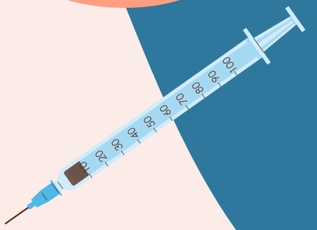
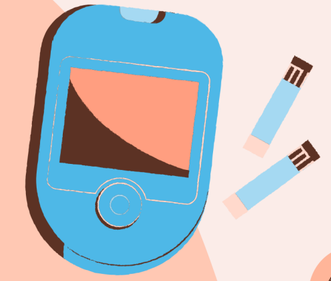


DIABETES STATUS CLASSIFICATION

Nguyễn Xuân Thành - Nhóm 3



TODAY'S AGENDA

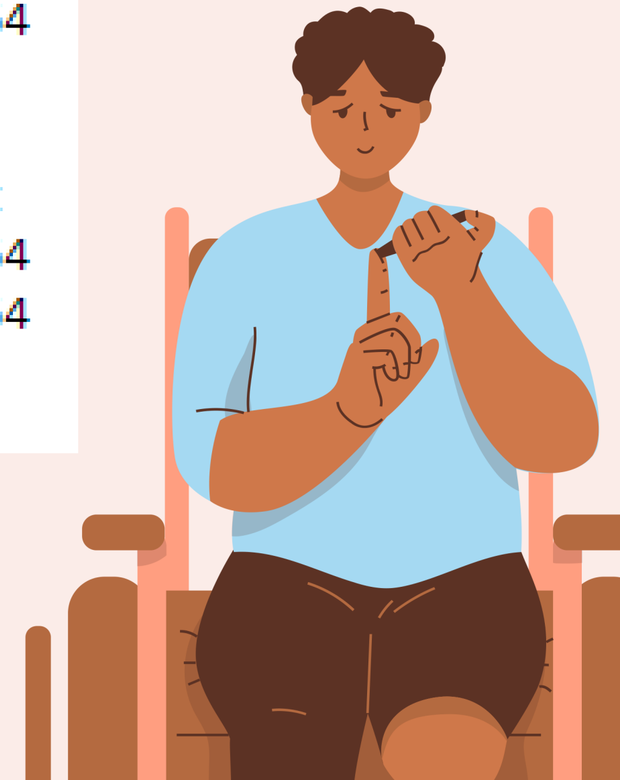
- Khám phá dữ liệu
- Làm sạch dữ liệu
- Mã hoá và chuẩn hoá dữ liệu
- Xây dựng mô hình
- Lựa chọn mô hình tối ưu

TỔNG QUAN VỀ DỮ LIỆU

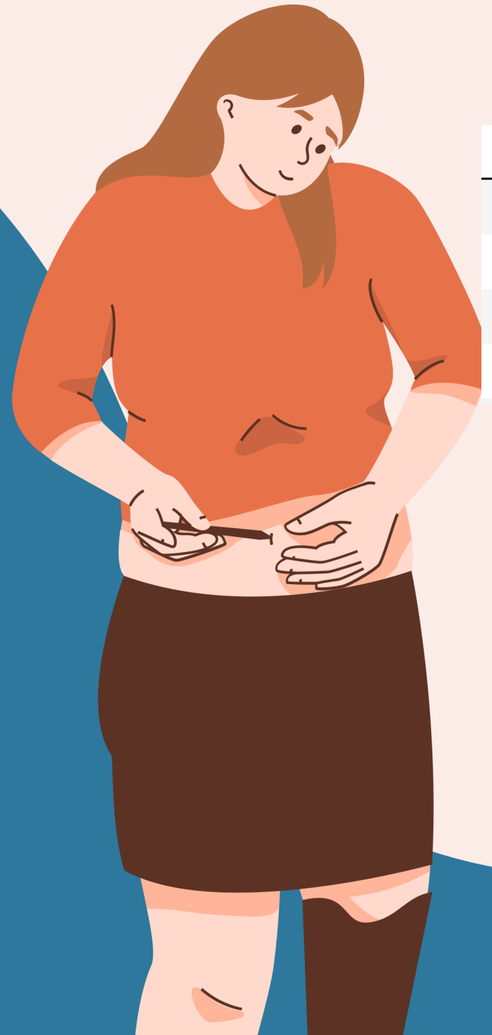
- Có nguồn gốc từ Electronic Health Records(EHR)
- Dùng để xây dựng mô hình học máy
- Nghiên cứu trong việc tìm ra nguyên nhân gây ra tiểu đường

Thông tin về dữ liệu

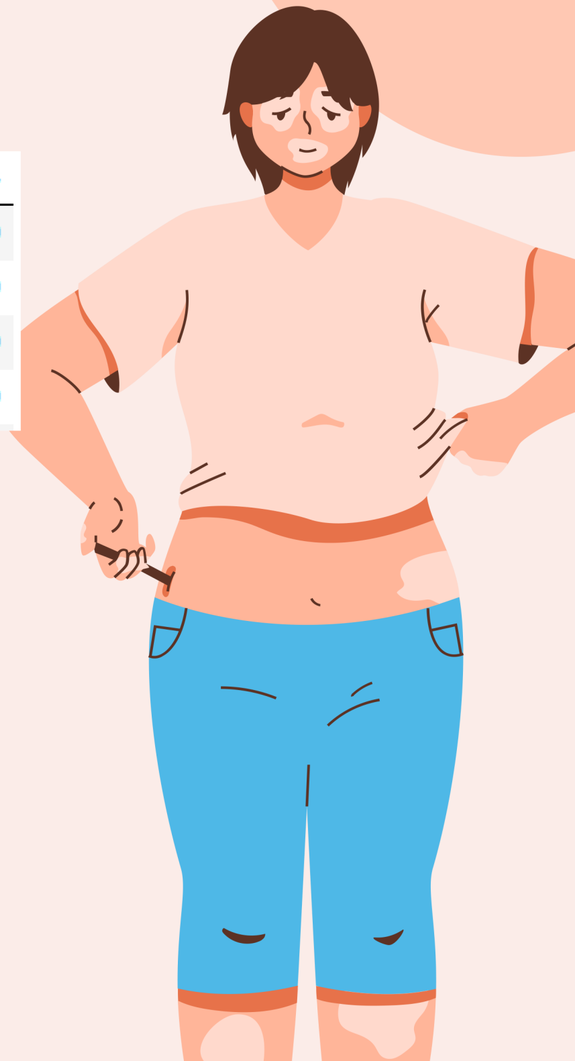
```
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
```



CHI TIẾT



gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
Female	80.0	0	1	never	25.19	6.6	140	0
Female	54.0	0	0	No Info	27.32	6.6	80	0
Male	28.0	0	0	never	27.32	5.7	158	0
Female	36.0	0	0	current	23.45	5.0	155	0



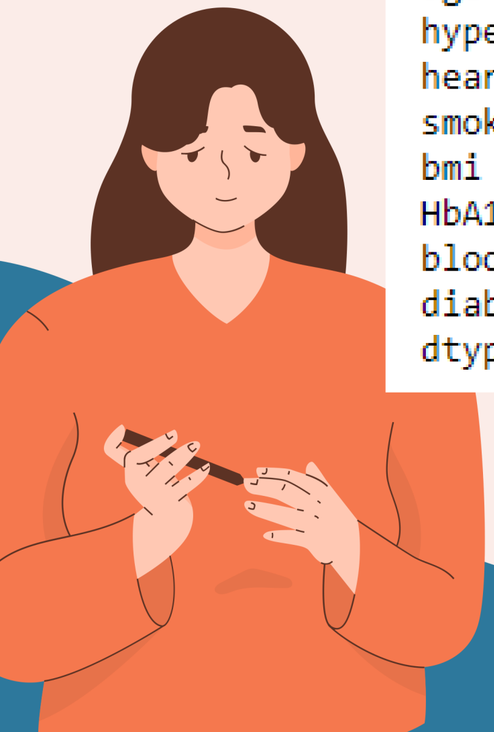
LÀM SẠCH DỮ LIỆU

NaN variable

Check NaN variable

```
df.isna().sum()
```

```
gender                0
age                   0
hypertension          0
heart_disease         0
smoking_history       0
bmi                   0
HbA1c_level           0
blood_glucose_level   0
diabetes              0
dtype: int64
```

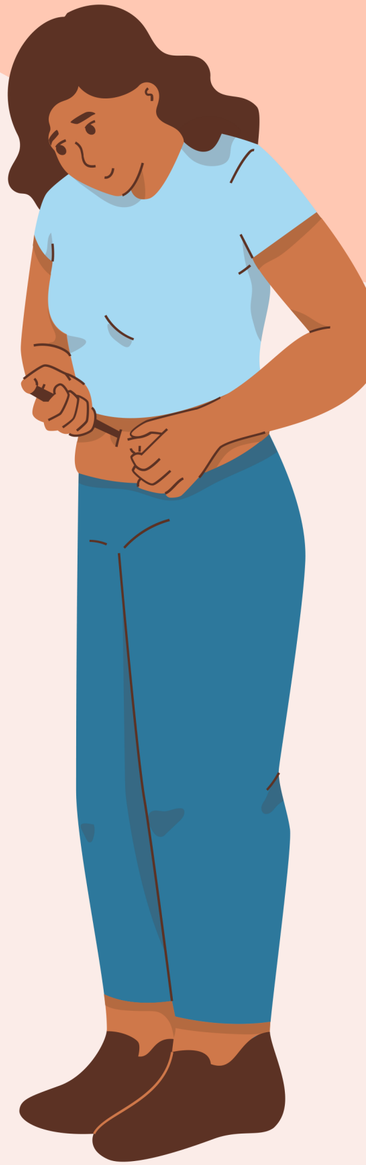


Null variable

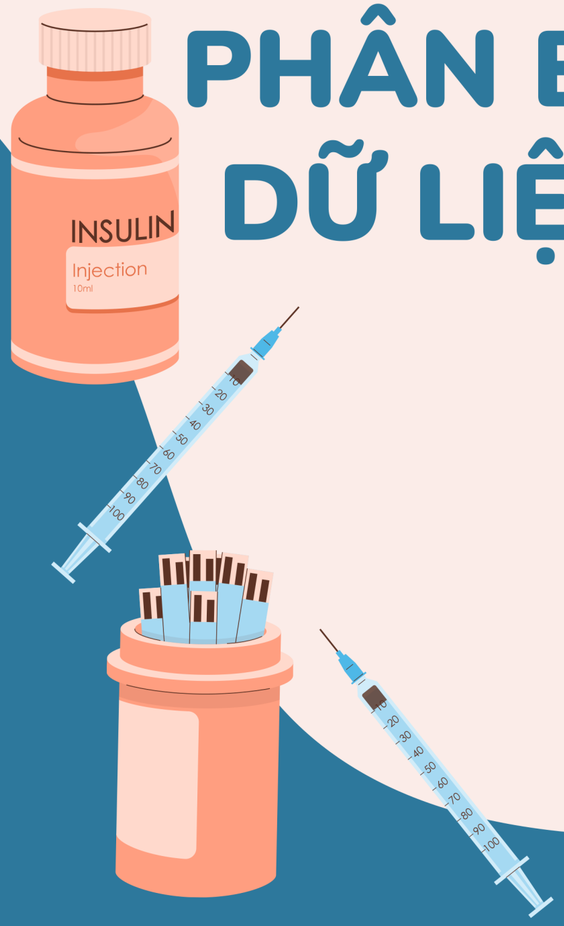
Check null variable

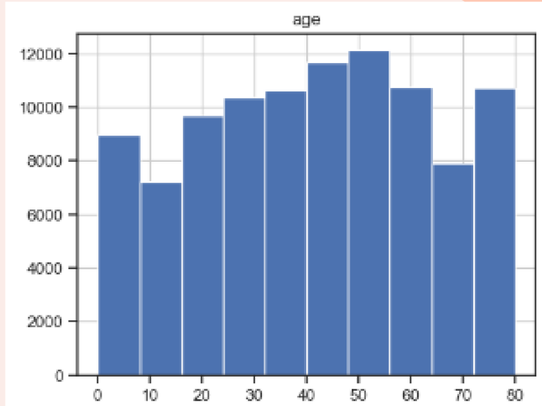
```
df.isnull().sum()
```

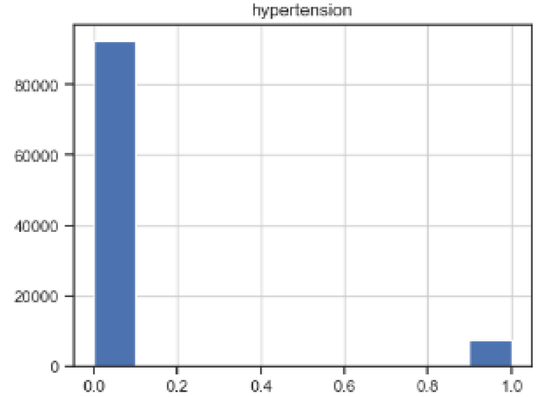
```
gender                0
age                   0
hypertension          0
heart_disease         0
smoking_history       0
bmi                   0
HbA1c_level           0
blood_glucose_level   0
diabetes              0
dtype: int64
```

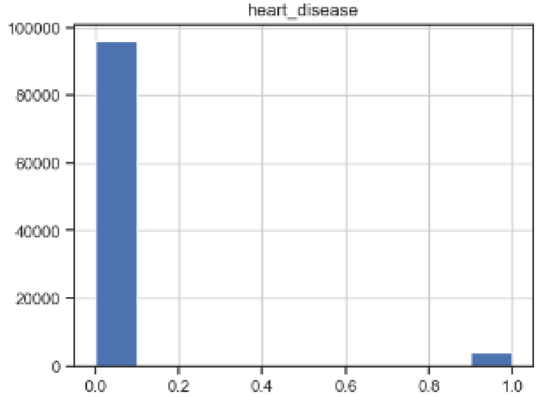


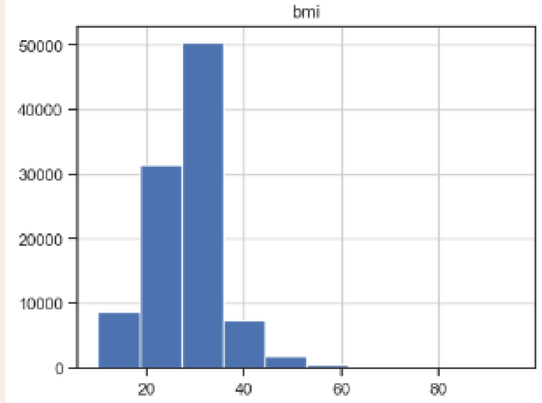
PHÂN BỐ DỮ LIỆU

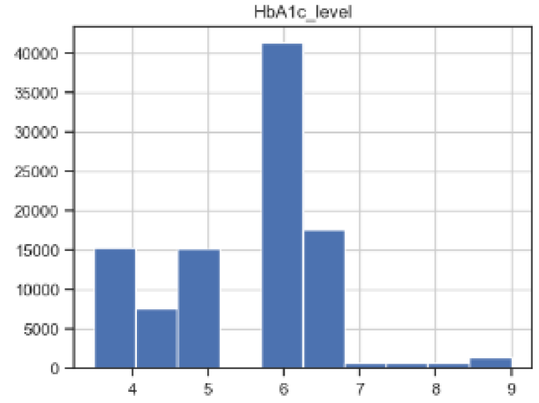


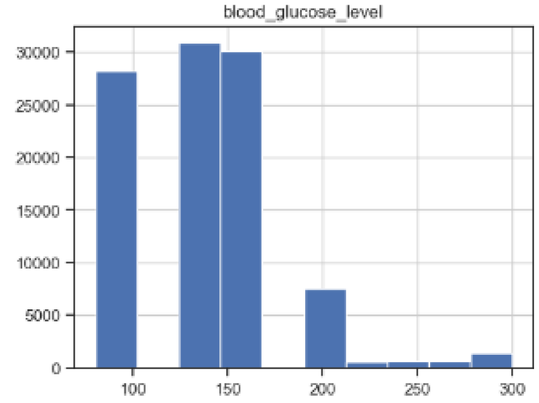


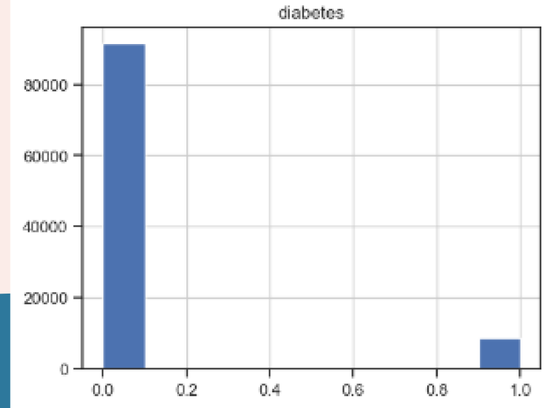


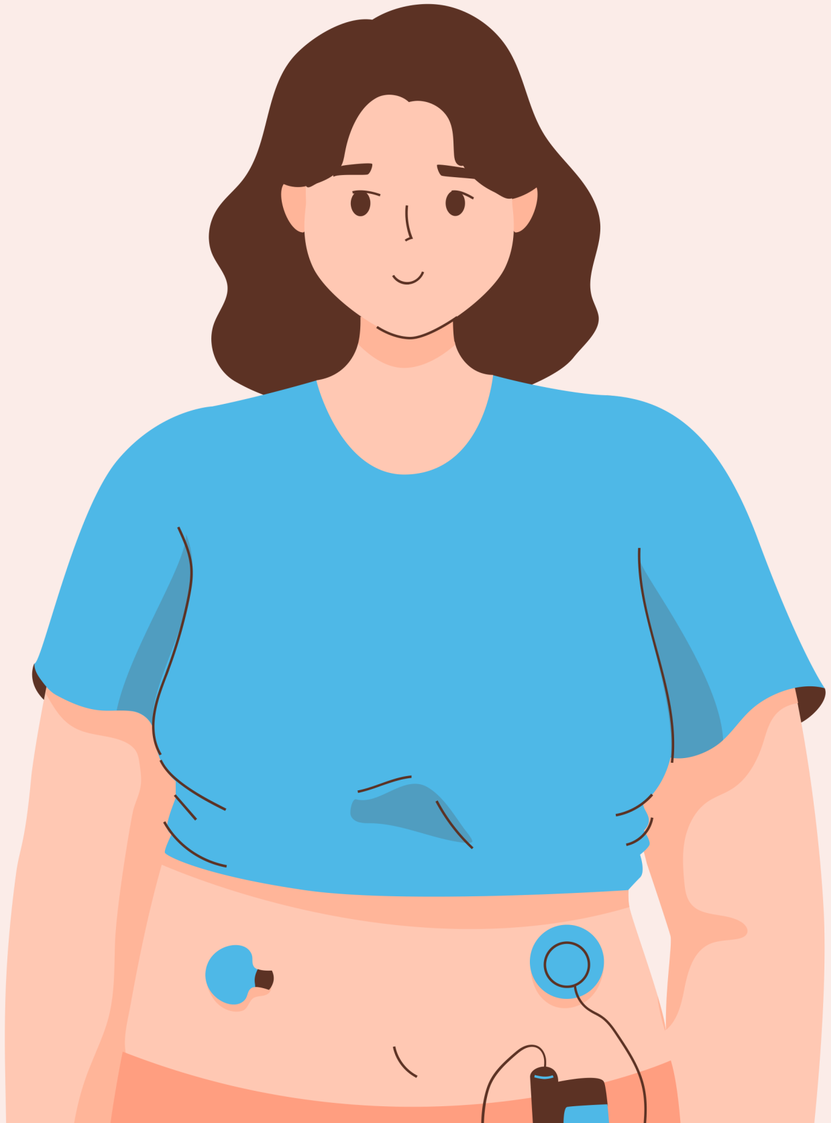






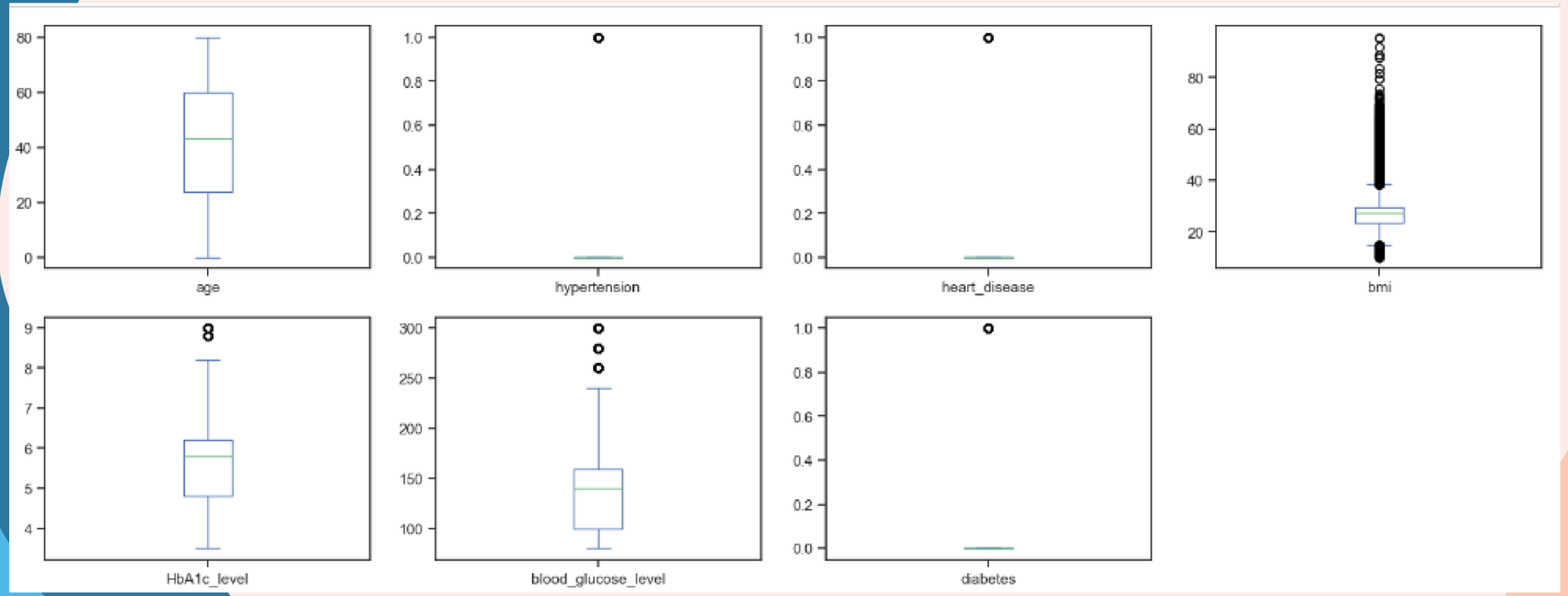


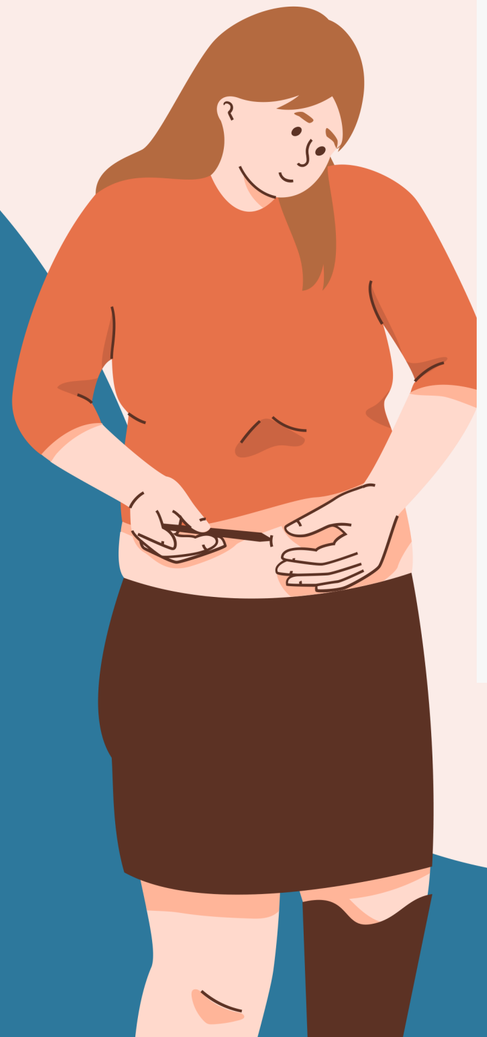




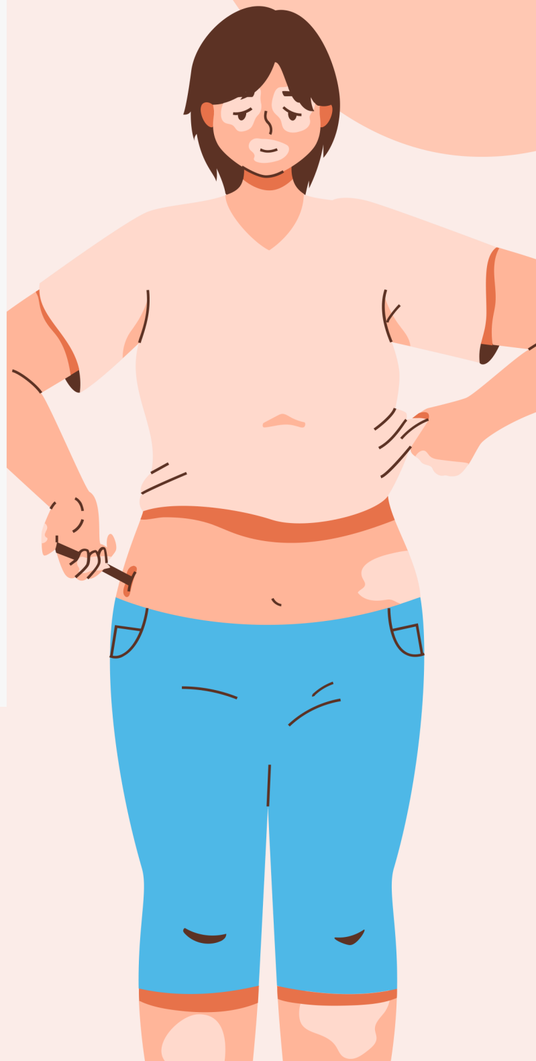
TÌM VÀ BỎ CÁC DỮ LIỆU NGOẠI LAI

KIỂM TRA DỮ LIỆU NGOẠI LAI

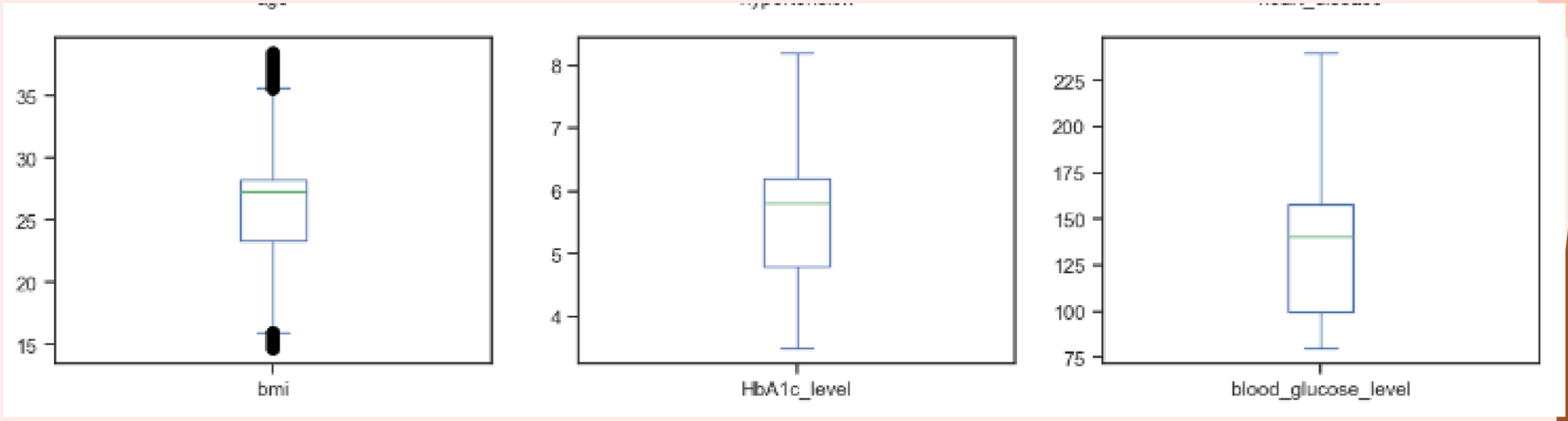


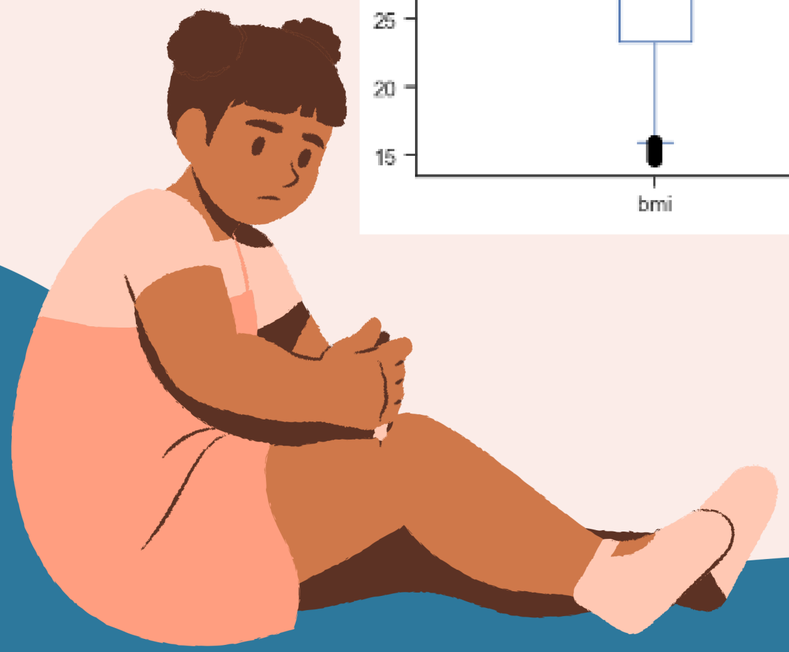


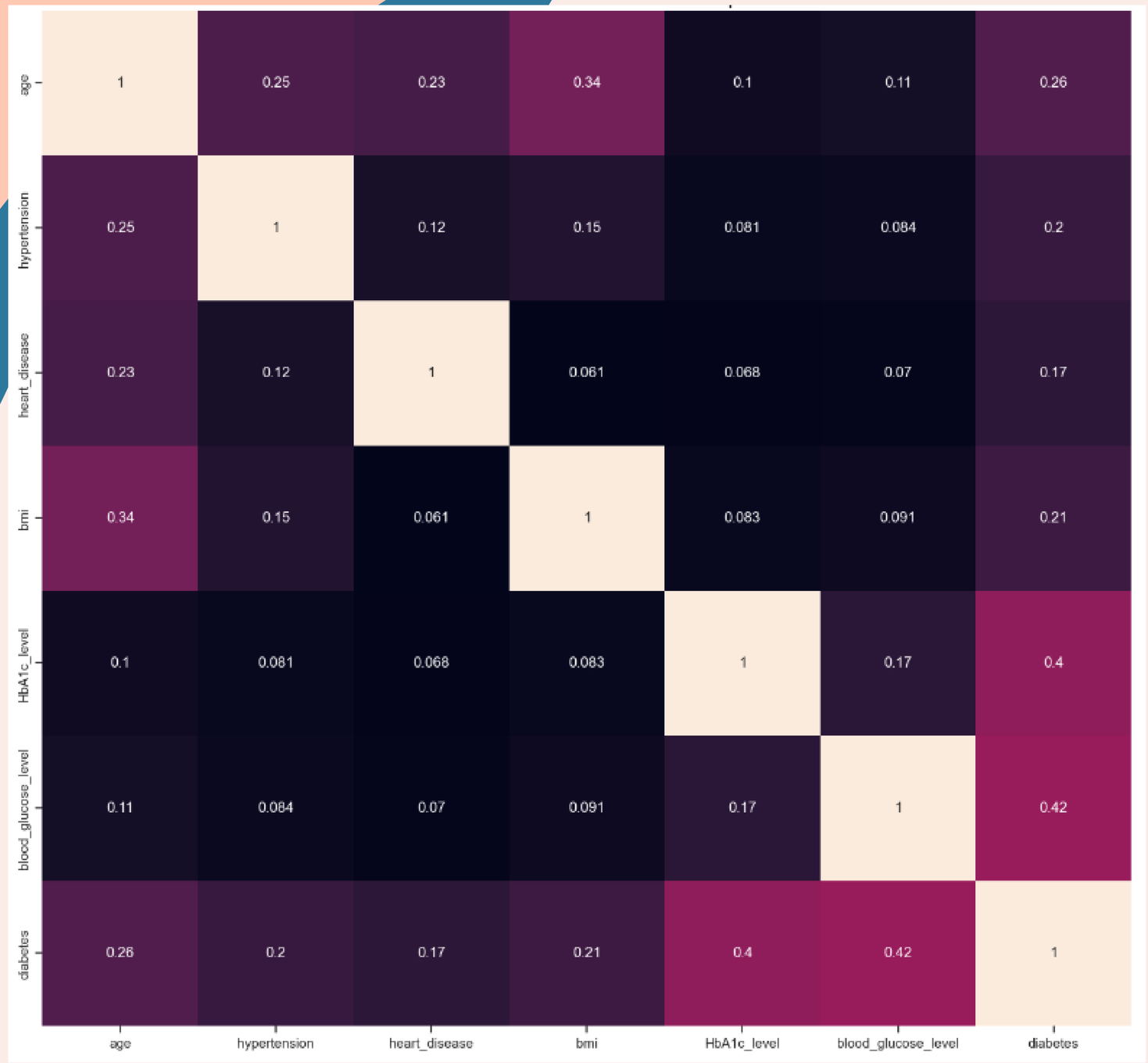
```python
def drop_outlier(df,column):
    # Calculate the quartiles and IQR
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1

    # Set a threshold for outlier detection (e.g., 1.5)
    threshold = 1.5

    # Determine the lower and upper bounds
    lower_bound = Q1 - threshold * IQR
    upper_bound = Q3 + threshold * IQR

    # Drop rows with values outside the lower and upper bounds
    df = df[(column >= lower_bound) & (column <= upper_bound)]
    return df
```



SAU KHI LOẠI BỎ







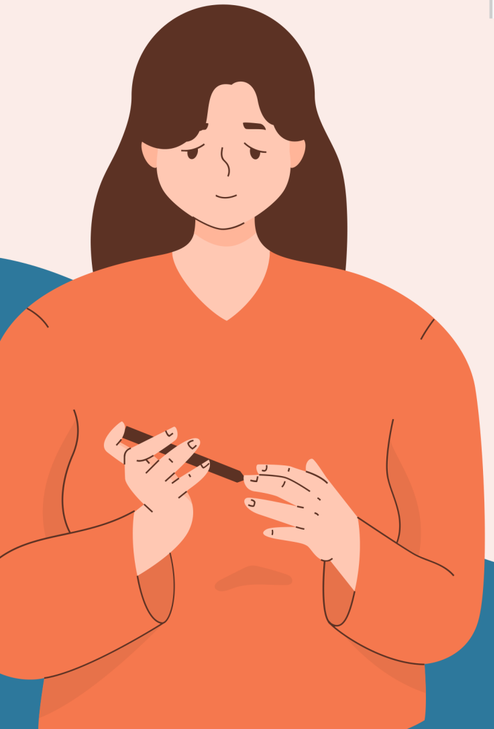
TƯƠNG QUAN GIỮA CÁC ĐẶC TRƯNG

Mã hoá

```
obj_col
```

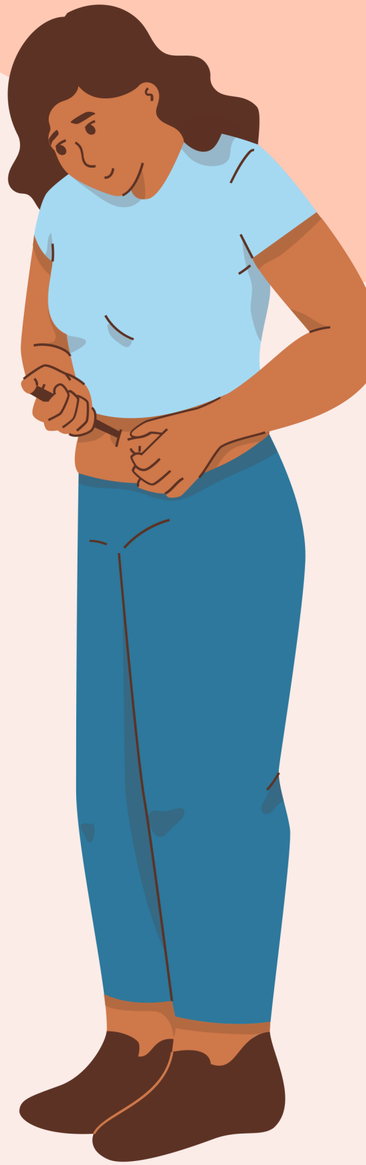
```
['gender', 'smoking_history']
```

```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df[obj_col] = df[obj_col].apply(le.fit_transform)
df
```



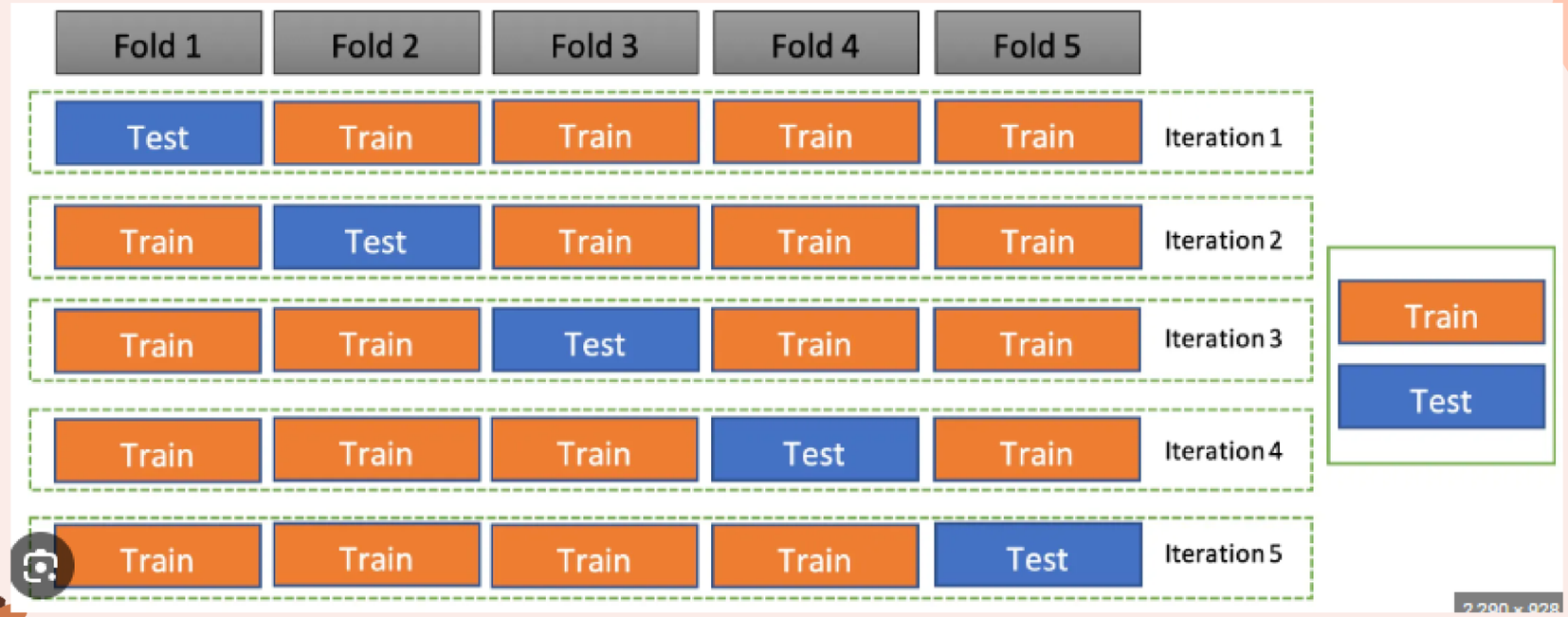
Chuẩn hoá

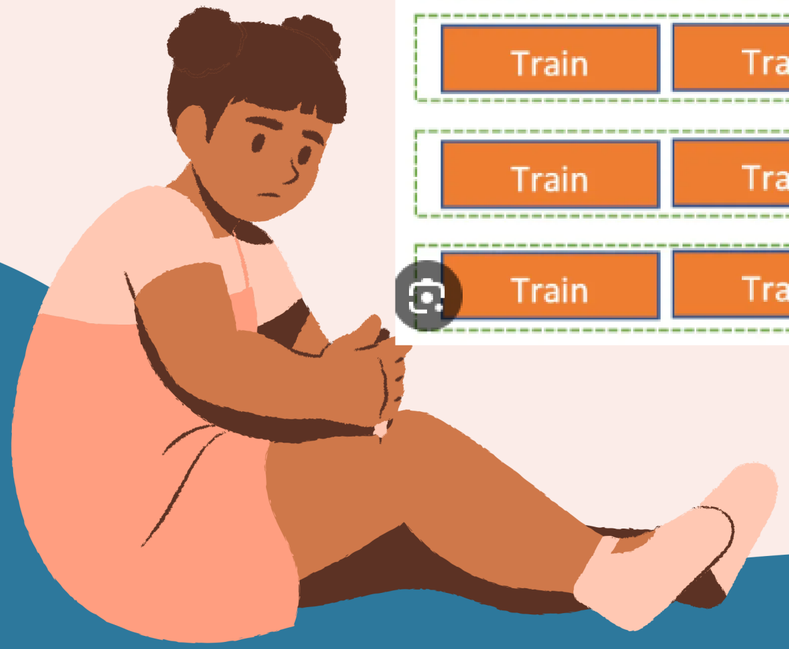
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df)
data_minmax= scaler.transform(df)
```



PHÂN TÁCH TẬP DỮ LIỆU

K - FOLD SPLITTING





```
# !conda install scikit-learn
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,f1_score,accuracy_score,precision_score,recall_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Initialize the KFold object
kf = KFold(n_splits=5)

# Loop over each fold
for train_index, test_index in kf.split(x):
    # Get the training and testing data for this fold
    x_train, x_test = x.iloc[list(train_index)], x.iloc[list(test_index)]
    y_train, y_test = y.iloc[list(train_index)], y.iloc[list(test_index)]
```

Using a 5-fold cross-validation is a common choice in machine learning because it strikes a good balance between model performance estimation and computational efficiency

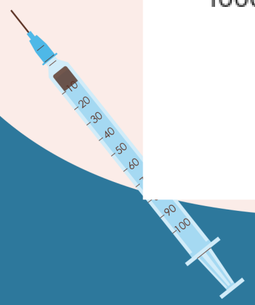
```
[ ] print(x_train.shape)
    print(y_train.shape)
    print(x_test.shape)
    print(y_test.shape)
```

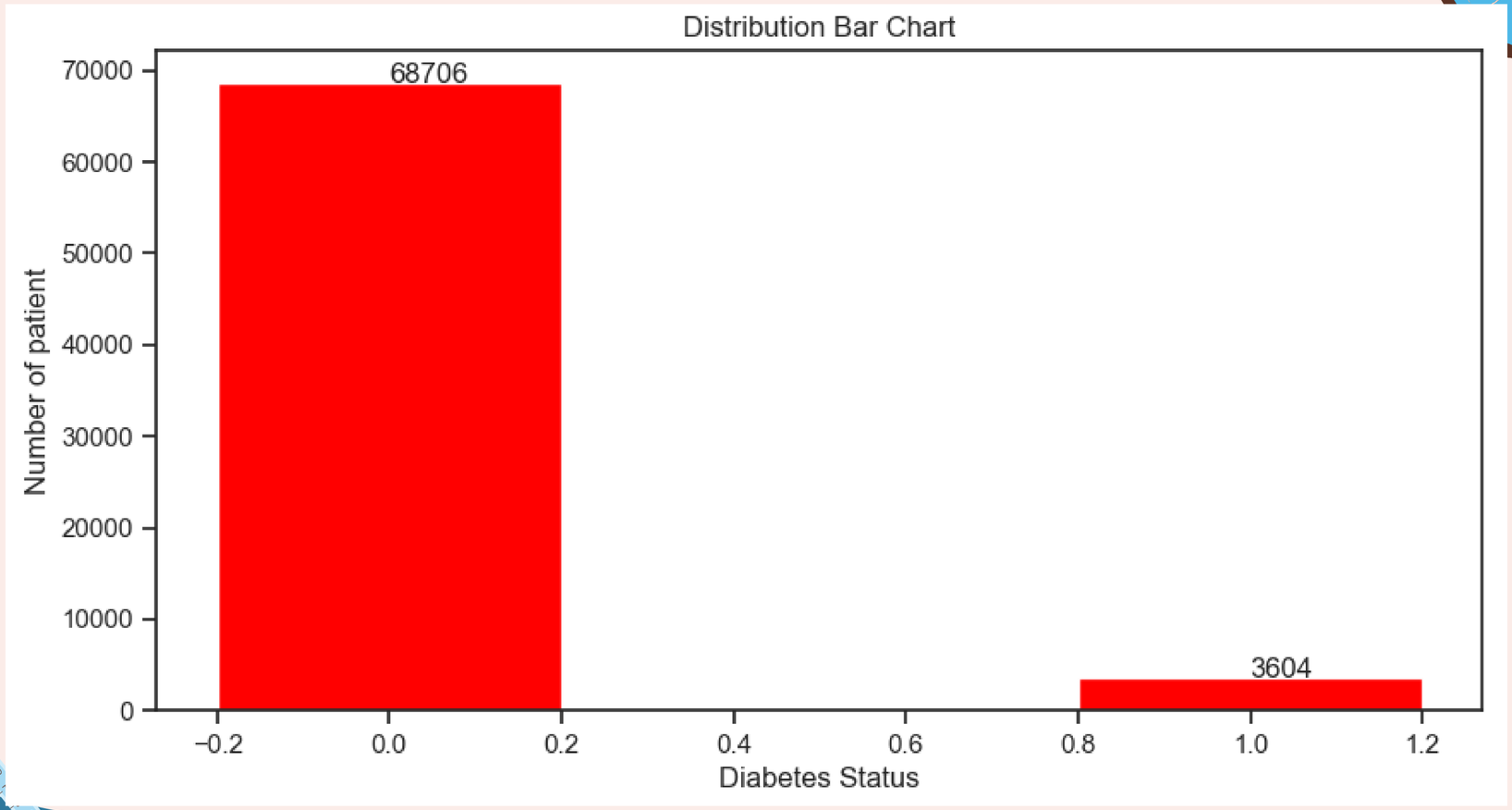
```
(72310, 8)
(72310,)
(18077, 8)
(18077,)
```

LỰA CHỌN THUẬT TOÁN

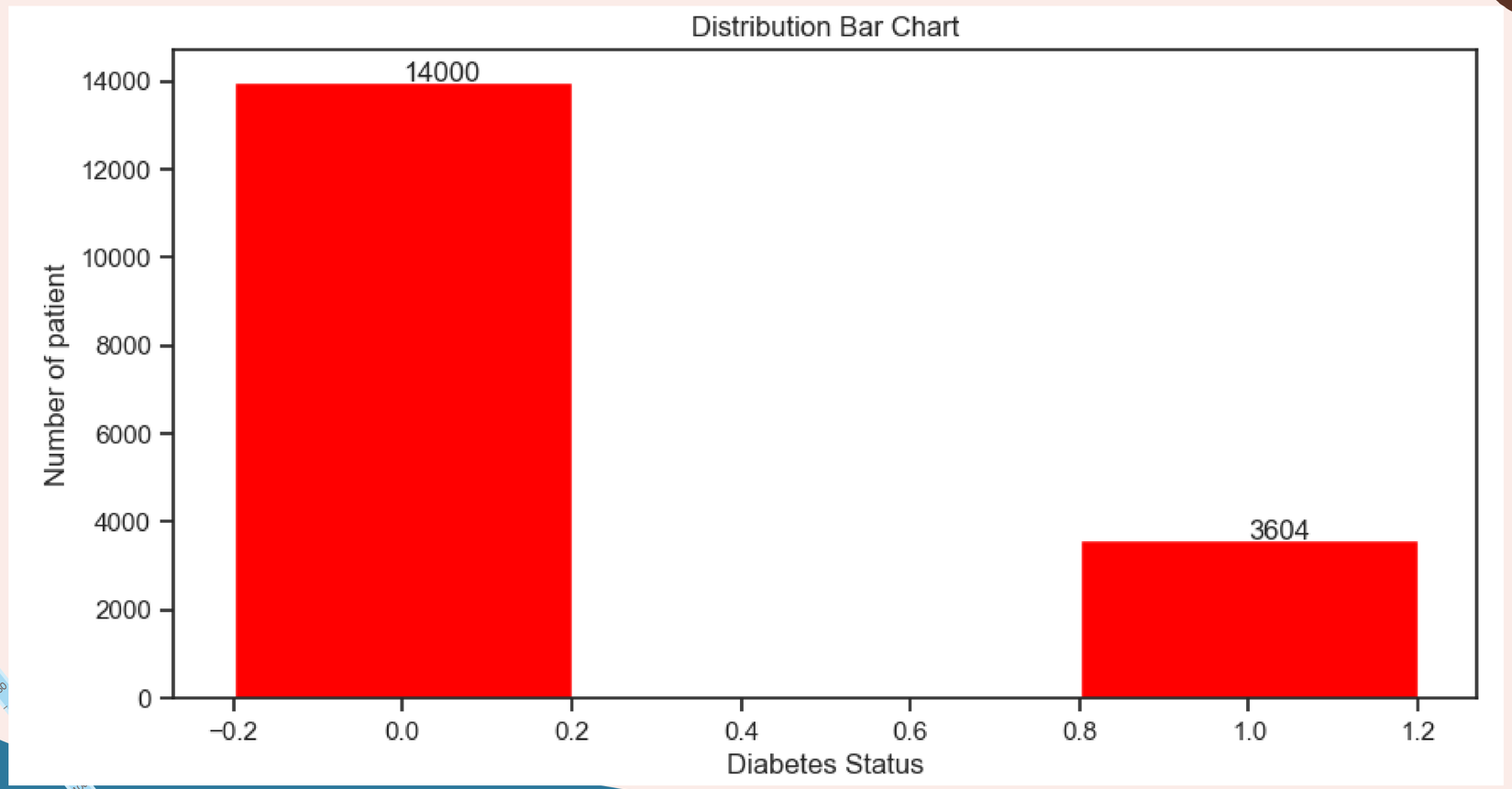
- LGBM
- LOGISTIC REGRESSION
- DECISION TREE
- GRADIENT BOOSTING
- RANDOM FOREST

TRƯỚC KHI UNDERSAMPLING





SAU KHI UNDERSAMPLING



LGBM

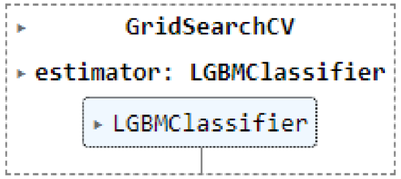
4.2 LGBM

```
[ ]  from sklearn.ensemble import GradientBoostingClassifier
     from lightgbm import LGBMClassifier

     lgbm = LGBMClassifier(n_estimators=100)
```

```
[ ]  ##Parameter for grid tunning

     # Create the param list
     param_list = {'learning_rate': [0.1, 0.05, 0.01],
                   'n_estimators': [100, 200, 300],
                   'max_depth': [3, 5, -1],
                   'num_leaves': [31, 50, 100]}
```

```
[ ]  lgbm_Grid = GridSearchCV(estimator = lgbm, param_grid
                              = param_list)
```

```
[ ]  lgbm_Grid.fit(x_over, y_over)
```




```
[ ]  # Determine the best parameters
     print(lgbm_Grid.best_params_)
     print(lgbm_Grid.best_score_)
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'num_leaves': 31}
0.915019574113759
```

LOGISTIC REGRESSION

4.3 Logistic Regression

```
[ ]  from sklearn.linear_model import LogisticRegression

     # define model
     model = LogisticRegression(random_state=42)
     # define evaluation
     #We have learn that for 2 fold yield the best result
     # define parameter grid
     parameters_list= {
         'penalty': ['l1', 'l2', 'None'],
         'C': [0.1, 1, 10],
         'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
         'max_iter': [100, 500, 1000]
     }

     # define search
     search =GridSearchCV(model, parameters_list)
     # execute search
     result = search.fit(x_over, y_over)
     # summarize result
     pred = search.predict(x_test)
```

DECISION TREE

4.4 Decision Tree

```
[ ]  from sklearn.tree import DecisionTreeClassifier
     from sklearn import tree

     dtree = tree.DecisionTreeClassifier(random_state=42)
     dtree.fit(x_over, y_over)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
[ ]  # Create the parameter grid
     params_distribution= {
                  'criterion': ['gini', 'entropy'],
                  'max_depth': [None, 5, 10, 50, 100],
                  'min_samples_split': [2, 5, 10],
                  'min_samples_leaf': [ 2, 5, 10],
                  'max_features': [None, 'sqrt', 'log2']}
```

```
[ ]  dt = DecisionTreeClassifier(random_state=42)
     # Instantiate the grid search model
     grid_search = GridSearchCV(estimator=dt,
                                param_grid = params_distribution)

     #Execute search
     grid_search.fit(x_over, y_over)
     print(grid_search.best_params_)
     print(grid_search.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 10, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 2}
0.9055328741834707
```

GRADIENT BOOSTING

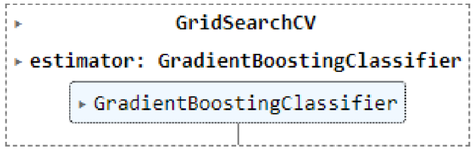
4.5 Gradient Boosting

```
[ ]  gbc = GradientBoostingClassifier(n_estimators=100)
```

```
[ ]  ##Parameter for grid tunning

     # Create the param list
     param_list = {'learning_rate': [0.1, 0.05, 0.01],
                   'n_estimators': [100, 200, 300],
                   'max_depth': [3, 5, None],
                   'min_samples_split': [2, 5, 10],
                   'min_samples_leaf': [1, 2, 4],
                   'max_features': ['sqrt', 'log2', None]}
```

```
[ ]  gbc_Grid = GridSearchCV(estimator = gbc, param_grid
                             = param_list)
```

```
[ ]  gbc_Grid.fit(x_over, y_over)
```




```
[ ]  # Determine the best parameters
     print(gbc_Grid.best_params_)
     print(gbc_Grid.best_score_)
```

```
{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}
0.9152467016085307
```

RANDOM FOREST

4.6 Random Forest

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     rfc = RandomForestClassifier(random_state=42,max_features  ="a")
     # rfc._parameter
```

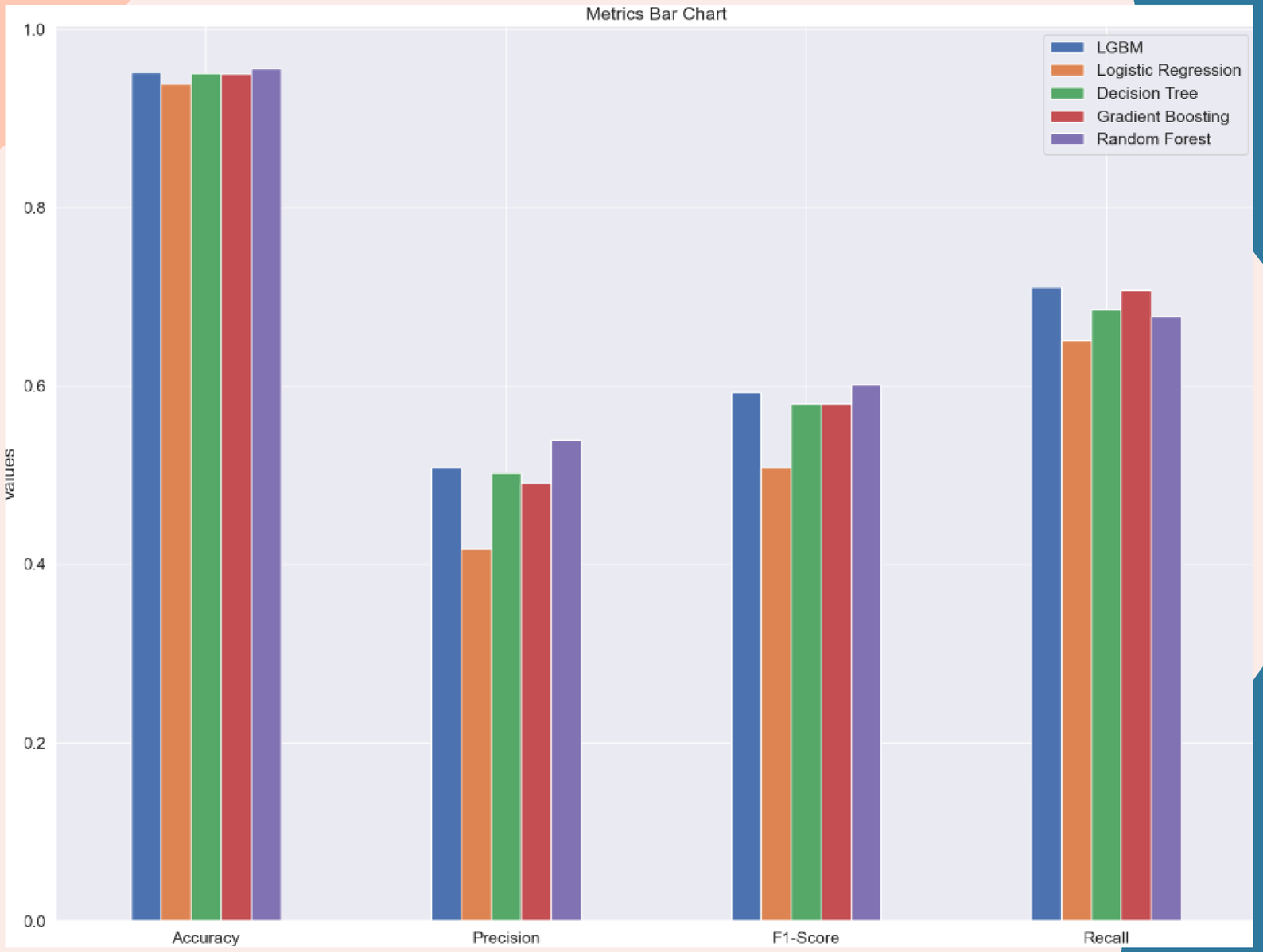
```
[ ]  ##Parameter for grid tunning

     # Create the param list
     param_list = {'n_estimators': [100,300],
                   'max_features': ['auto', 'sqrt'],
                   'max_depth': [5,10,20],
                   'min_samples_split': [2,3,5],
                   'min_samples_leaf': [1, 2, 4],
                   'criterion': ['gini','entropy']}
```

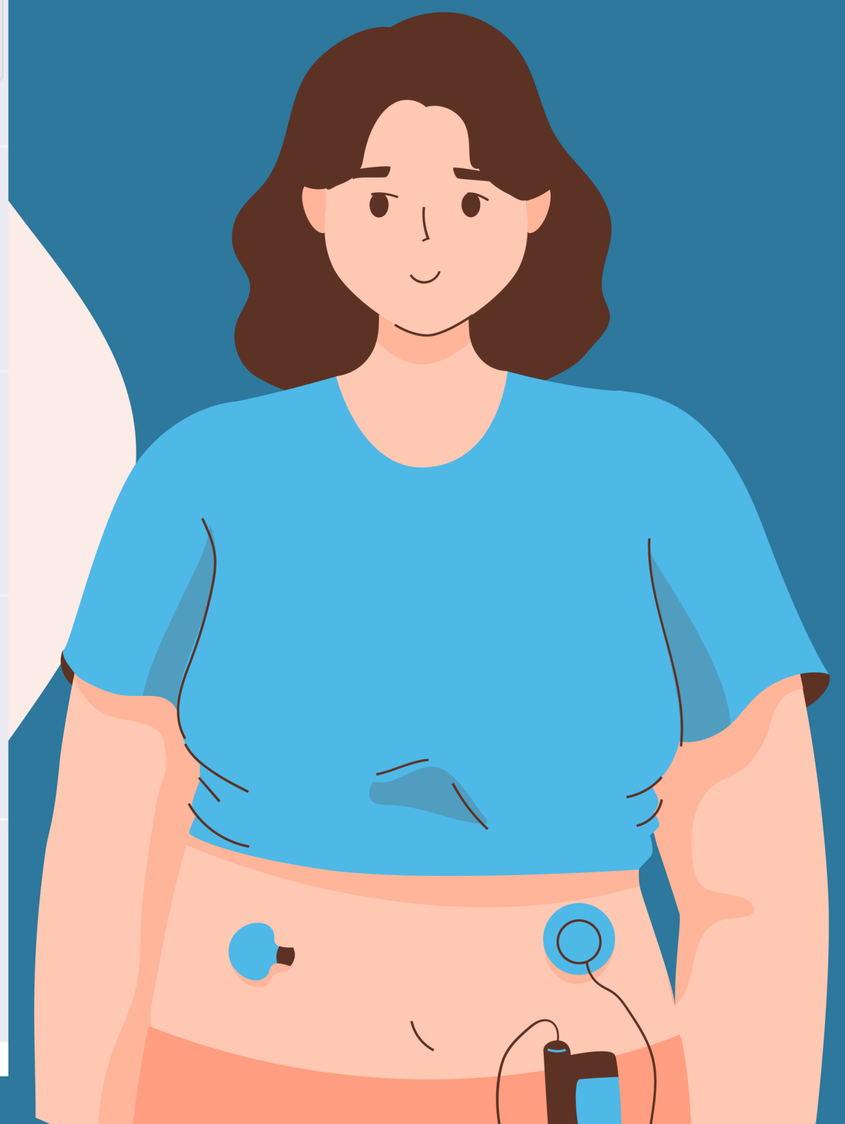
```
[ ]  rf_Grid = GridSearchCV(estimator = rfc, param_grid
                            = param_list)
```

LỰA CHỌN CHỈ SỐ

- F1 và Recall
- F1: cân bằng giữa precision và recall. Phù hợp cho dữ liệu mất cân bằng
- Recall: tỉ lệ các dữ liệu được phân loại đúng dựa trên tổng dữ liệu thực tế đúng
- Ảnh hưởng mật thiết đến việc chuẩn đoán và chữa trị bệnh

ĐÁNH GIÁ HIỆU SUẤT





KẾT LUẬN

	LGBM	Logistic Regression	Decision Tree	Gradient Boosting	Random Forest
Accuracy	0.95	0.93	0.95	0.95	0.96
Precision	0.5	0.42	0.49	0.48	0.55
F1-score	0.59	0.5	0.57	0.57	0.6
Recall	0.7	0.64	0.67	0.69	0.66

LIGHTGBM LÀ MÔ HÌNH ĐƯỢC CHỌN BỞI CÓ TỈ SỐ PRECISION VÀ F1 CAO NHẤT SO VỚI CÁC MODEL CÒN LẠI