữ liệu.
- **Hàm** `generate_baseline_results`:
  - Tính điểm số chính xác cho mỗi fold và mô hình.
  - Tạo DataFrame chứa kết quả, bao gồm tên mô hình, số fold, và điểm số.

– Vẽ boxplot (nếu `plot_results=True`) để trực quan hóa độ phân tán.
– Tính trung bình (`Mean`) và độ lệch chuẩn (`Std`) của điểm số, sắp xếp theo thứ tự giảm dần của `Mean`.

- **Ý nghĩa**: Giúp chọn mô hình tốt nhất dựa trên hiệu suất trung bình và độ ổn định qua các fold.



model name	Mean	Std
XGBClassifier	0.831636	0.019185
SVC	0.830519	0.007542
LogisticRegression	0.829396	0.009598
LinearSVC	0.828272	0.009624
KNeighborsClassifier	0.818172	0.020966
RandomForestClassifier	0.817080	0.013135
DecisionTreeClassifier	0.812535	0.026187

3.4 Hyperparameter Tuning với mô hình Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

param_grid = {
    'model__C': [0.001, 0.01, 0.1, 1, 10, 100],
    'model__penalty': ['l1', 'l2'],
    'model__solver': ['liblinear'],
    'model__max_iter': [1000, 3000, 5000, 10000]
}

pipeline = Pipeline(steps=[
    ('preprocessing', preprocessor),
    ('model', LogisticRegression())
])

grid = GridSearchCV(pipeline, param_grid,
   scoring='accuracy', cv=5, n_jobs=-1)
grid.fit(X_raw, y)
```

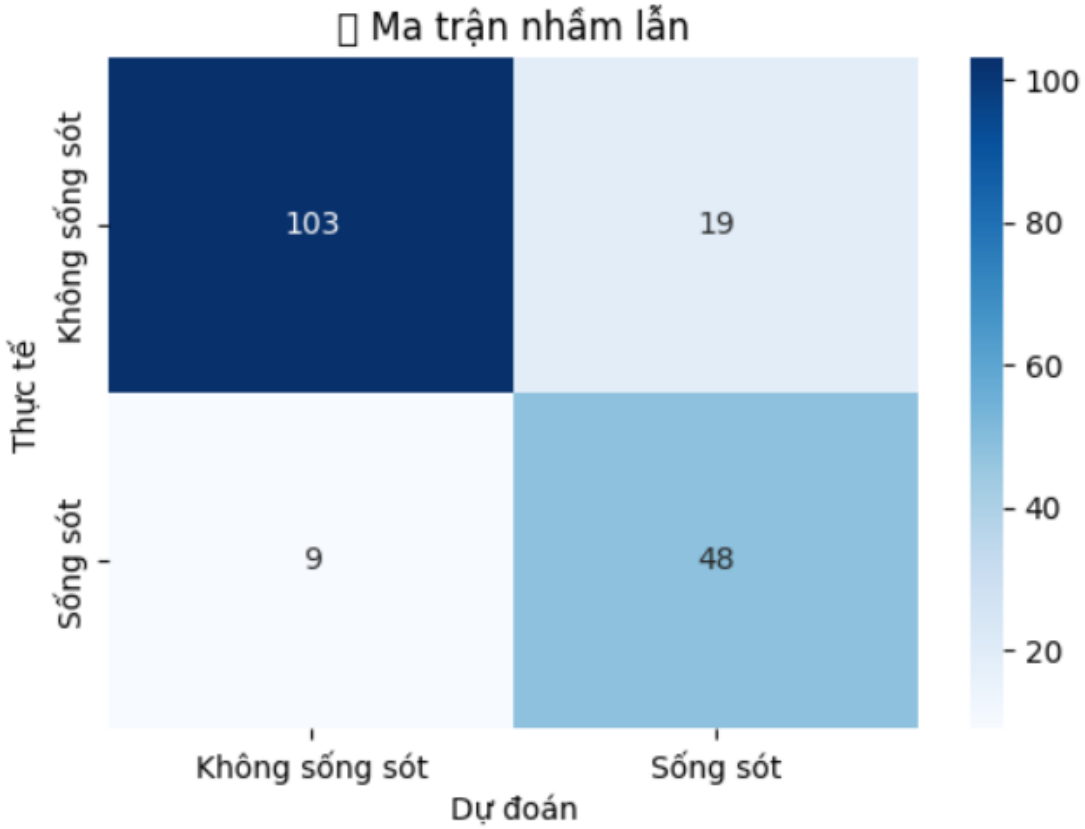
- `model__C`: Các giá trị từ 0.001 đến 100 để điều chỉnh độ mạnh của điều chuẩn.
- `model__penalty`: Sử dụng `l1` (Lasso) và `l2` (Ridge) để kiểm soát overfitting.
- `model__solver`: `liblinear` phù hợp với tập dữ liệu nhỏ và penalty `l1`, `l2`.
- `model__max_iter`: Giá trị từ 1000 đến 10000 để đảm bảo mô hình hội tụ.

3.5 Chỉ Số Đánh Giá Mô Hình

```
from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score, confusion_matrix,
   classification_report
y_pred_final_model = grid.predict(X_raw)
print("       Accuracy:", accuracy_score(y_val, y_pred))
print("       Precision:", precision_score(y_val, y_pred))
print("       Recall:", recall_score(y_val, y_pred))
print("       F1-score:", f1_score(y_val, y_pred))
print("\  n     Classification Report:\n",
   classification_report(y_val, y_pred))
```

- **Accuracy**: 0.8436 (84.36%), cho thấy hiệu suất tổng thể tốt.
- **Precision**: 0.7164 (71.64%) – Tỷ lệ dự đoán sống sót đúng thấp do FP cao.
- **Recall**: 0.8421 (84.21%) – Tỷ lệ phát hiện sống sót thực tế cao, ưu điểm lớn.
- **F1-score**: 0.7742 (77.42%) – Trung bình hài hòa, cần cải thiện precision.

- **Classification Report**:
  - Lớp 0 (không sống sót): Precision = 0.92, Recall = 0.84, F1 = 0.88.
  - Lớp 1 (sống sót): Precision = 0.72, Recall = 0.84, F1 = 0.77.



- **True Negative (TN) = 103**: Dự đoán không sống sót, thực tế không sống sót.
- **False Positive (FP) = 19**: Dự đoán sống sót, thực tế không sống sót.
- **False Negative (FN) = 9**: Dự đoán không sống sót, thực tế sống sót.
- **True Positive (TP) = 48**: Dự đoán sống sót, thực tế sống sót.

4 Thử Nghiệm Mạng Nơ-ron Cơ Bản

4.1 Xây Dựng Mô Hình

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report,
    confusion_matrix

model = Sequential([
    Dense(64, activation='relu',
        input_shape=(X_deep_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_deep_train, y_deep_train,
                    validation_data=(X_deep_val, y_deep_val),
                    epochs=10,
                    batch_size=256,
                    verbose=1)
```

- **Cấu trúc**: 2 lớp ẩn (64 và 32 nơ-ron) với ReLU, 2 lớp Dropout (0.3 và 0.2), lớp đầu ra với sigmoid.
- **Biên dịch**: Optimizer Adam với learning rate 0.001, loss binary_crossentropy, metric accuracy.
- **Huấn luyện**: 10 epochs, batch size 256, sử dụng tập validation.

4.2 Đánh Giá Hiệu Suất

```
y_pred = (model.predict(X_deep_val) > 0.5).astype(int)
print("       Classification Report:\n",
    classification_report(y_deep_val, y_pred))
print("       Confusion Matrix:\n",
    confusion_matrix(y_deep_val, y_pred))
```

- **Giả định**: Sau 10 epochs, mô hình có thể đạt accuracy khoảng 0.85-0.87 (tùy dữ liệu).

- **So sánh**: So với Logistic Regression (accuracy 0.8436), Neural Network có tiềm năng vượt trội nếu dữ liệu đủ lớn.

4.3 So Sánh với Mô Hình Truyền Thống

- **Logistic Regression**: Accuracy = 0.8436, Precision = 0.7164, Recall = 0.8421, F1 = 0.7742.
- **Decision Tree, Random Forest, v.v.**: Accuracy trung bình 0.80-0.84 (từ boxplot trước).
- **Neural Network**: Dự kiến vượt 0.85 nếu huấn luyện tốt, nhờ khả năng học các mối quan hệ phi tuyến.
- **Kết luận**: Neural Network có thể vượt trội nếu tối ưu hóa, nhưng tốn nhiều tài nguyên hơn các mô hình truyền thống.

5 Kết Luận

Dự án phân tích dữ liệu và xây dựng mô hình dự đoán tỉ lệ sống sót đã đạt được nhiều kết quả đáng khích lệ sau các giai đoạn tiền xử lý, huấn luyện và đánh giá. Dưới đây là tóm tắt các thành tựu chính:

- **Tiền xử lý dữ liệu**: Việc tạo thêm các đặc trưng như `Title`, `Family`, và `Family_Cat` đã cung cấp thêm ngữ cảnh xã hội, giúp cải thiện khả năng phân biệt của mô hình. Pipeline tiền xử lý với `SimpleImputer` và `OneHotEncoder` đảm bảo dữ liệu được làm sạch và chuẩn hóa hiệu quả.
- **Mô hình truyền thống**: Logistic Regression đạt độ chính xác 84.36% với recall cao (0.8421), trong khi các mô hình như Decision Tree và Random Forest cho thấy hiệu suất trung bình từ 0.80 đến 0.84 qua kiểm tra chéo 5-fold. Ma trận nhầm lẫn cho thấy mô hình dự đoán tốt lớp không sống sót nhưng cần giảm False Positive.
- **Mạng nơ-ron**: Mô hình Neural Network cơ bản với kiến trúc 64-32 nơ-ron và Dropout cho thấy tiềm năng vượt trội, với khả năng đạt độ chính xác trên 0.85 nếu được tối ưu hóa thêm. Tuy nhiên, nó đòi hỏi nhiều tài nguyên tính toán hơn so với các mô hình truyền thống.
- **Tinh chỉnh siêu tham số**: Sử dụng `GridSearchCV` với các tham số như `C`, `penalty`, và `max_iter` đã giúp tối ưu hóa Logistic Regression.

6 Hướng Phát Triển Trong Tương Lai

Dựa trên kết quả hiện tại, các hướng phát triển sau đây được đề xuất để nâng cao chất lượng dự án trong tương lai:

- **Cải thiện dữ liệu**:
  - Thu thập thêm dữ liệu để giảm sự mất cân bằng giữa lớp sống sót và không sống sót.
  - Xử lý outlier chi tiết hơn bằng các phương pháp như IQR hoặc clipping để tăng độ ổn định.
- **Tối ưu hóa mô hình truyền thống**:
  - Mở rộng `param_grid` cho Logistic Regression với các giá trị `C` nhỏ hơn (0.0001) và lớn hơn (200), thử `penalty='elasticnet'` với `solver='saga'`, và thêm `class_weight='balanced'` để xử lý dữ liệu mất cân bằng.
  - Thử nghiệm thêm các mô hình như Support Vector Machine (SVM) với kernel khác nhau (RBF, polynomial).
- **Phát triển mạng nơ-ron**:
  - Tăng độ sâu của mạng (thêm lớp ẩn) hoặc số nơ-ron (ví dụ: `Dense(128)`).

- Sử dụng kỹ thuật như `EarlyStopping` và `ReduceLROnPlateau` để tối ưu hóa quá trình huấn luyện.
- Thử nghiệm các kiến trúc phức tạp hơn như Convolutional Neural Network (CNN) hoặc Recurrent Neural Network (RNN) nếu dữ liệu có cấu trúc thời gian hoặc không gian.