Classification

```
# import essential libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline


# load the dataset
data = pd.read_csv('/content/diabetes_risk_prediction_dataset.csv')

# check the dimension of the dataset
print('Dimension of the dataset: ', data.shape)

# check the attributes in the dataset
print('Attributes in the dataset: ', data.columns.values)

# view the first 5 rows of the dataset
data.head()
```

```
Dimension of the dataset:  (520, 17)
Attributes in the dataset:  ['Age' 'Gender' 'Polyuria' 'Polydipsia' 'sudden weight loss' 'weakness'
 'Polyphagia' 'Genital thrush' 'visual blurring' 'Itching' 'Irritability'
 'delayed healing' 'partial paresis' 'muscle stiffness' 'Alopecia'
 'Obesity' 'class']
```

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing	partial paresis	mu stiff
0	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	
1	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	
2	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	
3	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	
4	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 520 entries, 0 to 519
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Age                 520 non-null    int64 
 1   Gender              520 non-null    object
 2   Polyuria            520 non-null    object
 3   Polydipsia          520 non-null    object
 4   sudden weight loss  520 non-null    object
 5   weakness            520 non-null    object
 6   Polyphagia          520 non-null    object
 7   Genital thrush      520 non-null    object
 8   visual blurring     520 non-null    object
 9   Itching             520 non-null    object
 10  Irritability        520 non-null    object
 11  delayed healing     520 non-null    object
 12  partial paresis     520 non-null    object
 13  muscle stiffness    520 non-null    object
 14  Alopecia            520 non-null    object
 15  Obesity             520 non-null    object
 16  class               520 non-null    object
dtypes: int64(1), object(16)
memory usage: 69.2+ KB
```

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2, random_state=122)

# dimension of train and test dataset
print('Dimension of training data: ', train.shape)
print('Dimension of test data: ', test.shape)
```

```
Dimension of training data:  (416, 17)
Dimension of test data:  (104, 17)
```

```
# segregate the feature matrix and target vector from train and test data
Xtrain = train.drop(columns=['class'], axis=1)
ytrain = train['class']

Xtest = test.drop(columns=['class'], axis=1)
ytest = test['class']


# encode the target/label for train and test dataset
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
ytrain_encoded = encoder.fit_transform(ytrain)
ytest_encoded = encoder.transform(ytest)


from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

# extracted the list of categorical columns to be encoded using OneHotEncoder
excluded_col = 'Age'
categorical_col = [col for col in Xtrain.columns if col != excluded_col]

# define the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(drop='first'), categorical_col)
    ],
    remainder='passthrough'
)

# create the pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('scaler', MinMaxScaler())
])

# process the train and test data
Xtrain_transformed = pipeline.fit_transform(Xtrain)
Xtest_transformed = pipeline.transform(Xtest)


## Classification

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# create a Decision Tree Classifier
tree = DecisionTreeClassifier(random_state=122)

# Define the hyperparameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4]
}

# create the GridSearchCV object
grid_search_tree = GridSearchCV(tree, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# fit the grid search to the data
grid_search_tree.fit(Xtrain_transformed, ytrain_encoded)

# print the best parameters and the corresponding accuracy
print('Best Parameters: ', grid_search_tree.best_params_)
print('Best Accuracy: ', grid_search_tree.best_score_)

# get the best model
best_tree = grid_search_tree.best_estimator_
```

```
Best Parameters:  {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
Best Accuracy:  0.935140562248996
```