
รายงาน

เรื่อง Heart attack Risk Dataset

จัดทำโดย

นายชัยพร พูลสวัสดิ์ 6530200096

นายภควัต จิตรพรทรัพย์ 6530200321

นายรัตนพงศ์ ม่วงกระโทก 6530200410

นายวัชรากร รัศมีดิษฐ์ 6530200444

หลักสูตรวิทยาศาสตร์บัณฑิต สาขาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ ศรีราชา

ภาคปลาย ปีการศึกษา 2567

## 2 Class

ดาวโหลด Library ที่จำเป็น

```
[6] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.preprocessing import LabelEncoder, StandardScaler
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.naive_bayes import GaussianNB
    from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier
    from xgboost import XGBClassifier
    from imblearn.over_sampling import SMOTE, ADASYN
    from sklearn.metrics import confusion_matrix
    from imblearn.under_sampling import RandomUnderSampler
    from imblearn.pipeline import Pipeline
```

ดาวโหลด Dataset Heart attack risk และตัดความเสี่ยงระดับ Moderate ออก เพื่อจำแนกแค่ 2 คลาส และแสดงตัวอย่าง

```
# โหลดข้อมูล
df = pd.read_csv('/content/heart_attack_risk_dataset.csv')

# กรองข้อมูล: ตัดความเสี่ยงระดับ Moderate ออก
df = df[df["Heart_Attack_Risk"] != "Moderate"]
print(df["Heart_Attack_Risk"].value_counts())
df.head(5)
```

```
Heart_Attack_Risk
Low    25024
High   10072
Name: count, dtype: int64
```

| | Age | Gender | Smoking | Alcohol_Consumption | Physical_Activity_Level | BMI | Diabetes | Hypertension | Cholesterol_Level | Resting_BP | Heart_Rate | Family_History | Stress_Level |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 69 | Female | 1 | 0 | Moderate | 34.61 | 1 | 0 | 152.1 | 171 | 85 | 0 | Moderate |
| **2** | 89 | Male | 0 | 1 | Moderate | 35.32 | 0 | 0 | 272.3 | 123 | 127 | 0 | Low |
| **3** | 78 | Male | 0 | 1 | Moderate | 18.23 | 1 | 0 | 237.7 | 144 | 125 | 0 | Low |
| **5** | 41 | Male | 0 | 1 | Moderate | 36.11 | 0 | 0 | 271.2 | 141 | 119 | 0 | Low |
| **6** | 20 | Male | 1 | 0 | Low | 15.12 | 0 | 0 | 164.8 | 154 | 67 | 0 | Low |

สร้างคอลัมน์ BMI_Category และ Age_Categoty ที่แปลงค่าในคอลัมน์ BMI และ Age ให้เป็น String และหลังจากนั้น Encode ให้เป็น Int ตามลำดับ ลบคอลัมน์ BMI และ AGE แทนค่าในคอลัมน์ Heart_attack_risk ให้เป็น Binary และจัดการ MissingValues โดย คอลัมน์ที่เป็น Numerical จะแทนด้วยค่าเฉลี่ย และคอลัมนท์ที่เป็น Categorical จะแทนด้วย mode

```
[8] # Feature Engineering
    label_enc = LabelEncoder()
    for col in df.select_dtypes(include=['object']).columns:
        if col != "Heart_Attack_Risk":  # ไม่ต้องทำ Label Encoding สำหรับ Heart_Attack_Risk
            df[col] = label_enc.fit_transform(df[col])

    df["BMI_Category"] = pd.cut(df["BMI"], bins=[0, 18.5, 25, 30, 100], labels=["Underweight", "Normal", "Overweight", "Obese"])
    df["Age_Group"] = pd.cut(df["Age"], bins=[0, 30, 50, 100], labels=["Young", "Middle", "Senior"])
    df["BMI_Category"] = label_enc.fit_transform(df["BMI_Category"])
    df["Age_Group"] = label_enc.fit_transform(df["Age_Group"])

    # แทนค่า Heart_Attack_Risk เป็นไบนารี: Low -> 0, High -> 1
    df["Heart_Attack_Risk"] = df["Heart_Attack_Risk"].map({"Low": 0, "High": 1})

    # แก้ไข Missing Values
    numerical_cols = df.select_dtypes(include=['number']).columns
    categorical_cols = df.select_dtypes(include=['object']).columns

    df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].mean())

    if not categorical_cols.empty:
        for col in categorical_cols:
            mode_values = df[col].mode()
            if not mode_values.empty:
                df[col] = df[col].fillna(mode_values[0])
            else:
                df[col].fillna("Unknown", inplace=True)
```

ตัวอย่างข้อมูล

| | Gender | Smoking | Alcohol_Consumption | Physical_Activity_Level | Diabetes | Hypertension | Cholesterol_Level | Resting_BP | Heart_Rate | Family_History | Stress_Level | Chest_Pain_T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 2 | 1 | 0 | 152.1 | 171 | 85 | 0 | 2 | |
| **2** | 1 | 0 | 1 | 2 | 0 | 0 | 272.3 | 123 | 127 | 0 | 1 | |
| **3** | 1 | 0 | 1 | 2 | 1 | 0 | 237.7 | 144 | 125 | 0 | 1 | |
| **5** | 1 | 0 | 1 | 2 | 0 | 0 | 271.2 | 141 | 119 | 0 | 1 | |
| **6** | 1 | 1 | 0 | 1 | 0 | 0 | 164.8 | 154 | 67 | 0 | 1 | |

แยก Feature และ Target โดยทำ **One-Hot Encoding**: สำหรับคอลัมน์ categorical ที่เหลือ Gender, Chest_Pain_Type, Thalassemia, ECG_Results หลังจากนั้นจึงแบ่งข้อมูลไว้สำหรับ Test และ Train

```
# แยก Features และ Target
target_col = "Heart_Attack_Risk"
X = df.drop(columns=[target_col])
X = pd.get_dummies(X, columns=["Gender", "Chest_Pain_Type", "Thalassemia", "ECG_Results"])
y = df[target_col]

# แบ่งข้อมูลแบบ Stratified Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# ทำ Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

ปรับสเกล Features ให้มีค่าเฉลี่ยเป็น 0 และส่วนเบี่ยงเบนมาตรฐานเป็น 1 และจึงปรับสมดุลข้อมูลเนื่องจาก Class Low มีจำนวน 20,019และ Class High มีจำนวน 8057 โดยใช้วิธี เพิ่มจำนวน Class High 50% และลดจำนวน Class Low 80% จากนั้นจึงสร้าง pipeline ให้ทำตามขั้นตอน

```
# ทำ Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

over = SMOTE(sampling_strategy=0.5, random_state=42)  # เพิ่มคลาส "1" ให้เป็น 50% ของคลาส "0"
under = RandomUnderSampler(sampling_strategy=0.8, random_state=42)  # ลดคลาส "0" ให้เป็น 80% ของคลาส

# สร้าง Pipeline
steps = [('over', over), ('under', under)]
pipeline = Pipeline(steps=steps)

# ปรับสมดุลข้อมูล
X_train_res, y_train_res = pipeline.fit_resample(X_train, y_train)
```

ตัวอย่างข้อมูลหลังจากปรับสมดุลแล้ว

```
Before resampling: Heart_Attack_Risk
0   20019
1    8057
Name: count, dtype: int64
After resampling: Heart_Attack_Risk
0   12511
1   10009
Name: count, dtype: int64
```

จากนั้นจึงสร้างและฝึกโมเดลโดยใช้ Naïve Bayes , Logistic Regression, Random Forest , XGBoost และการประเมินผลใช้ Accuracy , Classification Report (Precision, Recall, F1-Score) , Confusion Matrix

```
models = {
    "Naïve Bayes": GaussianNB(),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=10, class_weight='balanced', random_state=42),
    "XGBoost": XGBClassifier(scale_pos_weight=len(y_train[y_train == 0]) / len(y_train[y_train == 1]), n_estimators=200, max_depth=5, random_state=42),
}

results = {}
for name, model in models.items():
    model.fit(X_train_res, y_train_res)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"\n 🔹 {name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
    results[name] = acc
    cm = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:")
    print(cm)
    print("-" * 30)
```

## ตัวอย่าง Naïve Bayes

```
🔹 Naïve Bayes Accuracy: 0.6588
              precision    recall  f1-score   support

           0       0.71      0.88      0.79      5005
           1       0.27      0.11      0.15      2015

    accuracy                           0.66      7020
   macro avg       0.49      0.49      0.47      7020
weighted avg       0.58      0.66      0.60      7020

Confusion Matrix for Naïve Bayes:
[[4410  595]
 [1800  215]]
```

## ตัวอย่าง Logistic Regression

```
🔹 Logistic Regression Accuracy: 0.7103
              precision    recall  f1-score   support

           0       0.71      0.99      0.83      5005
           1       0.21      0.00      0.01      2015

    accuracy                           0.71      7020
   macro avg       0.46      0.50      0.42      7020
weighted avg       0.57      0.71      0.59      7020

Confusion Matrix for Logistic Regression:
[[4979   26]
 [2008    7]]
```

## ตัวอย่าง Random Forest

```
🔹 Random Forest Accuracy: 0.6345
              precision    recall  f1-score   support

           0       0.71      0.82      0.76      5005
           1       0.28      0.17      0.21      2015

    accuracy                           0.63      7020
   macro avg       0.49      0.50      0.49      7020
weighted avg       0.59      0.63      0.60      7020

Confusion Matrix for Random Forest:
[[4114  891]
 [1675  340]]
```

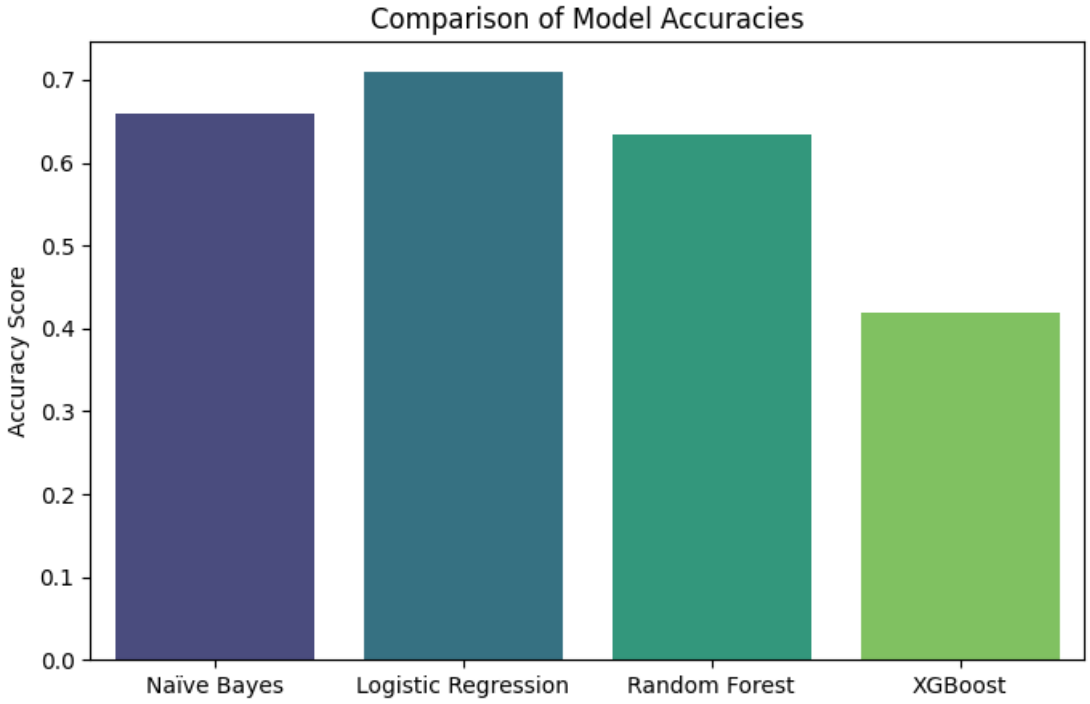
## ตัวอย่าง XGBoost

```
🔹 XGBoost Accuracy: 0.4181
              precision    recall  f1-score   support

           0       0.72      0.30      0.43      5005
           1       0.29      0.70      0.41      2015

    accuracy                           0.42      7020
   macro avg       0.50      0.50      0.42      7020
weighted avg       0.59      0.42      0.42      7020

Confusion Matrix for XGBoost:
[[1515 3490]
 [ 595 1420]]
```

## เปรียบเทียบ Accuracy แต่ละโมเดลโดยใช้กราฟแท่ง

```
plt.figure(figsize=(8, 5))
sns.barplot(x=list(results.keys()), y=list(results.values()), palette="viridis")
plt.title("Comparison of Model Accuracies")
plt.ylabel("Accuracy Score")
plt.show()
```

ตัวอย่างกราฟ


Comparison of Model Accuracies
Accuracy Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Naïve Bayes
Logistic Regression
Random Forest
XGBoost