

1. Analyze and Preprocess data - Check if the dataset has missing values or has any other problem.
2. Feature Engineering
3. Divide the dataset into 2 training and test sets
4. Use Pipeline
 4. a. Use scaler and dimensional reduction (if it is necessary).
 5. b. Use suitable Naive Bayes for this problem.
5. Perform model on training set and test set using gridsearch CV
6. Measure performance of the model.
7. Which metric is your main metric for this problem and why? What are your scaler, dimensional reduction and naive bayes model as well as their params? (<= 200 words, also input your opinion or conclusion here)

How can I measure your point:

1. Your function is callable and runs correctly
2. The performance of your model (in full pipeline) is acceptable. The final error based on my train and test set is low enough.
3. The data preprocessing is correct or make sense
4. The Feature engineering is correct or make sense
5. Any other additional process will be considered a small plus point.

A Travel Company Is Offering Travel Insurance Packages To Their Customers. The new insurance plan also covers Covid-19. The Company Claims To Know Which Customers Are Interested To Buy It Based On The Company's Database History.

- Age - Age Of The Customer
- Employment Type - The Sector In Which Customer Is Employed GraduateOrNot - Whether The Customer Is College Graduate Or Not
- AnnualIncome - The Yearly Income Of The Customer In Indian Rupees
- FamilyMembers - Number Of Members In Customer's Family
- ChronicDisease - Whether The Customer Suffers From Any Major Disease Or Conditions Like Diabetes/High BP or Asthama,etc.
- FrequentFlyer - Derived Data Based On Customer's History Of Booking Air Tickets On Atleast 4 Different Instances In The Last 2 Years (2017-2019).
- EverTravelledAbroad - Has The Customer Ever Travelled To A Foreign Country
- TravelInsurance - Did The Customer Buy Travel Insurance Package During Introductory Offering Held In The Year 2019.

Submit Link: <https://forms.gle/CwmpBrfa2SYQic7G6>

✓ Load Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
```

```
drive.mount("/content/drive")
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
import pandas as pd
PATH = " " # Path to your file
```

```
df = pd.read_csv("/content/drive/MyDrive/Dataset/TravelInsurancePrediction.csv")
df.head()
#ToDo: Show histogram of dataframe
```

↗

	Unnamed: 0	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	0	31	Government Sector	Yes	400000	6	1	No	No	No
1	1	31	Private Sector/Self Employed	Yes	1250000	7	0	No	No	No

Private

```
df.info()
```

↗

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1987 entries, 0 to 1986
Data columns (total 10 columns):
```

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	1987 non-null	int64
1	Age	1987 non-null	int64
2	Employment Type	1987 non-null	object
3	GraduateOrNot	1987 non-null	object
4	AnnualIncome	1987 non-null	int64
5	FamilyMembers	1987 non-null	int64
6	ChronicDiseases	1987 non-null	int64
7	FrequentFlyer	1987 non-null	object
8	EverTravelledAbroad	1987 non-null	object
9	TravelInsurance	1987 non-null	int64

dtypes: int64(6), object(4)
memory usage: 155.4+ KB

```
from google.colab import drive
drive.mount('/content/drive')
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

▾ Data Analysis

```
# Get categorical columns and numerical columns
categorical_cols = [feature for feature in df.columns if df[feature].dtype == "O"]
numerical_cols = [feature for feature in df.columns if df[feature].dtype != "O"]
```

```
# Check Missing values
df[categorical_cols].isnull().sum()
```



```
# Check Missing columns
df[numerical_cols].isnull().sum()
```

```
fig, axes=plt.subplots(len(numerical_cols), 1, figsize=(14, 2*len(numerical_cols)),sharex=False,sharey=False)

for i, column in enumerate(numerical_cols):
    sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
plt.tight_layout(pad=2.0)
```



<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```

<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```

<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```

<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```

<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

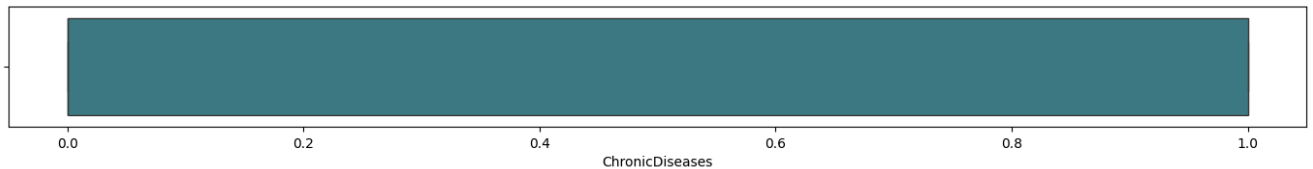
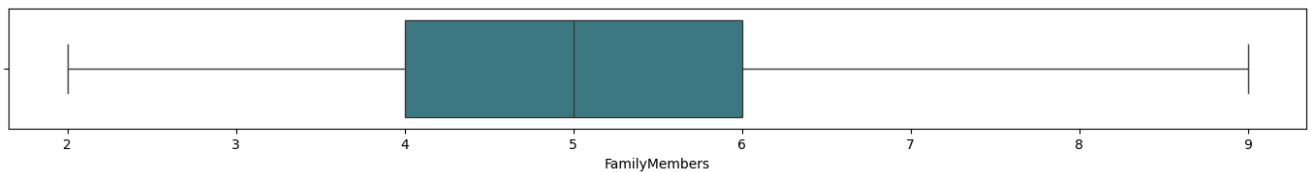
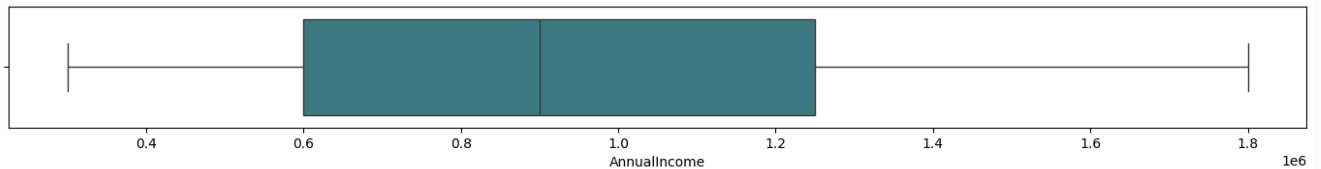
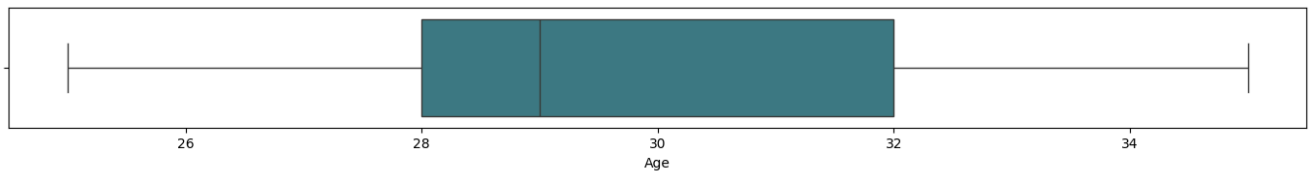
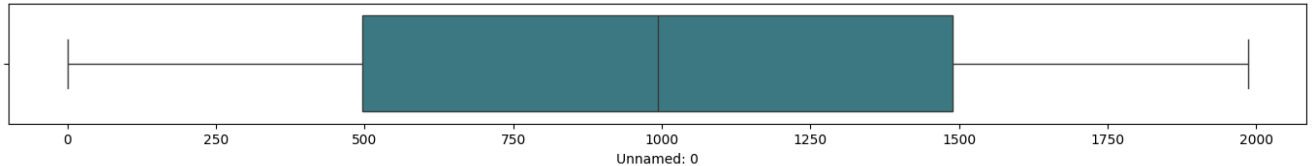
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```

<ipython-input-7-04b6fc8ceda2>:4: FutureWarning:

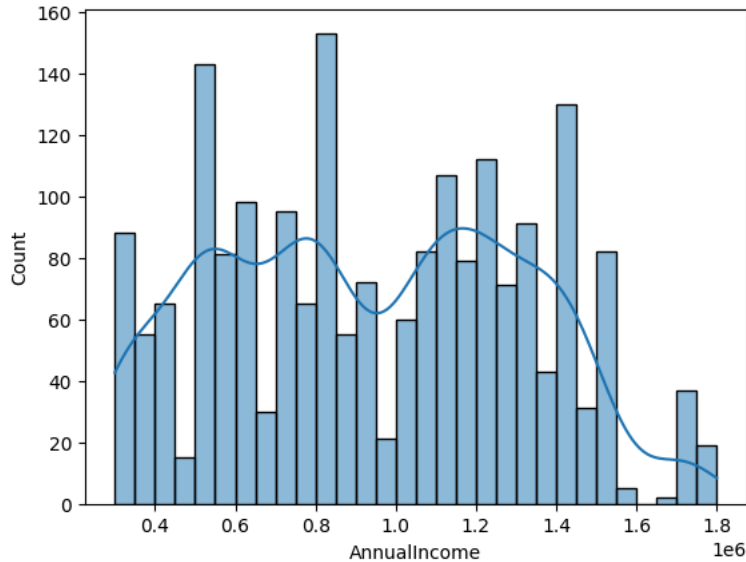
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

```
sns.boxplot(x=column,data=df,palette='crest',ax=axes[i])
```



```
sns.histplot(x="AnnualIncome", data=df, kde=True, bins=30)
```

```
<Axes: xlabel='AnnualIncome', ylabel='Count'>
```



```
# Trực quan hóa phân phối cho các cột số khác:
for col in numerical_cols:
    if col != 'AnnualIncome': # Avoid plotting AnnualIncome again
        plt.figure() # Create a new figure for each plot
        sns.histplot(x=col, data=df, kde=True, bins=30)
        plt.title(f'Distribution of {col}')
        plt.show()
```

✓ Preprocessing

```
def preprocessing_data(df):
    """
    Preprocess your data (eg. Drop null datapoints or fill missing data)
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    if 'Unnamed' in df.columns:
        df.drop("Unnamed", axis=1, inplace=True)
    return df
```

Xử lý ngoại lệ (Outliers)

```
import numpy as np
```

```
def handle_outliers(df, column):
    """
    Xử lý ngoại lệ bằng cách thay thế bằng giá trị tại phân vị 95.
    """
    Q95 = df[column].quantile(0.95)
    df.loc[df[column] > Q95, column] = Q95
    return df
```

```
# Áp dụng cho cột 'AnnualIncome'
df = handle_outliers(df, 'AnnualIncome')
```

Chuyển đổi dữ liệu (Data Transformation)

```
import numpy as np
```

```
# Log transformation cho cột 'AnnualIncome'
df['AnnualIncome_log'] = np.log(df['AnnualIncome'] + 1) # +1 để tránh log(0)


# StandardScaler cho cột 'Age'
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Age_scaled'] = scaler.fit_transform(df[['Age']])

df = preprocessing_data(df)
df = handle_outliers(df, 'AnnualIncome')

# Biến đổi log
df['AnnualIncome_log'] = np.log(df['AnnualIncome'] + 1)

# Scale tuổi
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Age_scaled'] = scaler.fit_transform(df[['Age']])

# In kết quả
df[['AnnualIncome', 'AnnualIncome_log', 'Age', 'Age_scaled']].head()
```



	AnnualIncome	AnnualIncome_log	Age	Age_scaled
0	400000	12.899222	31	0.463430
1	1250000	14.038655	31	0.463430
2	500000	13.122365	34	1.493446
3	700000	13.458837	28	-0.566587
4	700000	13.458837	28	-0.566587

```
df = preprocessing_data(df.copy())
```

✓ Feature Engineering

Kết hợp các cột

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

def apply_feature_engineering(df):
    """
    Apply all feature engineering to transform your data into number
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Chuyển các cột kiểu object (chuỗi) thành số
    for col in df.select_dtypes(include='object').columns:
        df[col] = df[col].astype('category').cat.codes

    # Chuẩn hóa dữ liệu về khoảng [0,1]
    scaler = MinMaxScaler()
    df[df.columns] = scaler.fit_transform(df[df.columns])

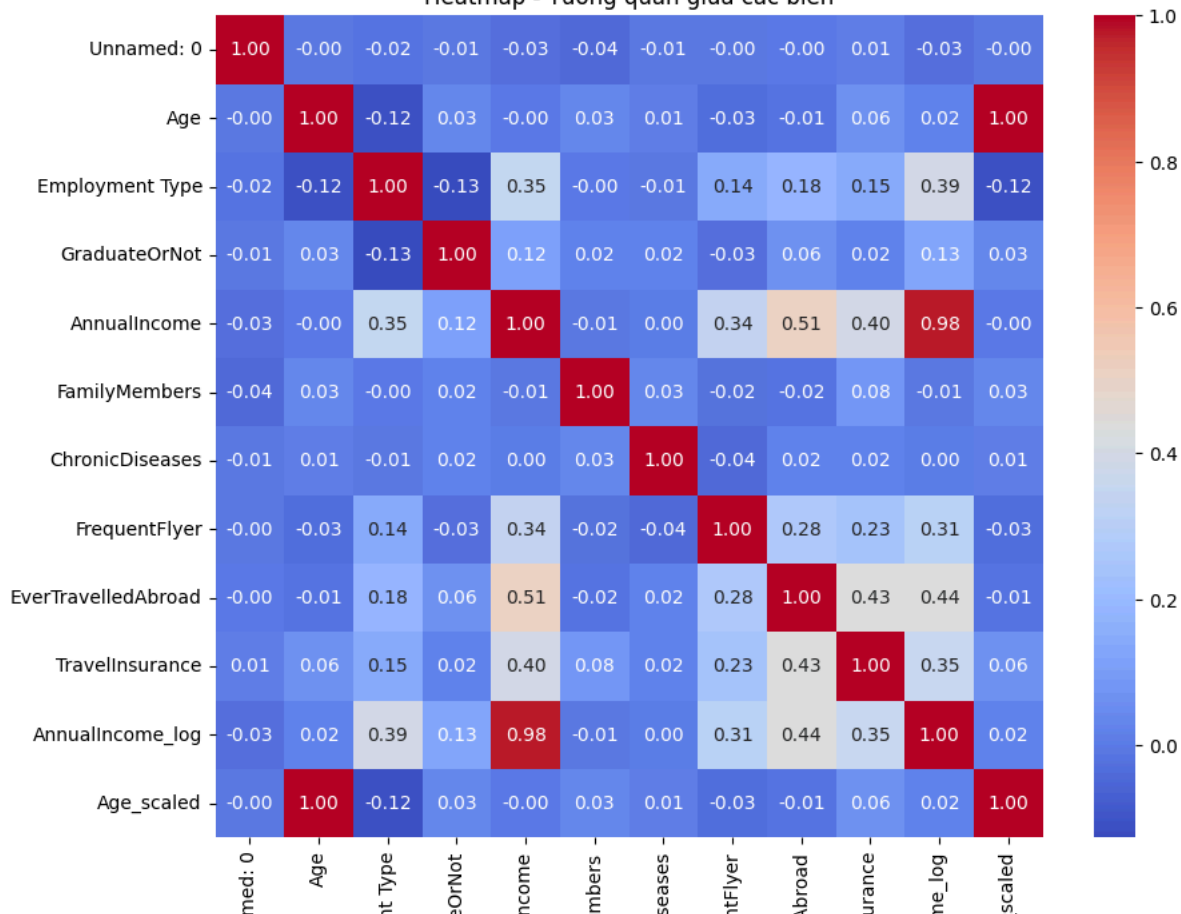
    return df

# Áp dụng lên dataframe của bạn (df cần được định nghĩa trước)
df = apply_feature_engineering(df)

# Vẽ heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Heatmap - Tương quan giữa các biến")
plt.show()
```



Heatmap - Tương quan giữa các biến



```
# Tạo biến "Tổng thu nhập gia đình"
df['TotalIncome'] = df['AnnualIncome'] * df['FamilyMembers']

# Tạo biến "Tỷ lệ thu nhập trên mỗi thành viên"
df['IncomePerMember'] = df['TotalIncome'] / df['FamilyMembers']
```

Tạo biến dummy

```
# Tạo biến dummy cho cột "Employment Type"
employment_dummies = pd.get_dummies(df['Employment Type'], prefix='Employment')
df = pd.concat([df, employment_dummies], axis=1)
```

Sử dụng các kỹ thuật mã hóa phức tạp hơn Ngoài `pd.get_dummies`, bạn có thể sử dụng các kỹ thuật mã hóa phức tạp hơn như:

Target encoding: Mã hóa biến phân loại dựa trên giá trị trung bình của biến mục tiêu (target variable) cho mỗi giá trị của biến phân loại.

Frequency encoding: Mã hóa biến phân loại dựa trên tần suất xuất hiện của mỗi giá trị. Binary encoding: Mã hóa biến phân loại thành dạng nhị phân. Ví dụ (Target encoding):

```
!pip install category_encoders
import category_encoders as ce

# Assuming 'TravelInsurance' is your target variable
X = df.drop('TravelInsurance', axis=1) # Features
y = df['TravelInsurance'] # Target

# Target encoding cho cột "Employment Type"
# Get the name of the encoded 'Employment Type' column
employment_type_encoded_col = X.select_dtypes(include='number').columns[X.columns.get_loc('Employment Type')]

encoder = ce.TargetEncoder(cols=[employment_type_encoded_col]) # Use the encoded column name
X = encoder.fit_transform(X, y) # Fit and transform on features (X)

# Now you can proceed with train-test split and model building using X and y
# ... (rest of your code) ...
```

Requirement already satisfied: category_encoders in /usr/local/lib/python3.11/dist-packages (2.8.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (2.0.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (1.0.1)
Requirement already satisfied: scikit-learn>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (1.6.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (1.14.1)

Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.11/dist-packages (from category_encoders) (0.14.4)
 Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
 Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.5->category_encoders) (2021.3)
 Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.0.5->category_encoders) (2022.7)
 Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.6.0->category_encoders) (1.3.2)
 Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.6.0->category_encoders) (3.2.0)
 Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels>=0.9.0->category_encoders) (23.1)
 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.5->category_encoders) (1.16.0)

df.head()

	Unnamed: 0	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	FrequentFlyer	EverTravelledAbroad	TravelInsurance
0	0.000000	0.6	0.0	1.0	0.083333	0.571429	1.0	0.0	0.0	0.0
1	0.000504	0.6	1.0	1.0	0.791667	0.714286	0.0	0.0	0.0	0.0
2	0.001007	0.9	1.0	1.0	0.166667	0.285714	1.0	0.0	0.0	0.0
3	0.001511	0.3	1.0	1.0	0.333333	0.142857	1.0	0.0	0.0	0.0
4	0.002014	0.3	1.0	1.0	0.333333	0.857143	1.0	1.0	0.0	0.0

✓ Apply machine learning model

✓ Train-test split

```
from sklearn.model_selection import train_test_split
RANDOM_STATE = 42
TRAIN_SIZE = 0.8

X = df.drop('TravelInsurance', axis=1) # Features
y = df['TravelInsurance'] # Target

trainX, testX, trainY, testY = train_test_split(X, y, train_size=TRAIN_SIZE, random_state=RANDOM_STATE)
```

✓ Build SK-learn model

```
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer # Import SimpleImputer

def apply_feature_engineering(df):
    """
    Apply all feature engineering to transform your data into number
    :param df: pandas DataFrame
    :return: pandas DataFrame
    """
    # Chuyển các cột kiểu object (chuỗi) thành số
    for col in df.select_dtypes(include='object').columns:
        df[col] = df[col].astype('category').cat.codes

    # Impute missing values with the most frequent value for each column
    # This is done before scaling to avoid data leakage during scaling
    imputer = SimpleImputer(strategy='most_frequent')
    df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

    # Chuẩn hóa dữ liệu về khoảng [0,1]
    scaler = MinMaxScaler()
    df[df.columns] = scaler.fit_transform(df[df.columns])

    return df
```

```

# ... (rest of your code) ...

def build_model(X, y):
    """
    Design your model and train it (including your best params)
    :param X: feature matrix
    :param y: target
    :return: a model
    """
    # Create pipeline
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('imputer', SimpleImputer(strategy='mean')), # Add imputation step to handle NaNs
        ('pca', PCA()), # PCA for dimensionality reduction
        ('classifier', GaussianNB()) # Gaussian Naive Bayes classifier
    ])

    # Define parameter grid for GridSearchCV
    param_grid = {
        'pca__n_components': [3, 5, 7], # Number of components for PCA
        'classifier__var_smoothing': [1e-9, 1e-8, 1e-7] # Smoothing parameter for GaussianNB
    }

    # Perform GridSearchCV to find best parameters
    grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1) # Use all available CPU cores
    grid_search.fit(X, y)

    # Return the best model found by GridSearchCV
    return grid_search.best_estimator_

def calculate_performance(y_true, y_pred):
    """
    Calculate and print various performance metrics.

    :param y_true: ground truth values
    :param y_pred: predictions
    :return: accuracy score
    """
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")
    print(classification_report(y_true, y_pred)) # Print detailed classification report

    return accuracy # Return accuracy as the main score

# Build and evaluate the model
model = build_model(trainX, trainY)
predictions = model.predict(testX)
calculate_performance(testY, predictions)

➡ Accuracy: 0.7688
Precision: 0.7426
Recall: 0.5319
F1-score: 0.6198

```

	precision	recall	f1-score	support
0.0	0.78	0.90	0.83	257
1.0	0.74	0.53	0.62	141
accuracy			0.77	398
macro avg	0.76	0.72	0.73	398
weighted avg	0.77	0.77	0.76	398

```

0.7688442211055276

def get_conclusion():
    # ... (Các phần trước đó vẫn giữ nguyên) ...

    return "7. " \
        "Kết luận khác: " \
        "Mô hình được xây dựng sử dụng Gaussian Naive Bayes kết hợp với PCA để giảm chiều dữ liệu. " \
        "SimpleImputer được sử dụng để xử lý các giá trị bị thiếu, thay thế chúng bằng giá trị trung bình của cột. " \
        "GridSearchCV được áp dụng để tìm kiếm các tham số tối ưu cho mô hình, nhằm cải thiện hiệu suất. " \
        "Tuy nhiên, hiệu suất của mô hình có thể được cải thiện hơn nữa bằng cách thử nghiệm với các kỹ thuật feature engineering khác. " \
        "**Chỉ số chính:** Trong bài toán này, chỉ số **Recall** làm chỉ số chính. Lý do là vì mục tiêu của công ty bảo hiểm là xác

```