ĐẠI HỌC HUẾ

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

BÁO CÁO ĐỒ ÁN

Học kỳ I, năm học 2023 - 2024

Học phần:

Học máy

Số phách
(Do hội đồng chấm thi ghi)

Thừa Thiên Huế, tháng 1 năm 2024

ĐẠI HỌC HUẾ

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

BÁO CÁO ĐỒ ÁN

Học kỳ I, năm học 2023 - 2024

Học phần:

Học máy

Giảng viên hướng dẫn: TS. Hoàng Hữu Trung

Lớp: Khoa học dữ liệu và trí tuệ nhân tạo K3

Sinh viên thực hiện:

Hồ Tăng Nhật Hiếu – 22E1010006

Văn Khiêm Chương – 22E1020002

Số phách
(Do hội đồng chấm thi ghi)

Thừa Thiên Huế, tháng 1 năm 2024

LỜI CÁM ƠN

Kính gửi thầy Hoàng Hữu Trung chúng em xin gửi lời cám ơn sâu sắc nhất đến thầy về sự hướng dẫn và hỗ trợ mà thầy đã dành cho đồ án của chúng em. Đây là một hành trình học tập đầy thách thức và ý nghĩa, và chúng em cảm thấy may mắn được có thầy là người hướng dẫn.

Thầy đã không ngừng truyền đạt kiến thức chuyên sâu, sự tận tâm của thầy đã giúp chúng em vượt qua những khó khăn, cũng như mở mang tầm nhìn và ý thức trong lĩnh vực chuyên ngành của mình.

Chúng em muốn bày tỏ lòng biết ơn đặc biệt đến những góp ý chi tiết và xây dựng của thầy, giúp chúng em hoàn thiện đồ án một cách toàn diện hơn. Sự quan tâm và tận tâm của thầy không chỉ giúp chúng em hoàn thành đồ án mà còn giúp chúng em phát triển kỹ năng và tư duy nghiên cứu.

Chúng em cảm thấy tự hào và biết ơn vì đã có một người hướng dẫn như thầy, người luôn sẵn lòng chia sẻ kiến thức và kinh nghiệm.

Xin chân thành cảm ơn thầy một lần nữa vì sự hỗ trợ và sự dẫn dắt trong suốt thời gian qua. Chúng em hi vọng sẽ tiếp tục nhận được sự hướng dẫn tận tâm của thầy trong tương lai.

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

MỤC LỤC

CÂU 1: HỌC CÓ GIÁM SÁT

I. Giới thiệu dữ liệu

Bộ dữ liệu dự đoán rủi ro bệnh tim mạch.

Gồm các cột:

General_Health: Trạng thái sức khỏe tổng quát.

Checkup: Thông tin về việc kiểm tra sức khỏe định kỳ hoặc các cuộc kiểm tra y tế.

Exercise: Hoạt động thể dục, có thể là số lần tập luyện trong một khoảng thời gian cụ thể hoặc mức độ hoạt động.

Heart_Disease: Thông tin về bệnh tim mạch, có thể là có hoặc không có.

Skin_Cancer: Thông tin về ung thư da, có thể là có hoặc không có.

Other_Cancer: Thông tin về bất kỳ loại ung thư khác.

Depression: Trạng thái tâm lý, có thể là có hoặc không có triệu chứng của bệnh trầm cảm.

Diabetes: Thông tin về bệnh tiểu đường, có thể là có hoặc không có.

Arthritis: Thông tin về việc có hoặc không có bệnh viêm khớp.

Sex: Giới tính.

Age_Category: Nhóm độ tuổi.

Height_(cm): Chiều cao, được đo bằng đơn vị centimet.

Weight_(kg): Cân nặng, được đo bằng đơn vị kilogram.

BMI (Body Mass Index): Chỉ số khối cơ thể, được tính từ chiều cao và cân nặng để đánh giá mức độ thừa cân hoặc thiếu cân.

Smoking_History: Lịch sử hút thuốc, có thể là "Có" hoặc "Không".

Alcohol_Consumption: Mức tiêu thụ rượu.

Fruit_Consumption: Mức tiêu thụ hoa quả.

Green_Vegetables_Consumption: Mức tiêu thụ rau xanh.

FriedPotato_Consumption: Mức tiêu thụ khoai tây chiên.

II. Đọc dữ liệu

1. Nhập thư viện

Việc import thư viện là một bước quan trọng trong lập trình, giúp chúng ta sử dụng các chức năng, lớp, hoặc phương thức đã được định nghĩa sẵn trong thư viện đó. Khi bắt đầu viết code, chúng ta thường cần sử dụng các tính năng mà ngôn ngữ lập trình đó không cung cấp sẵn, và đó là lúc thư viện đến đầu tiên.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from lazypredict.Supervised import LazyClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

2. Đọc dữ liệu

Khi làm việc với dữ liệu được lưu trữ trong các tệp tin Excel, việc đọc dữ liệu trở nên quan trọng để nắm bắt thông tin và thực hiện các phân tích hoặc xử lý tiếp theo.

```
df = pd.read_excel('CVD.xlsx')
df.head()
```

Kết quả:

	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diabetes	Arthritis	Sex	Age_Category	Height_(cm)	Weight_(kg)	BMI	Smoking_History
0	Poor	Within the past 2 years	No	No	No	No	No	No	Yes	Female	70-74	150	32.66	14.54	Yes
1	Very Good	Within the past year	No	Yes	No	No	No	Yes	No	Female	70-74	165	77.11	28.29	No
2	Very Good	Within the past year	Yes	No	No	No	No	Yes	No	Female	60-64	163	88.45	33.47	No
3	Poor	Within the past year	Yes	Yes	No	No	No	Yes	No	Male	75-79	180	93.44	28.73	No
4	Good	Within the past year	No	No	No	No	No	No	No	Male	80+	191	88.45	24.37	Yes

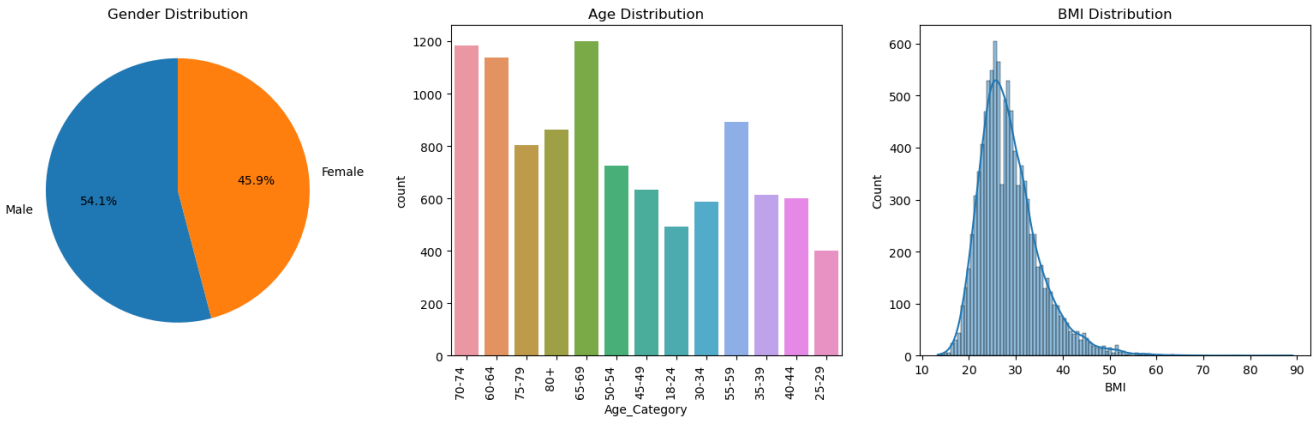
Hình 1: Kết quả hiển thị DataFrame gốc

III. Trực quan một số thuộc tính trong DataFrame

Thực hiện vẽ trực quan hóa dữ liệu là một bước quan trọng trong quá trình phân tích dữ liệu, giúp chúng ta hiểu rõ hơn về xu hướng, mối quan hệ của dữ liệu. Trực quan hóa không chỉ làm cho thông tin trở nên dễ hiểu hơn, mà còn giúp tạo ra các hình ảnh, hỗ trợ trong việc giải thích và trình bày kết quả.

```
fig, ax = plt.subplots(1,3,figsize=(20, 5))
# Tròn
ax[0].pie(df['Sex'].value_counts(), labels = ['Male', 'Female'],
autopct='%1.1f%%', startangle=90)
ax[0].set_title('Gender Distribution')
# Cột
sns.countplot(x = 'Age_Category', data = df, ax = ax[1]).set_title('Age
Distribution')
# Histogram
ax[1].set_xticklabels(ax[1].get_xticklabels(), rotation=90, ha='right')
sns.histplot(x = 'BMI', data = df, ax = ax[2], kde = True).set_title('BMI
Distribution')
```

Kết quả:



Hình 2: Kết quả trực quan một số thuộc tính lần 1

Ba biểu đồ trên giải thích thông tin của bệnh nhân trong tập dữ liệu.

Từ biểu đồ hình tròn, có thể thấy rõ phần lớn bệnh nhân là nam với 54.1%, tiếp theo là nữ với 45.9%.

Nhìn vào sự phân bổ độ tuổi, chúng chúng em biết rằng phần lớn bệnh nhân đều trên 40 tuổi, điều này có nghĩa là tập dữ liệu nghiêng về những bệnh nhân lớn tuổi.

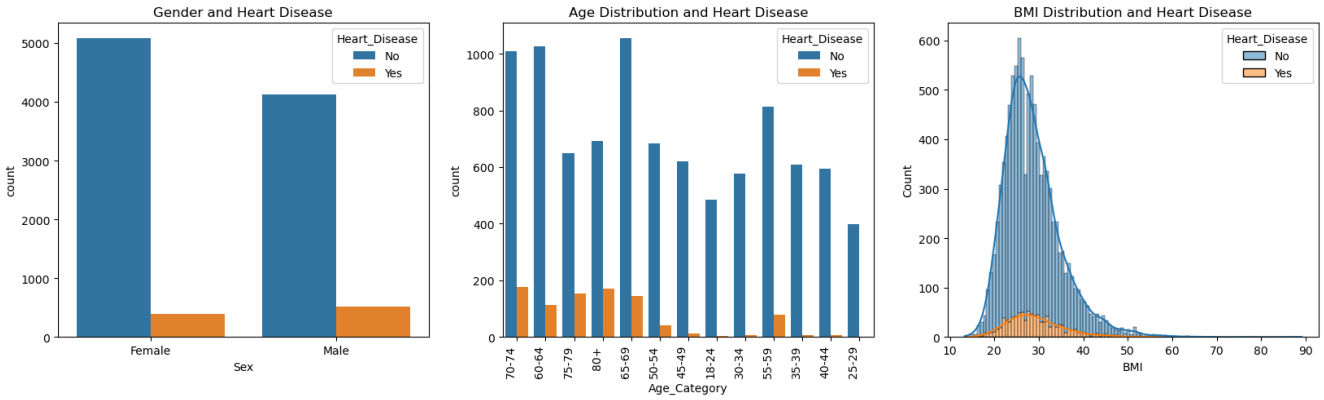
Biểu đồ BMI cho thấy chỉ số BMI của phần lớn bệnh nhân nằm trong khoảng từ 25 đến 30.

Điều này có nghĩa phần lớn bệnh nhân đều thừa cân.

Kiểm tra bệnh tim

```
# Trực quan mối quan hệ giữa các thuộc tính 'Sex', 'Age_Category', 'BMI' và
'Heart_Disease'
fig, ax = plt.subplots(1,3,figsize=(20, 5))
# Đồ thị cột (Count plot) cho mối quan hệ giữa 'Gender' và 'Heart_Disease'
sns.countplot(x = 'Sex', data = df, hue  = 'Heart_Disease', ax =
ax[0]).set_title('Gender and Heart Disease')
# Đồ thị cột (Count plot) cho mối quan hệ giữa 'Age_Category' và 'Heart_Disease':
sns.countplot(x = 'Age_Category', data = df, ax = ax[1], hue =
'Heart_Disease').set_title('Age Distribution and Heart Disease')
ax[1].set_xticklabels(ax[1].get_xticklabels(), rotation=90, ha='right')
# Đồ thị histogram (Histogram) cho mối quan hệ giữa 'BMI' và 'Heart_Disease'
sns.histplot(x = 'BMI', data = df, ax = ax[2], kde = True, hue = 'Heart_Disease',
multiple = 'stack').set_title('BMI Distribution and Heart Disease')
```

Kết quả:



Hình 3: Kết quả trực quan một số thuộc tính lần 2

Đầu tiên nhìn vào biểu đồ Giới tính, chúng ta có thể thấy rằng, nam giới mắc bệnh tim hơn so với nữ giới.

Trong biểu đồ thứ hai có thể thấy rằng những bệnh nhân có độ tuổi trên 50 tuổi có tỷ lệ mắc bệnh tim cao hơn so với các nhóm tuổi khác. Điều này có nghĩa là bệnh nhân lớn tuổi dễ mắc bệnh tim mạch hơn và nguy cơ mắc bệnh tim mạch tăng theo tuổi.

Biểu đồ thứ ba về chỉ số BMI cho thấy, những bệnh nhân có chỉ số BMI từ 25-30, tức là thừa cân, có nguy cơ mắc bệnh tim cao hơn.

IV. Xử lý dữ liệu

1. Các bước kiểm tra

- Kiểm tra thông tin của DataFrame.

```
df.info()
```

Kết quả:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10133 entries, 0 to 10132
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   General_Health                10133 non-null  object
 1   Checkup                       10133 non-null  object
 2   Exercise                      10133 non-null  object
 3   Heart_Disease                 10133 non-null  object
 4   Skin_Cancer                   10133 non-null  object
 5   Other_Cancer                  10133 non-null  object
 6   Depression                    10133 non-null  object
 7   Diabetes                      10133 non-null  object
 8   Arthritis                     10133 non-null  object
 9   Sex                           10133 non-null  object
 10  Age_Category                  10133 non-null  object
 11  Height_(cm)                   10133 non-null  int64
 12  Weight_(kg)                   10133 non-null  float64
 13  BMI                           10133 non-null  float64
 14  Smoking_History               10133 non-null  object
 15  Alcohol_Consumption           10133 non-null  int64
 16  Fruit_Consumption             10133 non-null  int64
 17  Green_Vegetables_Consumption  10133 non-null  int64
 18  FriedPotato_Consumption       10133 non-null  int64
dtypes: float64(2), int64(5), object(12)
memory usage: 1.5+ MB
```

Hình 4: Kết quả thông tin DataFrame

- Kiểm tra hình dạng của DataFrame.

```
df.shape
```

Kết quả:

```
(10133, 19)
```

Hình 5: Kết quả hình dạng DataFrame

Có tổng cộng 10133 hàng dữ liệu và 19 cột.

- Kiểm tra giá trị thiếu.

```
df.isnull().sum()
```

Kết quả:

```
General_Health                  0
Checkup                         0
Exercise                        0
Heart_Disease                   0
Skin_Cancer                     0
Other_Cancer                    0
Depression                      0
Diabetes                        0
Arthritis                       0
Sex                             0
Age_Category                    0
Height_(cm)                     0
Weight_(kg)                     0
BMI                             0
Smoking_History                 0
Alcohol_Consumption             0
Fruit_Consumption               0
Green_Vegetables_Consumption    0
FriedPotato_Consumption         0
dtype: int64
```

Hình 6: Kết quả giá trị thiếu DataFrame

Kết quả cho thấy không có dữ liệu thiếu trong tập dữ liệu.

- Kiểm tra kiểu dữ liệu.

```
df.dtypes
```

Kết quả:

```
General_Health                  object
Checkup                         object
Exercise                        object
Heart_Disease                   object
Skin_Cancer                     object
Other_Cancer                    object
Depression                      object
Diabetes                        object
Arthritis                       object
Sex                             object
Age_Category                    object
Height_(cm)                      int64
Weight_(kg)                    float64
BMI                            float64
Smoking_History                 object
Alcohol_Consumption              int64
Fruit_Consumption                int64
Green_Vegetables_Consumption     int64
FriedPotato_Consumption          int64
dtype: object
```

Hình 7: Kết quả kiểm tra kiểu dữ liệu DataFrame

- Kiểm tra các hàng trùng lặp.

```
df.duplicated().sum()
```

Kết quả:

```
0
```

Hình 8: Kết quả kiểm tra các hàng trùng lặp

Kết quả bằng 0 cho thấy không có hàng bị trùng lặp.

- Đếm số lượng giá trị duy nhất của DataFrame.

```
df.nunique()
```

Kết quả:

```
General_Health                  5
Checkup                         5
Exercise                        2
Heart_Disease                   2
Skin_Cancer                     2
Other_Cancer                    2
Depression                      2
Diabetes                        4
Arthritis                       2
Sex                             2
Age_Category                   13
Height_(cm)                    43
Weight_(kg)                   282
BMI                          1524
Smoking_History                 2
Alcohol_Consumption            29
Fruit_Consumption              40
Green_Vegetables_Consumption   43
FriedPotato_Consumption        38
dtype: int64
```

Hình 9: Kết quả số lượng giá trị duy nhất DataFrame

2. Xóa cột không cần thiết

Tập dữ liệu có các cột Height_(cm), Weight_(kg) và BMI.

Tuy nhiên, cột BMI được tính dựa trên 2 cột Height_(cm), Weight_(kg). Do đó sẽ bị loại khỏi tập dữ liệu và in lại thông tin dữ liệu.

```
df.drop(columns=['Weight_(kg)', 'Height_(cm)'], inplace=True)
df.info()
```

Kết quả:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10133 entries, 0 to 10132
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   General_Health               10133 non-null  object
 1   Checkup                      10133 non-null  object
 2   Exercise                     10133 non-null  object
 3   Heart_Disease                10133 non-null  object
 4   Skin_Cancer                  10133 non-null  object
 5   Other_Cancer                 10133 non-null  object
 6   Depression                   10133 non-null  object
 7   Diabetes                     10133 non-null  object
 8   Arthritis                    10133 non-null  object
 9   Sex                          10133 non-null  object
 10  Age_Category                 10133 non-null  object
 11  BMI                          10133 non-null  float64
 12  Smoking_History              10133 non-null  object
 13  Alcohol_Consumption          10133 non-null  int64
 14  Fruit_Consumption            10133 non-null  int64
 15  Green_Vegetables_Consumption 10133 non-null  int64
 16  FriedPotato_Consumption      10133 non-null  int64
dtypes: float64(1), int64(4), object(12)
memory usage: 1.3+ MB
```

Hình 10: Thông tin DataFrame sau khi xóa cột

Cột Height_(cm), Weight_(kg) đã được xóa khỏi tập dữ liệu.

3. Loại bỏ giá trị nhiễu

Loại bỏ các giá trị ngoại lệ từ dữ liệu để cải thiện chất lượng và độ chính xác của mô hình học máy. Giảm ảnh hưởng của giá trị ngoại lệ đối với các phân tích thống kê và học máy. Cải thiện tính ổn định của mô hình và kết quả phân tích.

```
# Loại bỏ các giá trị ngoại lệ (outliers)
cols  = ['BMI', 'Alcohol_Consumption', 'Fruit_Consumption',
'Green_Vegetables_Consumption', 'FriedPotato_Consumption']

# Tính các giá trị Q1 và Q3
Q1 = df[cols].quantile(0.25)
Q3 = df[cols].quantile(0.75)

# Tính phạm vi nội tâm (IQR) cho mỗi cột
IQR = Q3 - Q1

# Xác định ngưỡng cho việc xác định ngoại lệ
threshold = 1.5

# Tìm các chỉ mục của các giá trị ngoại lệ trong DataFrame
index = np.where((df[cols] < (Q1 - threshold * IQR)) | (df[cols] > (Q3 +
threshold * IQR)))[0]

# Loại bỏ
df = df.drop(df.index[index])
```

4. Chuyển đổi dữ liệu

Chuyển đổi giá trị trong cột 'Diabetes' của tập dữ liệu từ các giá trị ban đầu thành các giá trị mới.

'No, pre-diabetes or borderline diabetes' sẽ được chuyển thành 'Pre-Diabetes'.

'Yes, but female told only during pregnancy' sẽ được chuyển thành 'Gestational Diabetes'.

'Yes' sẽ được giữ nguyên là 'Yes'.

'No' sẽ được giữ nguyên là 'No'.

```
df['Diabetes'] = df['Diabetes'].map({'No, pre-diabetes or borderline diabetes':
'Pre-Diabetes' ,'Yes, but female told only during pregnancy' : 'Gestational
Diabetes', 'Yes': 'Yes', 'No': 'No'})
```

In ra các cột phân loại.

```
categorical_columns = df.select_dtypes(include=['object', 'category']).columns

print("Categorical Columns:")
print(categorical_columns)
```

Kết quả:

```
Categorical Columns:
Index(['General_Health', 'Checkup', 'Exercise', 'Heart_Disease', 'Skin_Cancer',
       'Other_Cancer', 'Depression', 'Diabetes', 'Arthritis', 'Sex',
       'Age_Category', 'Smoking_History'],
      dtype='object')
```

Hình 11: Kết quả in ra các cột phân loại

Chuyển đổi các cột dạng phân loại đó thành các số nguyên.

```
cols =
['General_Health','Checkup','Exercise','Heart_Disease','Skin_Cancer','Other_Cance
r','Depression','Diabetes','Arthritis', 'Sex',  'Age_Category',
'Smoking_History']

le = LabelEncoder()

for i in cols:
    le.fit(df[i])
    df[i] = le.transform(df[i])
    print(i, df[i].unique())
```

for i in cols: Lặp qua từng tên cột trong danh sách cols.

le.fit(df[i]): Fit ‘LabelEncoder’ trên cột hiện tại để xác định các giá trị chuỗi và gán chúng với các số nguyên.

df[i] = le.transform(df[i]): Thực hiện chuyển đổi dữ liệu của cột hiện tại từ chuỗi sang số nguyên.

print(i, df[i].unique()): In ra các giá trị duy nhất của cột sau khi chuyển đổi, giúp kiểm tra xem quá trình chuyển đổi đã diễn ra.

Kết quả:

```
General_Health [3 4 2 1 0]
Checkup [2 4 0 3 1]
Exercise [0 1]
Heart_Disease [0 1]
Skin_Cancer [0 1]
Other_Cancer [0 1]
Depression [0 1]
Diabetes [1 3 2 0]
Arthritis [1 0]
Sex [0 1]
Age_Category [10  8 11 12  9  6  5  2  7  0  3  4  1]
Smoking_History [1 0]
```

Hình 12: Kết quả chuyển đổi dữ liệu

In lại các dòng đầu của DataFrame để kiểm tra.

```
df.head()
```

Kết quả:

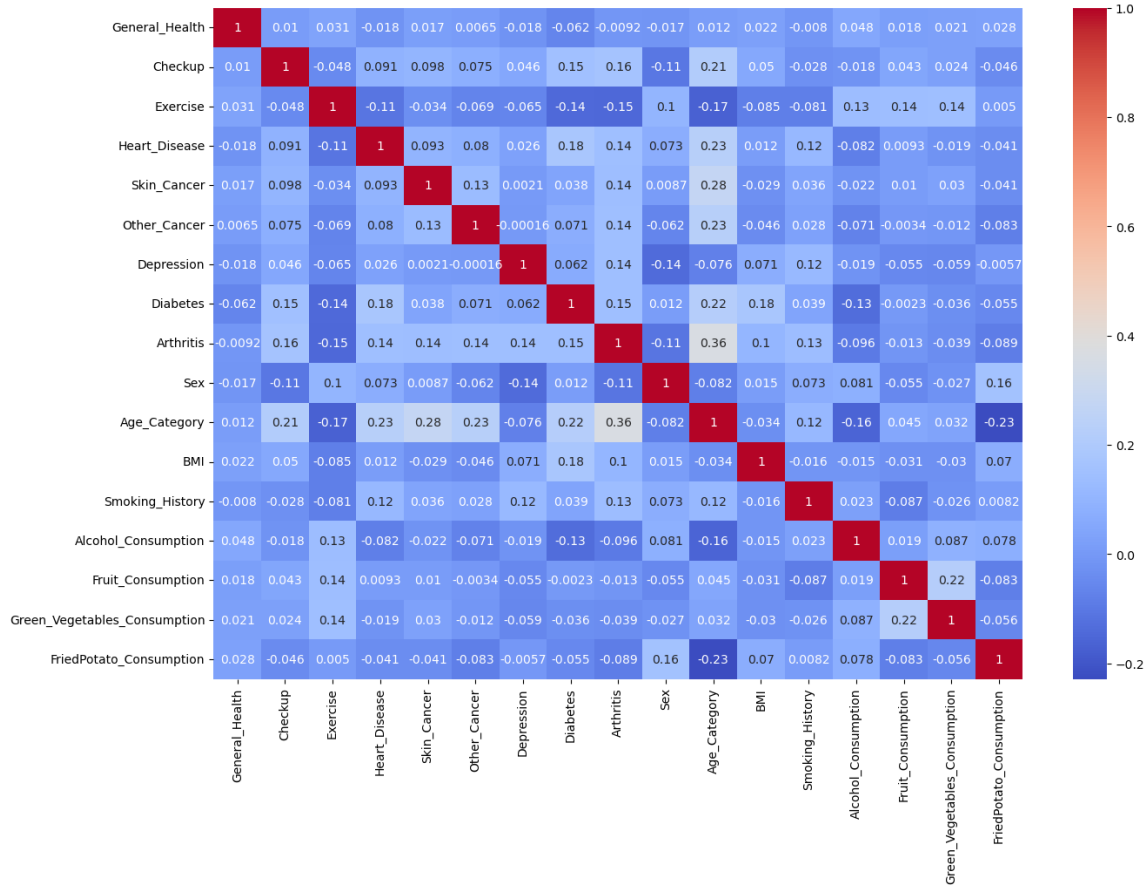
	General_Health	Checkup	Exercise	Heart_Disease	Skin_Cancer	Other_Cancer	Depression	Diabetes	Arthritis	Sex	Age_Category	BMI	Smoking_History	Alcohol_Consumption
0	3	2	0	0	0	0	0	1	1	0	10	14.54	1	0
1	4	4	0	1	0	0	0	3	0	0	10	28.29	0	0
2	4	4	1	0	0	0	0	3	0	0	8	33.47	0	4
3	3	4	1	1	0	0	0	3	0	1	11	28.73	0	0
4	2	4	0	0	0	0	0	1	0	1	12	24.37	1	0

Hình 13: Kết quả hiển thị DataFrame sau khi chuyển đổi

Vẽ biểu đồ heatmap, Biểu đồ heatmap được sử dụng để trực quan hóa mối quan hệ tương quan giữa các biến trong DataFrame.

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), annot = True, cmap = 'coolwarm')
```

Kết quả:



Hình 14: Biểu đồ heatmap trực quan mối quan hệ tương quan giữa các biến

V. Mô hình

Chia tập dữ liệu thành hai phần: một tập dữ liệu huấn luyện (train) và một tập dữ liệu kiểm thử (test).

```
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns =
['Heart_Disease']), df['Heart_Disease'], test_size = 0.2, random_state = 0)
```

Sử dụng Lazyprect để dự đoán mô hình tối ưu.

```
clf = LazyClassifier()
models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
print(predictions)
```

Kết quả:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	\
NearestCentroid	0.71	0.72	0.72	0.76	
GaussianNB	0.81	0.63	0.63	0.82	
QuadraticDiscriminantAnalysis	0.83	0.63	0.63	0.84	
DecisionTreeClassifier	0.83	0.60	0.60	0.83	
BernoulliNB	0.87	0.57	0.57	0.84	
LabelSpreading	0.83	0.56	0.56	0.82	
LabelPropagation	0.83	0.56	0.56	0.82	
Perceptron	0.87	0.56	0.56	0.84	
ExtraTreeClassifier	0.83	0.56	0.56	0.82	
PassiveAggressiveClassifier	0.87	0.55	0.55	0.84	
XGBClassifier	0.87	0.55	0.55	0.84	
AdaBoostClassifier	0.88	0.54	0.54	0.84	
BaggingClassifier	0.87	0.54	0.54	0.84	
ExtraTreesClassifier	0.88	0.52	0.52	0.83	
LGBMClassifier	0.87	0.52	0.52	0.82	
KNeighborsClassifier	0.87	0.51	0.51	0.82	
LinearDiscriminantAnalysis	0.87	0.50	0.50	0.82	
RandomForestClassifier	0.88	0.50	0.50	0.82	
LogisticRegression	0.87	0.50	0.50	0.82	
CalibratedClassifierCV	0.87	0.50	0.50	0.82	
DummyClassifier	0.88	0.50	0.50	0.82	
RidgeClassifier	0.88	0.50	0.50	0.82	
RidgeClassifierCV	0.88	0.50	0.50	0.82	
SGDClassifier	0.88	0.50	0.50	0.82	
SVC	0.88	0.50	0.50	0.82	
LinearSVC	0.88	0.50	0.50	0.82	

Hình 15: Kết quả thời gian thực hiện các mô hình

Model	Time Taken
NearestCentroid	0.02
GaussianNB	0.02
QuadraticDiscriminantAnalysis	0.03
DecisionTreeClassifier	0.04
BernoulliNB	0.01
LabelSpreading	1.79
LabelPropagation	1.48
Perceptron	0.04
ExtraTreeClassifier	0.02
PassiveAggressiveClassifier	0.03
XGBClassifier	1.32
AdaBoostClassifier	0.31
BaggingClassifier	0.23
ExtraTreesClassifier	0.58
LGBMClassifier	0.11
KNeighborsClassifier	0.30
LinearDiscriminantAnalysis	0.02
RandomForestClassifier	0.70
LogisticRegression	0.05
CalibratedClassifierCV	0.08
DummyClassifier	0.01
RidgeClassifier	0.02
RidgeClassifierCV	0.02
SGDClassifier	0.05
SVC	0.75
LinearSVC	0.29

Hình 16: Kết quả thời gian thực hiện mô hình

1. RandomForestClassifier

```
# Huấn luyện và đánh giá mô hình
# Tạo đối tượng
rfc = RandomForestClassifier(random_state=0, max_features='sqrt',
n_estimators=200, class_weight='balanced')

# Huấn luyện
rfc.fit(X_train, y_train)

# Độ chính xác trên tập huấn luyện
rfc.score(X_train, y_train)

# Dự đoán kết quả trên tập kiểm tra
rfc_pred = rfc.predict(X_test)

print('Random Forest')
print('Accuracy Score: ', accuracy_score(y_test, rfc_pred))
print('Precision Score: ', precision_score(y_test, rfc_pred))
print('Recall Score: ', recall_score(y_test, rfc_pred))
print('F1 Score: ', f1_score(y_test, rfc_pred))
```

Kết quả:

```
Random Forest
Accuracy Score:  0.8777506112469438
Precision Score:  1.0
Recall Score:  0.006622516556291391
F1 Score:  0.013157894736842105
```

Hình 17: Kết quả dự đoán mô hình Random Forest

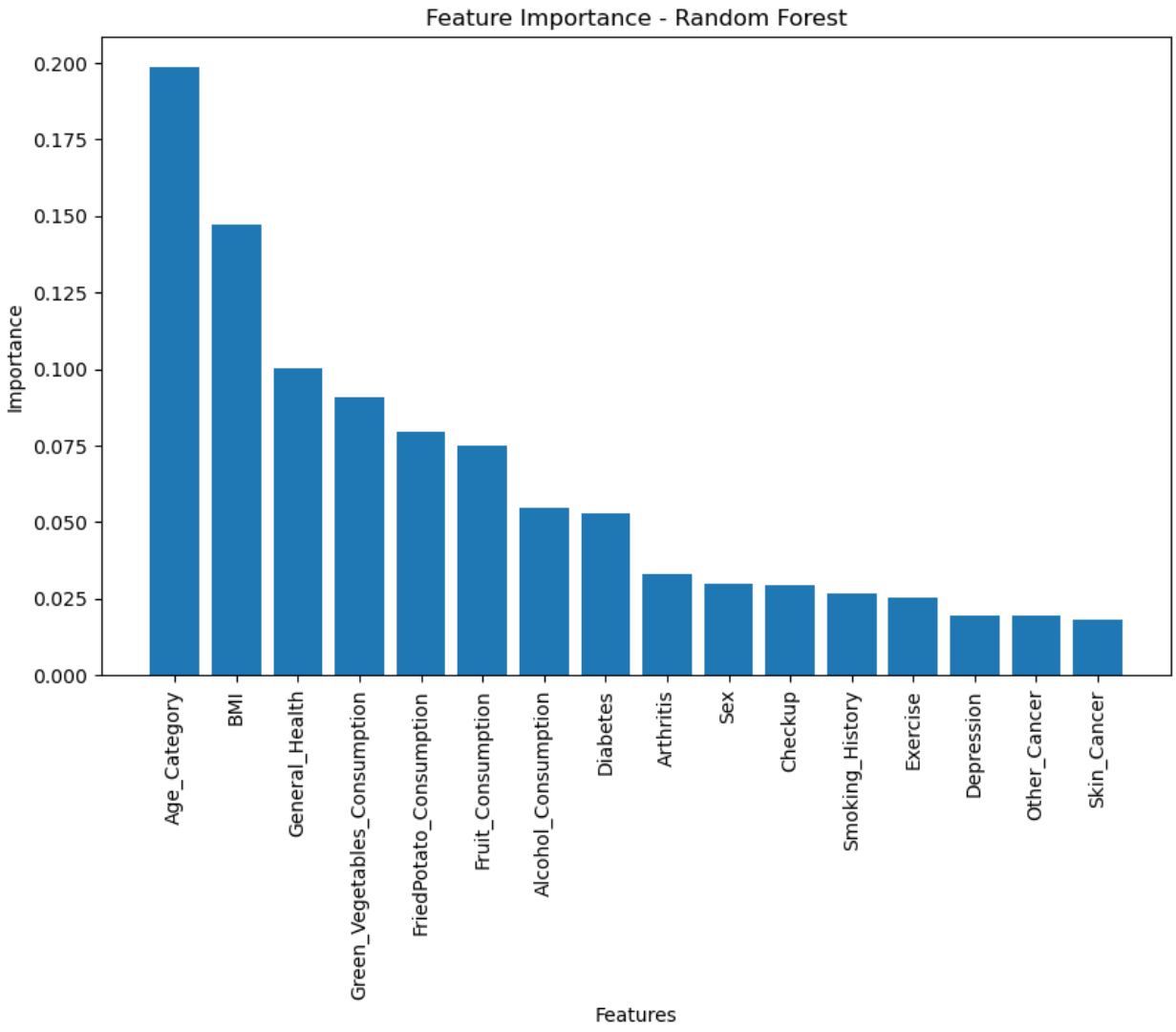
Độ chính xác của mô hình là 87,78%.

Trực quan hóa mô hình.

```
# Trích xuất độ quan trọng của các đặc trưng
importances = rfc.feature_importances_

# Sắp xếp độ quan trọng theo thứ tự giảm dần
indices = np.argsort(importances)[::-1]

# Vẽ biểu đồ cột của độ quan trọng
plt.figure(figsize=(10, 6))
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), X_train.columns[indices], rotation=90)
```

```
plt.title('Feature Importance - Random Forest')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```

Kết quả:



Hình 18: Trực quan mô hình Random Forest

Các đặc trưng được xếp theo thứ tự giảm dần theo mức độ quan trọng. Đặc trưng có mức độ quan trọng cao nhất là Age_Category, tiếp theo là BMI và General_Health. Điều này cho thấy rằng những đặc trưng này có tác động lớn nhất đến nguy cơ mắc bệnh tim.

2. LogisticRegression

```
# Khởi tạo và huấn luyện mô hình
lr = LogisticRegression()

# Huấn luyện mô hình trên tập huấn luyện
lr.fit(X_train, y_train)

# Độ chính xác trên tập huấn luyện
lr.score(X_train, y_train)

# Dự đoán kết quả trên tập kiểm tra
lr_pred = lr.predict(X_test)

print('Logistic Regression')
print('Accuracy Score: ', accuracy_score(y_test, lr_pred))
print('Precision Score: ', precision_score(y_test, lr_pred))
print('Recall Score: ', recall_score(y_test, lr_pred))
print('F1 Score: ', f1_score(y_test, lr_pred))
```

Kết quả:

```
Logistic Regression
Accuracy Score:  0.8736756316218419
Precision Score:  0.25
Recall Score:  0.013245033112582781
F1 Score:  0.025157232704402514
```

Hình 19: Kết quả dự đoán mô hình Logistic Regression

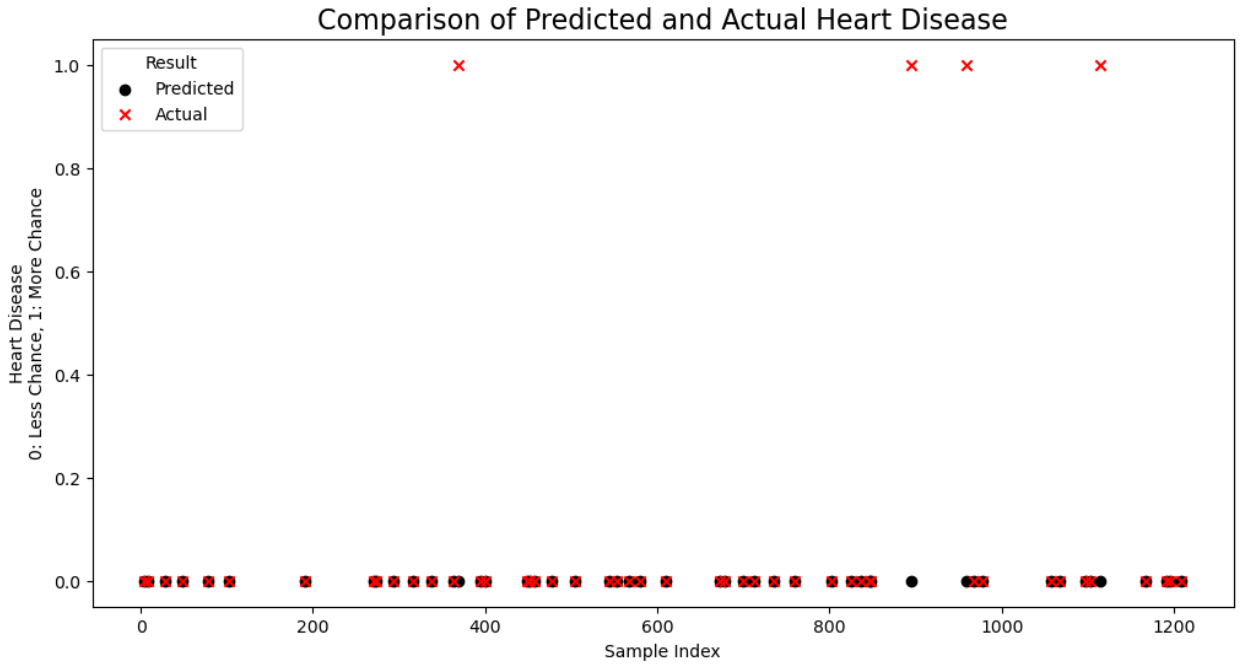
Độ chính xác của mô hình là 87,37%.

Trực quan kết quả.

```
# Số lượng điểm muốn hiển thị
num_points_to_display = 50

# Lựa chọn ngẫu nhiên một số lượng điểm từ dữ liệu
random_indices = np.random.choice(len(lr_pred), num_points_to_display,
replace=False)

# Tạo biểu đồ với các điểm được chọn ngẫu nhiên
plt.figure(figsize=(12, 6))
plt.scatter(random_indices, lr_pred[random_indices], color="black",
label="Predicted")
```

```
plt.scatter(random_indices, y_test.values[random_indices], color="red",
marker="x", label="Actual")
plt.title("Comparison of Predicted and Actual Heart Disease", fontsize=16)
plt.xlabel("Sample Index")
plt.ylabel("Heart Disease\n0: Less Chance, 1: More Chance")
plt.legend(title="Result")
plt.show()
```

Kết quả:



Hình 20: Trực quan mô hình Logistic Regression

3. LinearRegression

```
# Khởi tạo mô hình Linear Regression
linear_reg_model = LinearRegression()

# Huấn luyện mô hình trên tập huấn luyện
linear_reg_model.fit(X_train, y_train)

# Độ chính xác trên tập huấn luyện
linear_reg_model.score(X_train, y_train)

# Dự đoán kết quả trên tập kiểm tra
y_pred_linear_reg = linear_reg_model.predict(X_test)
```

```
# In các giá trị độ đo hiệu suất
print('Linear Regression Performance Metrics:')
print('Mean Squared Error (MSE): ', mean_squared_error(y_test,
y_pred_linear_reg))
print('Mean Absolute Error (MAE): ', mean_absolute_error(y_test,
y_pred_linear_reg))
print('R-squared (R2) Score: ', r2_score(y_test, y_pred_linear_reg))
```

Kết quả:

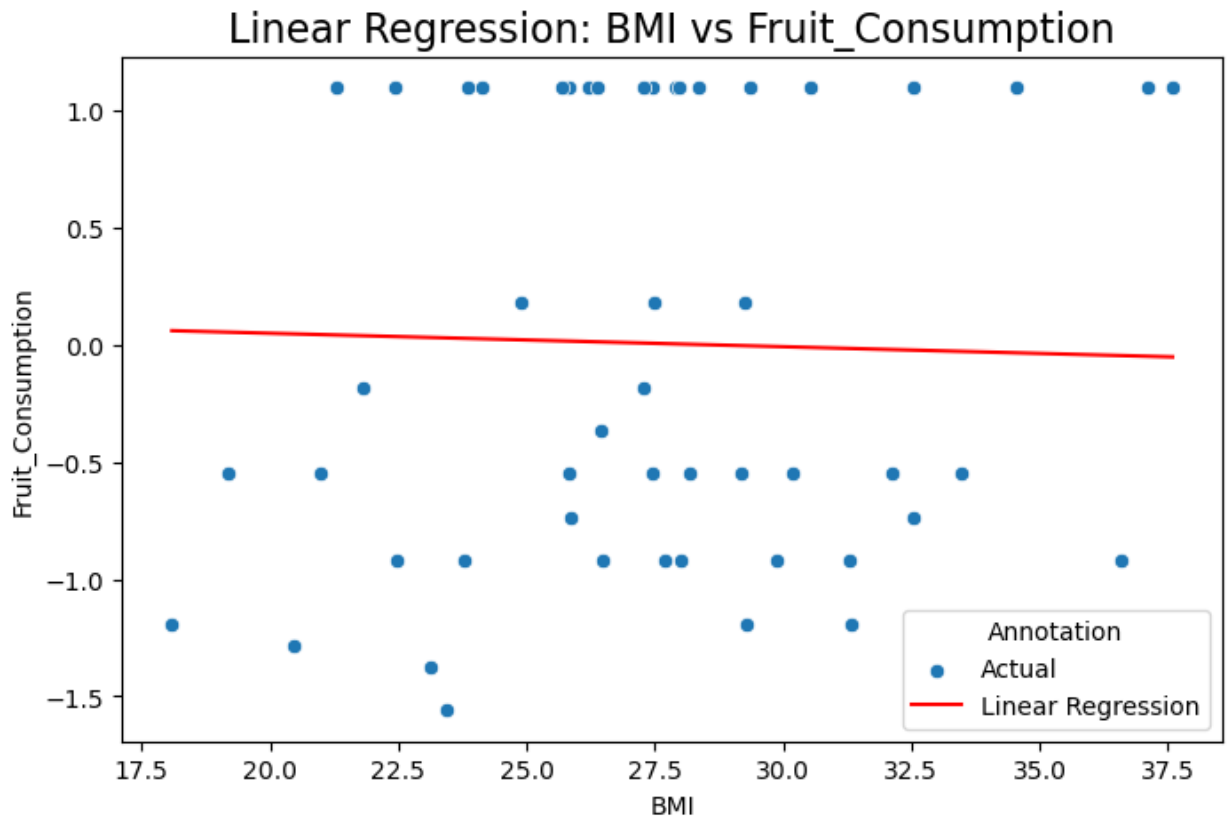
```
Linear Regression Performance Metrics:
Mean Squared Error (MSE):  0.09595550450541138
Mean Absolute Error (MAE):  0.18404446628939794
R-squared (R2) Score:  0.11086071332056757
```

Hình 21: Kết quả đánh giá hiệu suất hồi quy tuyến tính

Trực quan kết quả.

```
# Tính toán sự chênh lệch giữa giá trị thực tế và giá trị dự đoán
residuals = y_test - lr_predictions

# Vẽ histogram của sự chênh lệch
plt.figure(figsize=(10, 6))
sns.histplot(residuals, bins=30, kde=True)
plt.title('Residuals Distribution - Linear Regression')
plt.xlabel('Residuals (Actual - Predicted)')
plt.ylabel('Frequency')
plt.show()
```

Kết quả:



Hình 22: Trực quan mô hình Linear Regression

4. Support Vector Machine

```
# Khởi tạo mô hình SVM
svm_model = SVC(kernel='linear', C=1.0)

# Huấn luyện mô hình trên tập huấn luyện
svm_model.fit(X_train, y_train)

# Dự đoán lớp trên tập kiểm tra
y_pred_svm = svm_model.predict(X_test)

# Đánh giá hiệu suất của mô hình
accuracy_svm = accuracy_score(y_test, y_pred_svm)
report_svm = classification_report(y_test, y_pred_svm)

# In kết quả
print(f"Accuracy (SVM): {accuracy_svm:.2f}")
print("Classification Report (SVM):\n", report_svm)
```

Kết quả:

```
Accuracy (SVM): 0.88
Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      1076
           1       0.00      0.00      0.00       151

    accuracy                           0.88      1227
   macro avg       0.44      0.50      0.47      1227
weighted avg       0.77      0.88      0.82      1227
```

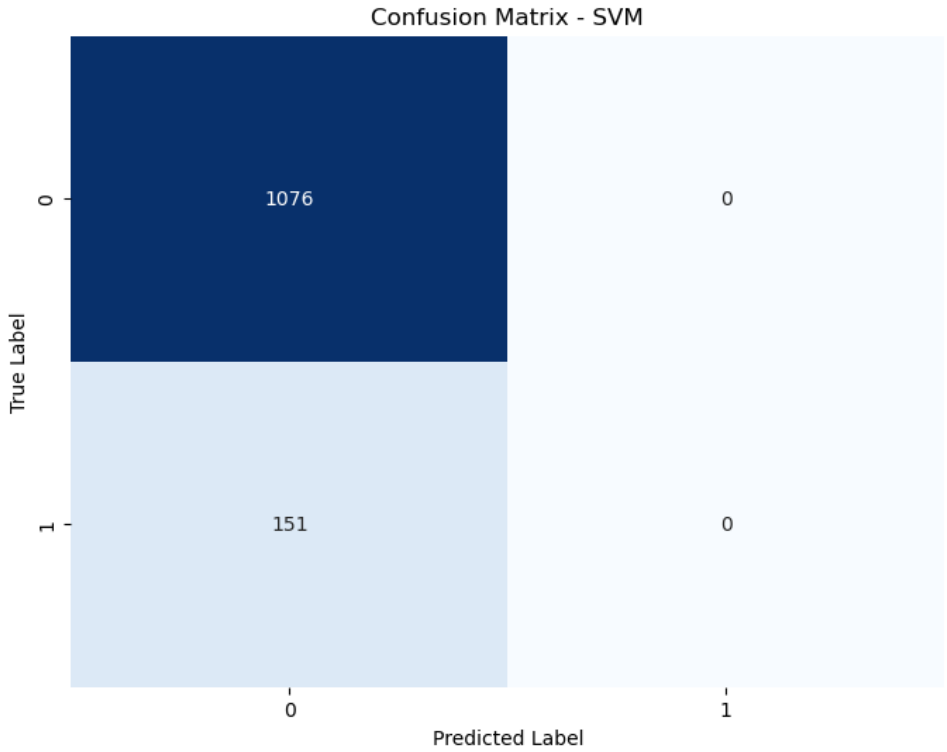
Hình 23: Kết quả dự đoán mô hình SVM

Độ chính xác của mô hình này là 88%.

Trực quan bằng heatmap.

```
# Ma trận nhầm lẫn
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)

# Vẽ heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - SVM')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Kết quả:



Hình 24: Trực quan SVM

5. K-Nearest Neighbor

```
# Chuyển đổi dữ liệu thành kiểu numpy array và kiểu số thực (float)
X_train = X_train.values.astype(float)
X_test = X_test.values.astype(float)

# Khởi tạo mô hình KNN
knn_model = KNeighborsClassifier()

# Huấn luyện mô hình trên tập huấn luyện
knn_model.fit(X_train, y_train)

# Dự đoán trên tập kiểm tra
y_pred = knn_model.predict(X_test)

# Đánh giá hiệu suất của mô hình
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# In kết quả
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:\n", report)
```

Kết quả:

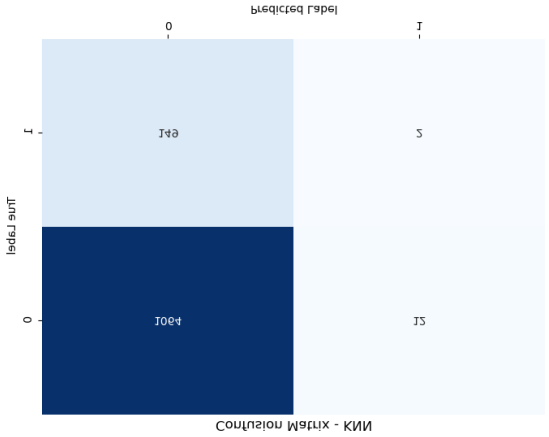
```
Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      1076
           1       0.14      0.01      0.02       151

    accuracy                           0.87      1227
   macro avg       0.51      0.50      0.48      1227
weighted avg       0.79      0.87      0.82      1227
```

Độ chính xác của mô hình này là 87%.

Trực quan bằng heatmap.

```
# Ma trận nhầm lẫn
conf_matrix_knn = confusion_matrix(y_test, y_pred)

# Vẽ heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - KNN')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Kết quả:



Hình 25: Trực quan KNN

CÂU 2: HỌC KHÔNG GIÁM SÁT

I. Giới thiệu dữ liệu

Thông tin về Tập dữ liệu:

Tập dữ liệu Bán lẻ trực tuyến II này chứa tất cả các giao dịch xảy ra đối với hoạt động bán lẻ trực tuyến có cửa hàng có trụ sở tại và đã đăng ký tại Vương quốc Anh trong khoảng thời gian từ ngày 12/01/2009 đến ngày 12/09/2011. Công ty chủ yếu bán quà tặng độc đáo cho mọi dịp. Nhiều khách hàng của công ty là người bán buôn.

Thông tin thuộc tính:

Invoice: Số hóa đơn. Trên danh nghĩa. Một số nguyên gồm 6 chữ số được gán duy nhất cho mỗi giao dịch. Nếu mã này bắt đầu bằng chữ cái 'c' thì nó biểu thị việc hủy.
StockCode: Mã sản phẩm (mặt hàng). Trên danh nghĩa. Một số nguyên gồm 5 chữ số được gán duy nhất cho mỗi sản phẩm riêng biệt.
Description: Tên sản phẩm (mặt hàng). Trên danh nghĩa.
Quantity: Số lượng của từng sản phẩm (mặt hàng) trên mỗi giao dịch. Số.
InvoiceDate: Ngày và giờ lập hóa đơn. Số. Ngày và giờ khi giao dịch được tạo.
Price: Đơn giá. Số. Giá sản phẩm trên mỗi đơn vị tính bằng đồng bảng Anh (Â £).
Customer ID: Mã số khách hàng. Trên danh nghĩa. Một số nguyên gồm 5 chữ số được gán duy nhất cho mỗi khách hàng.
Country: Tên quốc gia. Trên danh nghĩa. Tên quốc gia nơi khách hàng cư trú.

II. Đọc dữ liệu

1. Nhập thư viện

Việc import thư viện là một bước quan trọng trong lập trình, giúp chúng ta sử dụng các chức năng, lớp, hoặc phương thức đã được định nghĩa sẵn trong thư viện đó. Khi bắt đầu viết code, chúng ta thường cần sử dụng các tính năng mà ngôn ngữ lập trình đó không cung cấp sẵn, và đó là lúc thư viện đến đầu tiên.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
from itertools import product
import datetime as dt
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
import warnings
from pickle import dump
```

```
%matplotlib inline
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
```

2. Đọc dữ liệu

Khi làm việc với dữ liệu được lưu trữ trong các tệp tin Excel, việc đọc dữ liệu trở nên quan trọng để nắm bắt thông tin và thực hiện các phân tích hoặc xử lý tiếp theo.

```
retail=pd.read_excel("D:\code\Học máy 1\do_an\online_retail_II.xlsx")
retail.head()
```

Kết quả:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

Hình 26: Kết quả in ra DataFrame gốc

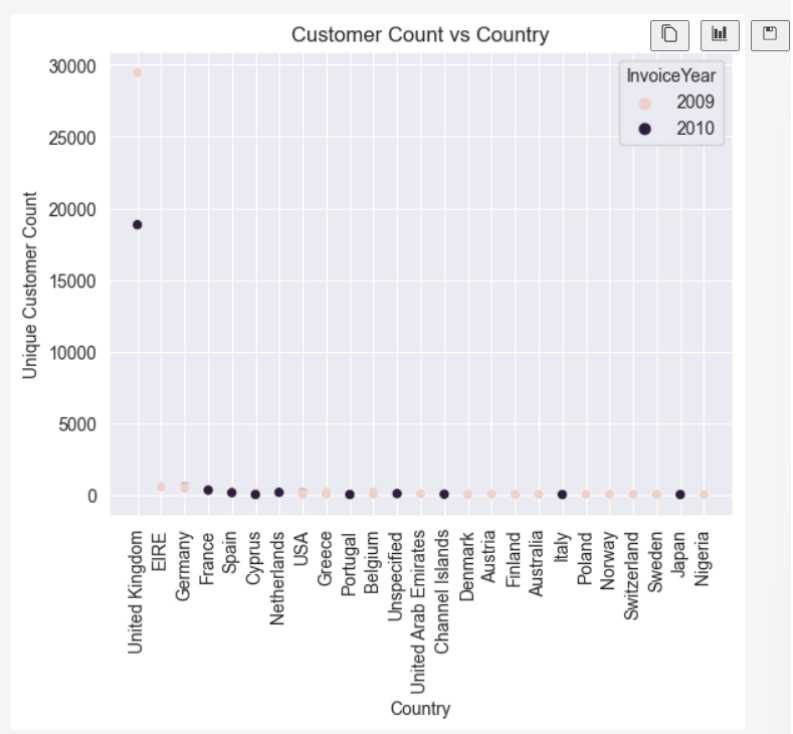
III. Trực quan một số thuộc tính trong DataFrame

Thực hiện vẽ trực quan hóa dữ liệu là một bước quan trọng trong quá trình phân tích dữ liệu, giúp chúng ta hiểu rõ hơn về xu hướng, mối quan hệ của dữ liệu. Trực quan hóa không chỉ làm cho thông tin trở nên dễ hiểu hơn, mà còn giúp tạo ra các hình ảnh, hỗ trợ trong việc giải thích và trình bày kết quả.

Số lượng khách hàng so với quốc gia

```
# Tách cột 'InvoiceDate' thành năm chỉ là 'InvoiceYear' và chỉ các tháng là
'InvoiceMonth'
retail['InvoiceYear'] = retail['InvoiceDate'].dt.year
retail['InvoiceMonth'] = retail['InvoiceDate'].dt.month
retail
# Phân bổ khách hàng theo quốc gia
customerid_vs_country = sns.scatterplot(x = 'Country', y = 'Customer ID', hue =
'InvoiceYear', data = customerid_country)
customerid_vs_country.set_xlabel('Country')
customerid_vs_country.set_ylabel('Unique Customer Count')
customerid_vs_country.set_title('Customer Count vs Country')
plt.xticks(rotation=90)
plt.plot()
```

Kết quả:



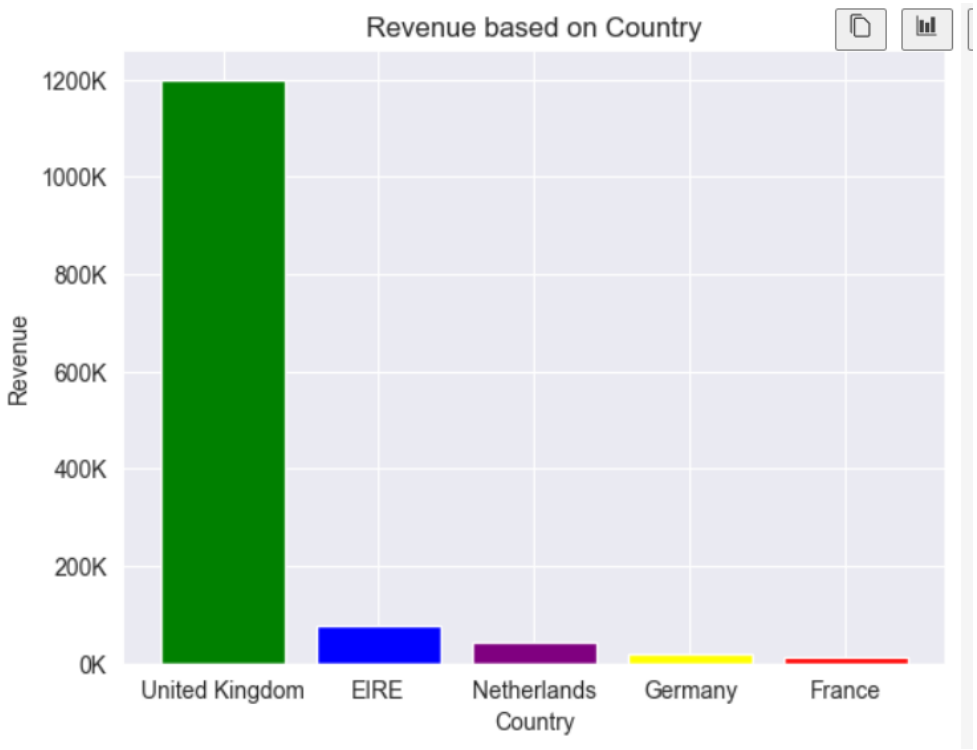
Hình 27:Trực quan số lượng khách hàng so với quốc gia

Biểu đồ thể hiện số lượng khách hàng ở các quốc gia khác nhau của năm2009 và năm 2010.Dựa vào biểu đồ ta có thể thấy số lượng khách united Kingdom có số lượng cao nhất và thấp nhất là nigeria.

Doanh thu dựa trên quốc gia

```
from matplotlib.ticker import FuncFormatter
# Hàm định dạng giá trị trục y
def format_thousands(x, pos):
    return f'{x/1000:.0f}K'
# Vẽ biểu đồ thanh
colors = ['Green', 'Blue', 'Purple', 'Yellow', 'Red']
plt.bar(country_revenue['Country'], country_revenue['Revenue'], color=colors)

# Định dạng trục y theo đơn vị hàng nghìn
plt.gca().yaxis.set_major_formatter(FuncFormatter(format_thousands))

plt.xlabel('Country')
plt.ylabel('Revenue')
plt.title('Revenue based on Country')
plt.show()
```

Kết quả:



Hình 28: Trực quan doanh thu dựa trên quốc gia

Biểu đồ thể hiện doanh thu của công ty ở các quốc gia khác nhau. Dựa vào biểu đồ ta có thể thấy là doanh thu thu được từ united kingdom là cao nhất và doanh thu của công ty ở các nước khác là tương đối thấp.

Số lượng so với ID khách hàng theo quốc gia

```
# Đếm số lượng và nhóm theo mã khách hàng và quốc gia
customerid_quantity = retail.groupby(['Customer ID',
'Country'])['Quantity'].sum().sort_values(ascending=False).reset_index().head(100
)
customerid_quantity
# Tạo biểu đồ phân tán
customerid_vs_quantity = sns.scatterplot(x = 'Customer ID', y = 'Quantity', hue =
'Country', data = customerid_quantity)
customerid_vs_quantity.set_xlabel('Customer ID')
customerid_vs_quantity.set_ylabel('Quantity')
customerid_vs_quantity.set_title('Quantity vs Customer ID according to the
Country')
plt.plot()
```

Kết quả:

Hình 29: Trực quan số lượng so với ID khách hàng theo quốc gia

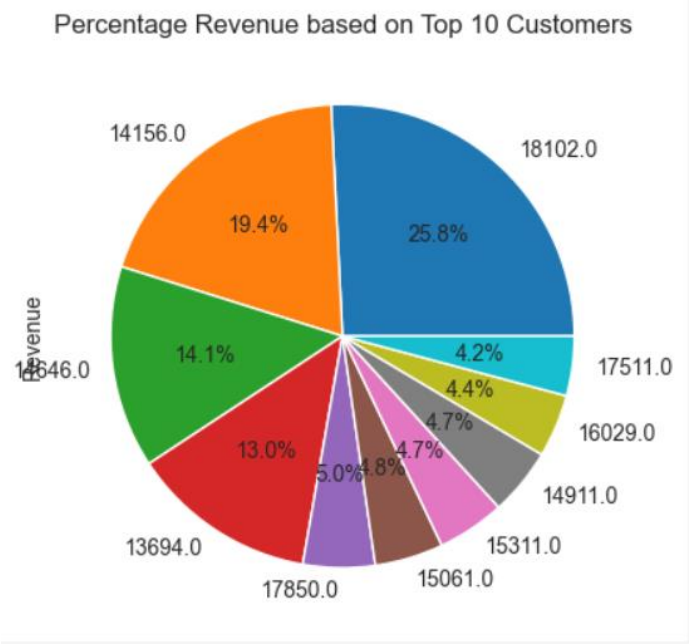
Biểu đồ thể hiện mối quan hệ giữa số lượng hàng hóa và số lượng mã khách hàng của mỗi quốc gia.Từ đó, chúng ta có thể biết được đâu là thi trường tiềm năng.

- Vương quốc Anh là thị trường lớn nhất, với số lượng hàng hóa và số lượng mã khách hàng cao nhất.
- Các quốc gia có số lượng hàng hóa nhỏ hơn có thể là những thị trường có thể khai thác thêm.

Phần trăm doanh thu dựa trên 10 khách hàng đầu tiên

```
customerid_revenue.plot(kind = 'pie', autopct='%1.1f%%')
plt.title('Percentage Revenue based on Top 10 Customers')
plt.show()
```

Kết quả:



Hình 30: Phần trăm doanh thu dựa trên 10 khách hàng đầu tiên

Biểu đồ thể hiện tỷ lệ doanh thu của một công ty theo 10 khách hàng đầu tiên.

- Khách hàng ‘18102’, chiếm hơn 25.8% tổng doanh thu.
- Các khách hàng tiếp theo là "14156", "14646" và "13694", chiếm lần lượt là 19.4%, 14.1% , 13% tổng doanh thu mỗi khách hàng.
- Các khách hàng còn lại chiếm dưới 10% tổng doanh thu.

IV. Xử lý dữ liệu

Kiểm tra thông tin của DataFrame

```
retail.info()
```

Kết quả:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74938 entries, 0 to 74937
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Invoice       74938 non-null  object
 1   StockCode     74938 non-null  object
 2   Description   74495 non-null  object
 3   Quantity      74938 non-null  int64
 4   InvoiceDate   74938 non-null  datetime64[ns]
 5   Price         74938 non-null  float64
 6   Customer ID   52752 non-null  float64
 7   Country       74938 non-null  object
 8   InvoiceYear   74938 non-null  int64
 9   InvoiceMonth  74938 non-null  int64
 10  Revenue       74938 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(3), object(4)
memory usage: 6.3+ MB
```

Hình 31: Kết quả thông tin DataFrame

Kiểm tra giá trị thiếu (missing values)

```
retail.isnull().sum()
```

Kết quả:

```
Invoice            0
StockCode          0
Description      443
Quantity           0
InvoiceDate        0
Price              0
Customer ID    22186
Country            0
InvoiceYear        0
InvoiceMonth       0
Revenue            0
dtype: int64
```

Hình 32: Kết quả các giá trị thiếu

Xử lý giá trị thiếu (missing values)

```
retail = retail[retail['Customer ID'].notnull()]
```

Kết quả:

```
Invoice         0
StockCode       0
Description     0
Quantity        0
InvoiceDate     0
Price           0
Customer ID     0
Country         0
InvoiceYear     0
InvoiceMonth    0
Revenue         0
dtype: int64
```

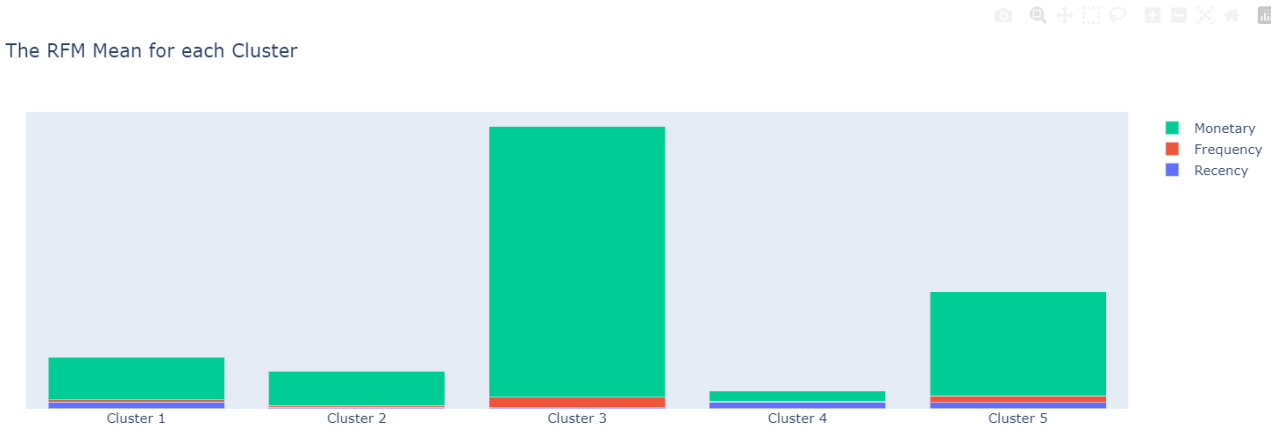
Hình 33: Kết quả các giá trị thiếu sau khi xử lý

V. Mô hình

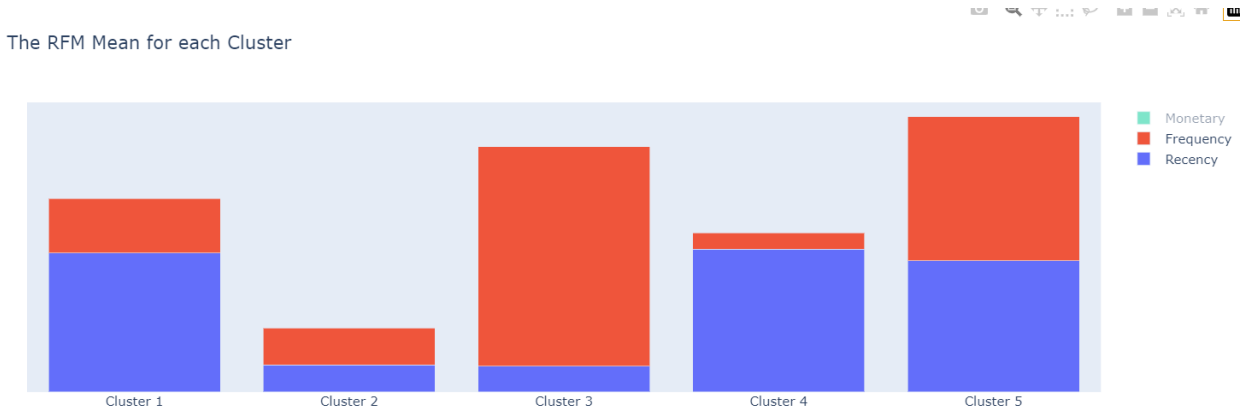
1. K-means

```
rfm_mean_df = rfm_details.rfm_mean_df
X = rfm_df[['R', 'F', 'M']]
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300)
kmeans.fit(X)
rfm_df['KMeans y_pred'] = kmeans.labels_
np.unique(rfm_df['KMeans y_pred'], return_counts=True)
def stacked_bar_mean(model_y_pred):
    cluster_names = [item[0] for item in rfm_mean_df.index.tolist()]
    r_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Recency'],
        name = 'Recency'
        )
    f_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Frequency'],
        name = 'Frequency'
        )
    m_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Monetary'],
        name = 'Monetary'
        )
    layout = go.Layout(title='The RFM Mean for each Cluster',
                       barmode='stack')

    fig = go.Figure(data=[r_trace, f_trace, m_trace], layout=layout)
    if model_y_pred == 'DBSCAN y_pred':
        fig.update_layout(yaxis_visible=False, xaxis_tickangle=-90)
        fig.show()
    else:
        fig.update_layout(yaxis_visible=False)
        fig.show()
stacked_bar_mean('KMeans y_pred')
```

Kết quả:



Hình 34: Số lượng các điểm trong 5 cụm của 3 cột

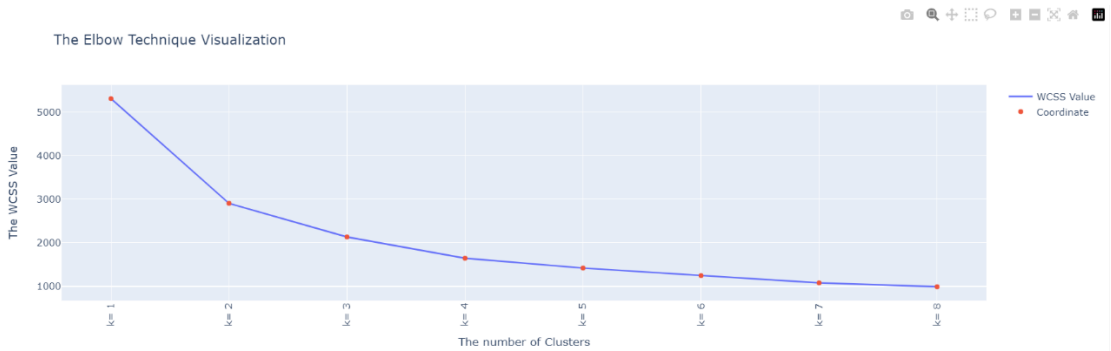


Hình 35: Số lượng các điểm trong 5 cụm của 2 cột

Phương pháp khuỷu tay

```
elbow = {}
for k in range(1, 9):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X)
    elbow[k] = kmeans.inertia_
point_plot_trace = go.Scatter(
    x = [f'k= {num}' for num in sorted(list(elbow.keys()))],
    y = list(elbow.values()),
    mode = 'markers',
    name = 'Coordinate'
)

line_trace = go.Scatter(
    x = [f'k= {num}' for num in sorted(list(elbow.keys()))],
    y = list(elbow.values()),
    mode = 'lines',
    name = 'WCSS Value'
)

layout = go.Layout(
    title = 'The Elbow Technique Visualization',
    xaxis = dict(title='The number of Clusters'),
    yaxis = dict(title='The WCSS Value')
)

point_plot = go.Figure(data=[line_trace, point_plot_trace], layout=layout)
point_plot.update_layout(xaxis_tickangle=-90)
point_plot.show()
```

Kết quả:



Hình 36: K-elbow

Đáng giá chất lượng phân cụm

```
rfm_df['KMeans y_pred'] = kmeans.labels_

print(silhouette_score(X=X, labels=rfm_df['KMeans y_pred']))
```

Kết quả:

```
0.9390292583386687
```

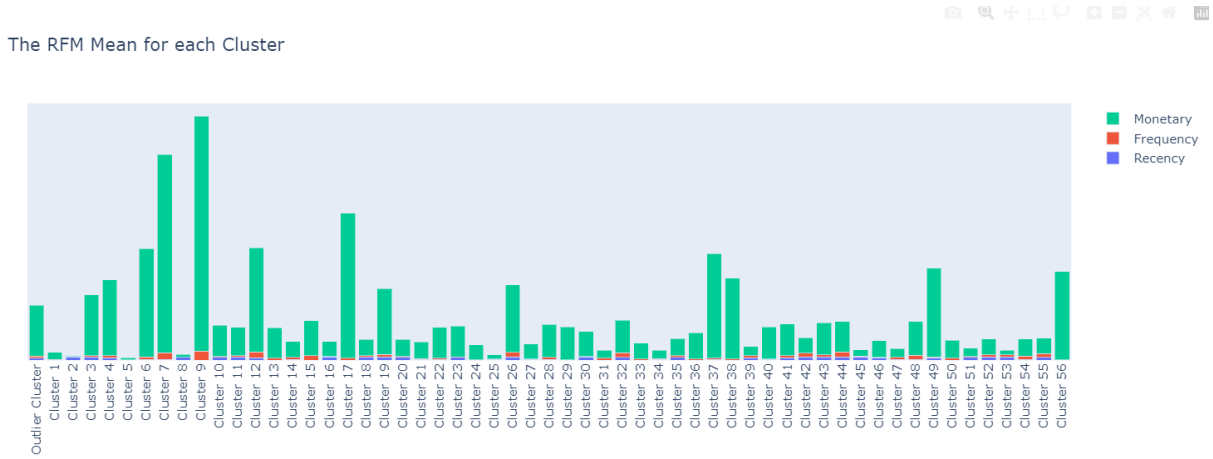
Hình 37: Kết quả đánh giá chất lượng phân cụm

2. DBScan

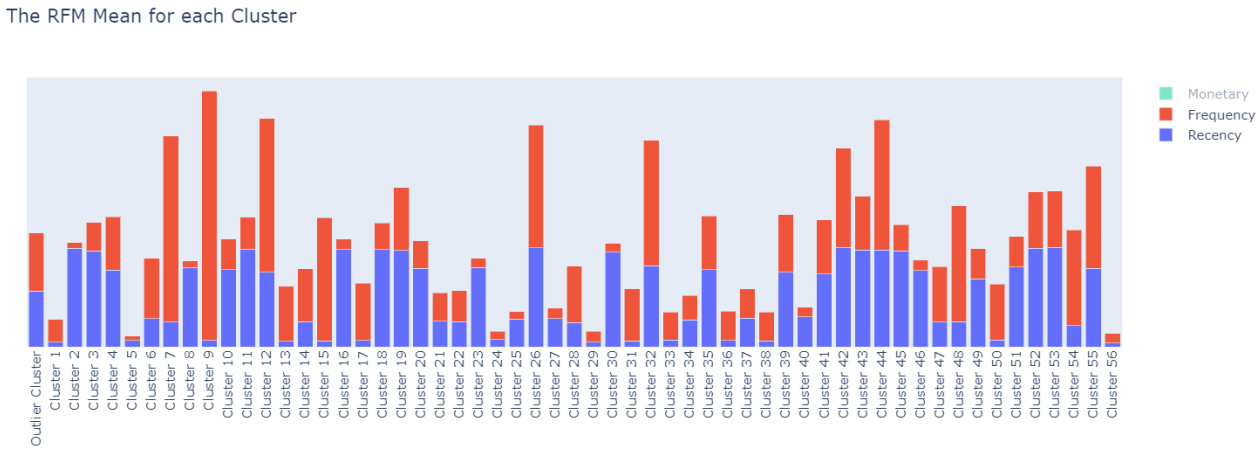
```
def stacked_bar_mean(model_y_pred):
    cluster_names = [item[0] for item in rfm_mean_df.index.tolist()]
    r_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Recency'],
        name = 'Recency'
        )

    f_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Frequency'],
        name = 'Frequency'
        )

    m_trace = go.Bar(
        x = cluster_names,
        y = rfm_mean_df['Monetary'],
        name = 'Monetary'
        )

    layout = go.Layout(title='The RFM Mean for each Cluster',
                       barmode='stack')

    fig = go.Figure(data=[r_trace, f_trace, m_trace], layout=layout)
    if model_y_pred == 'DBSCAN y_pred':
        fig.update_layout(yaxis_visible=False, xaxis_tickangle=-90)
        fig.show()
    else:
        fig.update_layout(yaxis_visible=False)
```

```
        fig.show()
dbscan = DBSCAN(eps=.5, min_samples=5)
dbscan.fit(X)
rfm_df['DBSCAN y_pred'] =dbscan.labels_
np.unique(rfm_df['DBSCAN y_pred'], return_counts=True)
rfm_details = RFMDetails('DBSCAN y_pred', with_average=False, start=-1)
rfm_mean_df = rfm_details.rfm_mean_df
rfm_df[rfm_df["DBSCAN y_pred"] == -1][['Recency', 'Frequency',
'Monetary']].mean()
stacked_bar_mean('DBSCAN y_pred')
```

Kết quả:



Hình 38: Kết quả trực quan 1



Hình 39: Kết quả trực quan 2

Đáng giá chất lượng phân cụm

```
print(silhouette_score(X=X, labels=rfm_df['DBSCAN y_pred']))
```

Kết quả

```
0.9719222955802724
```

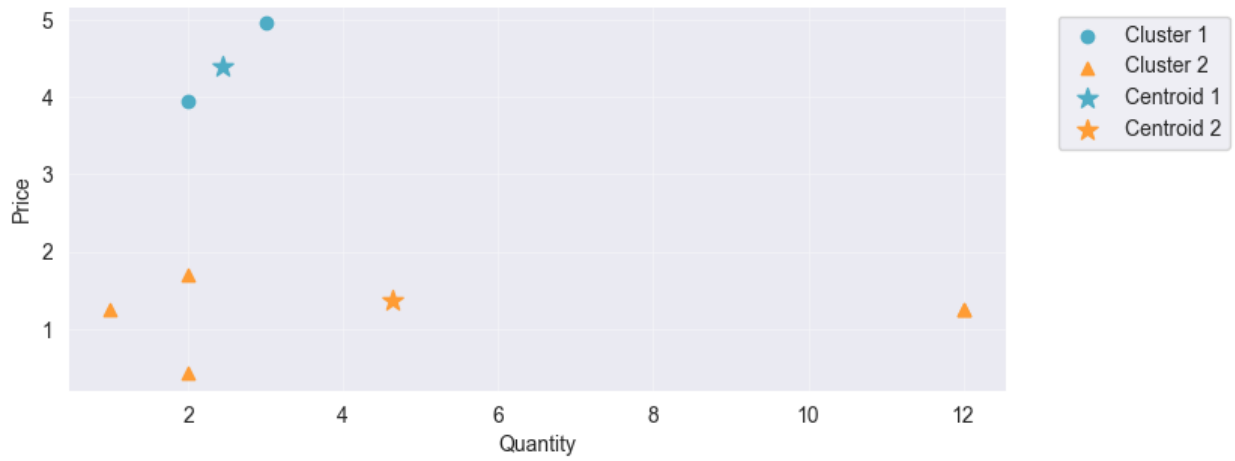
Hình 40: Kết quả đánh giá chất lượng phân cụm 2

3. MiniBatchKMeans

```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
from sklearn.impute import SimpleImputer
from sklearn.metrics.pairwise import pairwise_distances_argmin

# Giả sử 'df' là DataFrame chứa dữ liệu được cung cấp
# Chọn ngẫu nhiên các điểm từ dữ liệu
X_sample = df.sample(n=10, random_state=10)[['Quantity', 'Price', 'Customer ID']]

# Xử lý giá trị thiếu bằng cách điền NaN bằng giá trị trung bình của mỗi cột
imputer = SimpleImputer(strategy='mean')
X_cleaned = imputer.fit_transform(X_sample)

# MiniBatchKMeans
mbk = MiniBatchKMeans(
    init="k-means++",
    n_clusters=3,
    batch_size=4,
    n_init=5,
    max_no_improvement=5,
    verbose=0,
)
t0 = time.time()
mbk.fit(X_cleaned)
t_mini_batch = time.time() - t0

# Sắp xếp lại các tâm cụm của MiniBatchKMeans theo thứ tự của tâm cụm KMeans
order = pairwise_distances_argmin(mbk.cluster_centers_, mbk.cluster_centers_)
mbk_ordered_cluster_centers = mbk.cluster_centers_[order]

# Gán nhãn cho các điểm dữ liệu sử dụng các tâm cụm đã được sắp xếp
mbk_labels = pairwise_distances_argmin(X_cleaned, mbk_ordered_cluster_centers)
```

```
# Vẽ đồ thị
fig = plt.figure(figsize=(8, 3))
fig.subplots_adjust(left=0.2, right=0.98, bottom=0.05, top=0.9)

# Thêm lưới kẻ
plt.grid(which="both", linewidth=0.5, alpha=0.5)

# Đặt nhãn cho trục x và trục y
plt.xlabel("Quantity")
plt.ylabel("Price")

# Chọn marker và màu sắc cho các cụm
markers = ["o", "^", "x"]
colors = ["#4EACC5", "#FF9C34", "#4E9A06"]

# Vẽ các điểm dữ liệu
for k, col, marker in zip(range(2), colors, markers):
    my_members = mbk_labels == k
    cluster_center = mbk_ordered_cluster_centers[k]
    plt.scatter(X_cleaned[my_members, 0], X_cleaned[my_members, 1], c=col,
marker=marker, label=f'Cluster {k + 1}')

# Vẽ các tâm cụm
for k, col in zip(range(2), colors):
    plt.scatter(mbk_ordered_cluster_centers[k, 0], mbk_ordered_cluster_centers[k,
1], c=col, marker="*", s=100, label=f'Centroid {k + 1}')

# Hiển thị chú thích của cụm và tâm cụm
plt.legend(loc='upper right', bbox_to_anchor=(1.25, 1.0))

plt.show()
```

Kết quả:



Hình 41: Biểu đồ MinnibatchKMeans

Đánh giá chất lượng phân cụm

```
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Đánh giá silhouette score
silhouette_avg = silhouette_score(X_cleaned, mbk_labels)
print(f"Silhouette Score: {silhouette_avg}")
```

Kết quả:

```
0.7519034188597568
```

Hình 42: Kết quả đánh giá chất lượng phân cụm

CÂU 3: ĐÁNH GIÁ VÀ ĐƯA RA KẾT LUẬN VỀ KẾT QUẢ

I. Đánh giá phần 1

	Accuracy score	Precision score	Recall score	F1 score
Random Forest	0.8777	1.0	0.0066	0.0131
Logistic Regression	0.8736	0.25	0.0132	0.0251
Support Vector Machine	0.88	0.88	1.00	0.93
K-Nearest Neighbors	0.87	0.88	0.99	0.93

Linear Regression Performance Metrics:

Mean Squared Error (MSE): 0.09595550450541138

Mean Absolute Error (MAE): 0.18404446628939794

R-squared (R2) Score: 0.11086071332056757

Kết luận:

Về Accuracy Score:

Random Forest và Support Vector Machine (SVM) có độ chính xác cao nhất, đạt khoảng 0.88.

Logistic Regression và K-Nearest Neighbors (KNN) cũng có độ chính xác tốt, khoảng 0.87.

Về Precision Score:

Random Forest có precision score cao nhất, đạt 1.0, cho thấy mô hình này đưa ra ít dự đoán dương tính giả mạo hơn so với các mô hình khác.

Logistic Regression có precision score thấp nhất, chỉ 0.25, cho thấy mô hình này có xu hướng dự đoán dương tính ít chính xác hơn.

Về Recall Score:

SVM có recall score là 1.0, chỉ ra rằng mô hình này có khả năng tìm ra tất cả các trường hợp dương tính thực sự.

KNN cũng có recall score cao, 0.99, đồng nghĩa với việc mô hình này cũng có khả năng tìm ra hầu hết các trường hợp thực sự dương tính.

Random Forest và Logistic Regression có recall score thấp, chỉ 0.0066 và 0.0132, cho thấy khả năng bỏ sót các trường hợp dương tính là cao.

Về F1 Score:

SVM và KNN có F1 score cao nhất, 0.93.

Random Forest và Logistic Regression có F1 score thấp, dưới 0.03, cho thấy sự kết hợp kém giữa precision và recall.

Linear Regression:

Mean Squared Error (MSE) là 0.096, thể hiện sự chênh lệch lớn giữa dự đoán và giá trị thực tế.

Mean Absolute Error (MAE) là 0.184, là mức độ trung bình của sự chênh lệch tuyệt đối giữa dự đoán và giá trị thực tế.

R-squared (R2) Score là 0.111, chỉ ra mô hình Linear Regression không giải thích được một phần lớn sự biến động của dữ liệu.

Kết luận: SVM và KNN có vẻ là những mô hình tốt nhất dựa trên các điểm đánh giá trên.

II. Đánh giá phần 2

Ở phần 2 để so sánh hiệu quả của các mô hình phân cụm, chúng ta cũng có thể xem xét các chỉ số khác như silhouette score. Các chỉ số này đo lường mức độ tương đồng của các điểm dữ liệu trong cùng một cụm và sự khác biệt của các cụm với nhau. Theo kết quả thống kê, mô hình DBScan có silhouette score cao nhất (0.97), cho thấy mô hình này có khả năng phân cụm tốt nhất trên tập dữ liệu này. Mô hình K-means có kết quả tương đối kém hơn DBScan một chút với silhouette score là (0.93), trong khi mô hình MiniBatchKMeans là mô hình có kết quả kém nhất với silhouette score là (0.75) do sử dụng thuật toán tối ưu hóa xấp xỉ. Từ các so sánh trên ta có thể thấy mô hình DBScan và mô hình K-means là hai thuật toán phù hợp với tập dữ liệu này.

TÀI LIỆU THAM KHẢO

KẾT QUẢ KIỂM TRA ĐẠO VĂN