BMI predictor

Using k-nearest neighbor (KNN)

Created by Evan Tan

Scope

1. Introduction to K-NN
2. Data set (raw)
3. Python code for K-NN
4. Data visualisation (conclusions)
5. Application of model

K-Nearest Neighbor

K- Nearest neighbor

The K-Nearest Neighbors (KNN) algorithm is a simple, non-parametric, supervised learning algorithm used for both classification and regression tasks.

It operates on the principle that similar items are close to each other in a feature space, meaning that the class or value of a new data point is predicted based on the classes or values of its nearest neighbors.

K- Nearest neighbor (characteristics)

Supervised Learning: KNN requires a **labeled dataset** for training.

Non-parametric: It **doesn't make assumptions** about the underlying data distribution.

Lazy Learning: The algorithm **doesn't have a dedicated training phase**. It uses all the **data for classification** during prediction.

Distance Metric: KNN uses a **distance metric** (Euclidean distance) to determine the proximity of data points.

K- Nearest neighbor (characteristics)

K Value: The 'k' in K-Nearest Neighbors represents the number of nearest neighbors to consider when making a prediction.

Classification: For classification, the algorithm typically uses **majority voting** among the 'k' nearest neighbors to determine the class label of a new data point.

Regression: For regression, the average of the values of the 'k' nearest neighbors is used to predict the value of a new data point.

Data set (raw)

Selection of data set

Key considerations: A good dataset is one that is accurate, complete, consistent, relevant, and timely, while also being easy to manipulate and understand.

Dataset used:
https://www.kaggle.com/datasets/yersever/500-person-gender-height-weight-body massindex/data

Raw data set

Gender : Male / Female **Since KNN uses numerical inputs, there is a need to process the ‘gender’.**

Height : Number (cm) / Weight : Number (Kg).

Index : (identified as the target variable).

0 - Extremely Weak

1 - Weak

2 - Normal

3 - Overweight

4 - Obesity

5 - Extreme Obesity

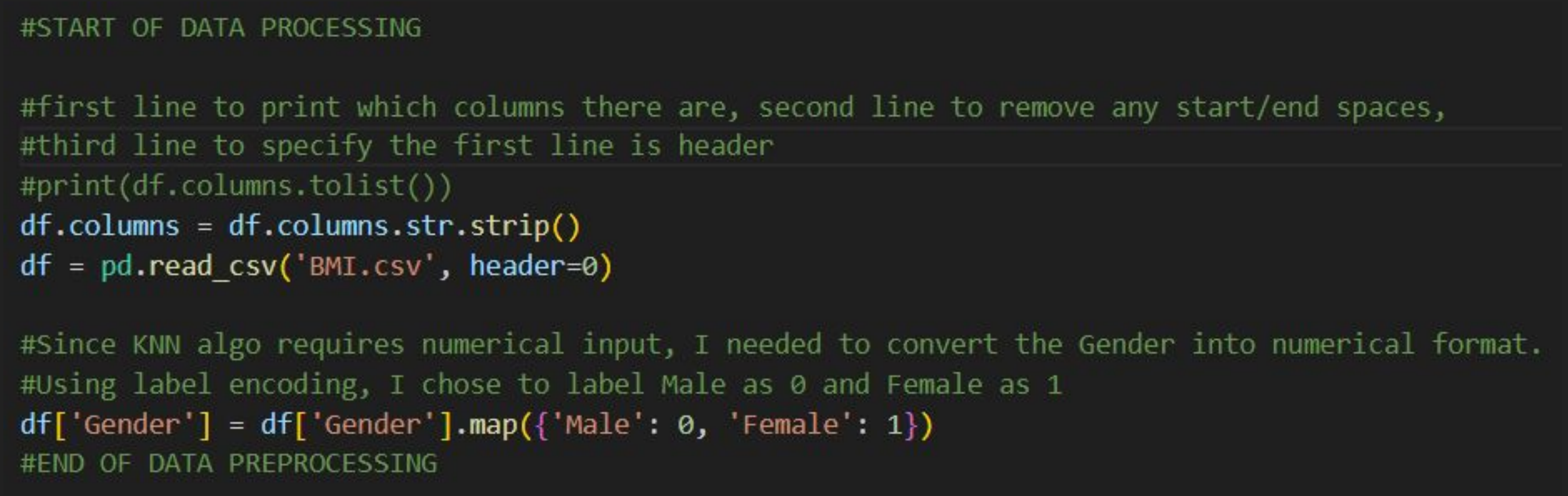
Code for model

Libraries

1. Pandas - for reading of csv files.
2. Sklearn - for usage of KNN algo.
3. Matplotlib - data visualisation.

Each import line brings in functions/classes needed later.

Data processing

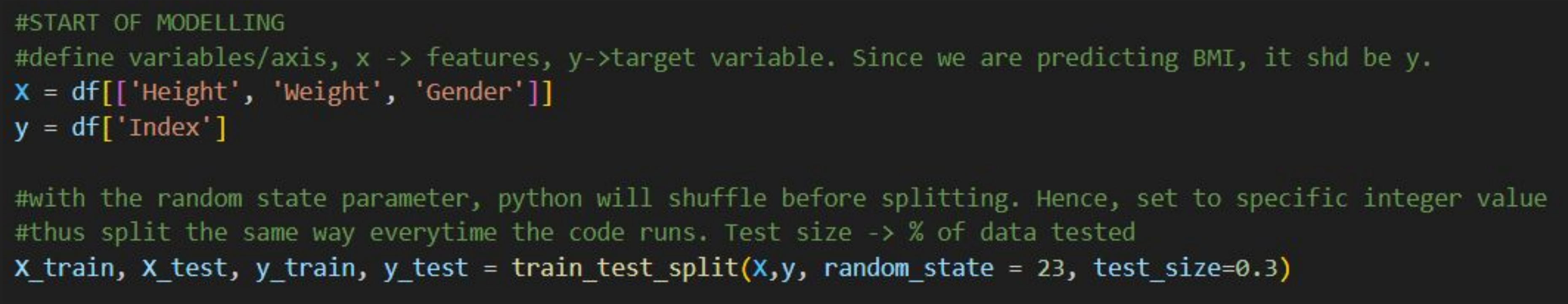


```
#START OF DATA PROCESSING

#first line to print which columns there are, second line to remove any start/end spaces,
#third line to specify the first line is header
#print(df.columns.tolist())
df.columns = df.columns.str.strip()
df = pd.read_csv('BMI.csv', header=0)

#Since KNN algo requires numerical input, I needed to convert the Gender into numerical format.
#Using label encoding, I chose to label Male as 0 and Female as 1
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
#END OF DATA PREPROCESSING
```

1.Remove any leading/trailing whitespace from all column names.

2.Replace ‘gender’ with numeric codes, because scikit-learn’s KNN requires numeric features.

Model’s code (1)



```
#START OF MODELLING
#define variables/axis, x -> features, y->target variable. Since we are predicting BMI, it shd be y.
X = df[['Height', 'Weight', 'Gender']]
y = df['Index']

#with the random state parameter, python will shuffle before splitting. Hence, set to specific integer value
#thus split the same way everytime the code runs. Test size -> % of data tested
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 23, test_size=0.3)
```

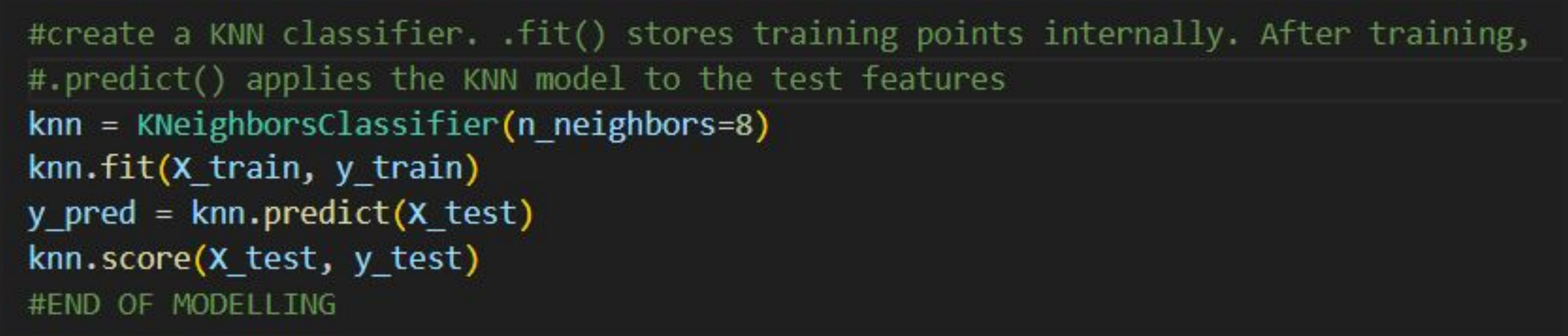
Here X is a Data Frame of shape (n_samples, 3) containing height, weight, and gender; y is a Series of the BMI classification. This selects only the relevant columns for modeling.

We need to create a Min-Max scaler and apply it to X. This rescales each feature to the range [0,1], which is important for distance-based algorithms like KNN.

The test_size=0.2 means 20% of rows go to test.

The random_state= fixed value of ‘n’ ensures the split is reproducible.

Model’s code (2)

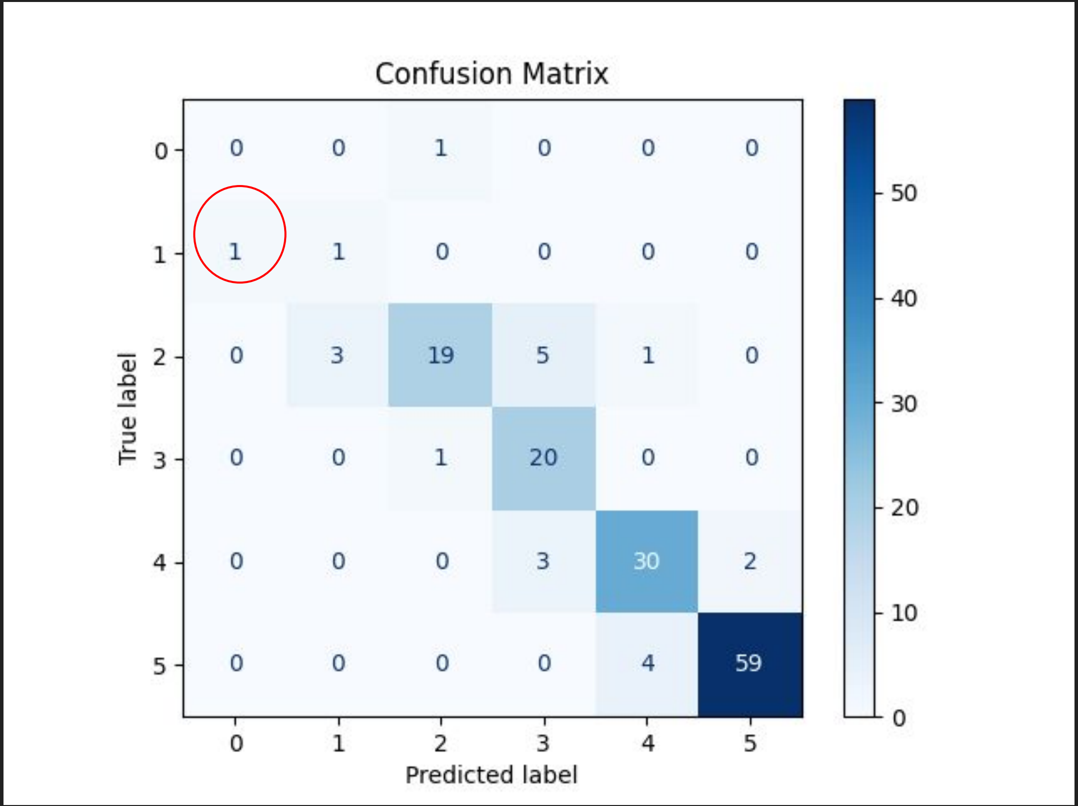


```
#create a KNN classifier. .fit() stores training points internally. After training,
#.predict() applies the KNN model to the test features
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
knn.score(X_test, y_test)
#END OF MODELLING
```

KNN classifier with k=n neighbors. In this scenario, n = 8.

After training, y_pred = knn.predict(X_test) applies the learned KNN model to the test features. This finds, for each test point, the majority class among its k closest points in the training set and returns those predictions.

Data visualisation



	0	1	2	3	4	5
0	0	0	1	0	0	0
1	1	1	0	0	0	0
2	0	3	19	5	1	0
3	0	0	1	20	0	0
4	0	0	0	3	30	2
5	0	0	0	0	4	59

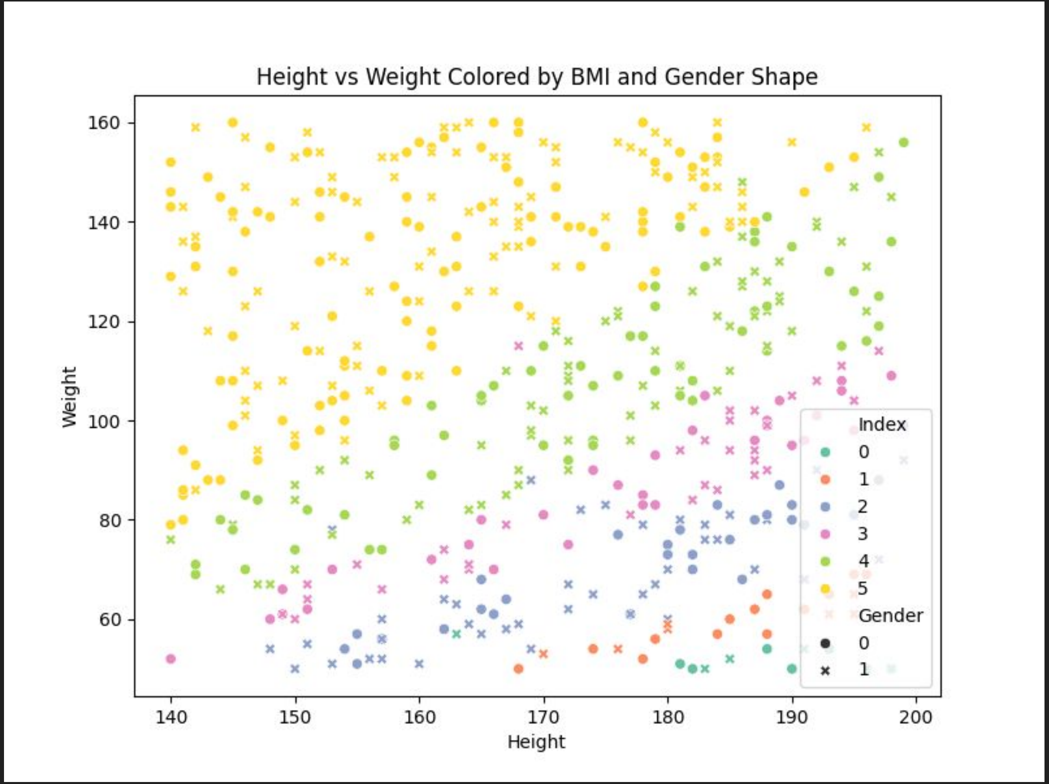
*misclassification of 1 as 0

The model does really well on classes 3, 4, and 5.

It struggles with class 0 and class 1, likely because:

1. There are very few samples

2. Not enough training examples to learn from



From the graph, some conclusions that can be made are:

1.High BMI

These individuals have high weight for their height. Circles and crosses are both present, so it's not gender-specific.

2. Gender Shapes are Fairly Evenly Distributed

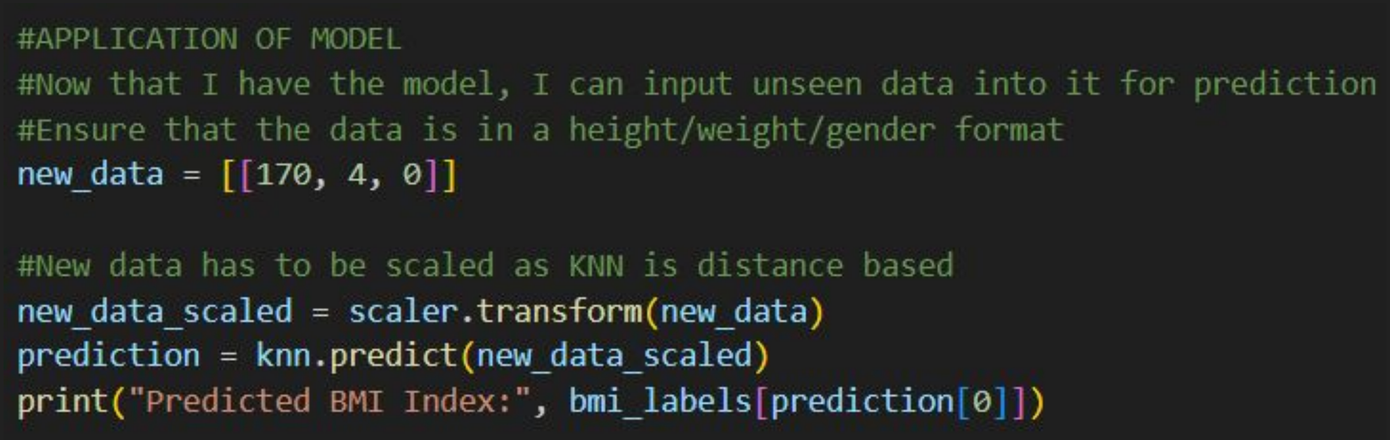
Suggests no dramatic gender imbalance in the dataset or BMI categories, though a deeper look may reveal small difference

Scatter plot (2D)

Created by Evan Tan

Application of Model

Prediction of new data



```
#APPLICATION OF MODEL
#Now that I have the model, I can input unseen data into it for prediction
#Ensure that the data is in a height/weight/gender format
new_data = [[170, 4, 0]]

#New data has to be scaled as KNN is distance based
new_data_scaled = scaler.transform(new_data)
prediction = knn.predict(new_data_scaled)
print("Predicted BMI Index:", bmi_labels[prediction[0]])
```

New data can be run through the model.