# K-Nearest-Neighbors T-Shirt Size Classification Project


## Nawal Obaid


### 9/6/2022

# PROBLEM STATEMENT

You own an online clothing business and you would like to develop a new app (or in-store) feature in which customers would enter their own height and weight and the system would predict what T-shirt size should they wear. Features are height and weight and output is either L (Large) or S (Small).

In [179…

```
# import image module
from IPython.display import Image
# get the image
Image(url="shirt.jpg", width=800, height=300)
```

Out[179]:

### Imoprt the needed libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

### Load the dataset

In [2]:

```
shirt_df = pd.read_csv('Tshirt_Sizing_Dataset.csv')
```

In [3]:

```
shirt_df.head()
```

Out[3]:

| | Height (in cms) | Weight (in kgs) | T Shirt Size |
|---|---|---|---|
| **0** | 158 | 58 | S |
| **1** | 158 | 59 | S |
| **2** | 158 | 63 | S |
| **3** | 160 | 59 | S |
| **4** | 160 | 60 | S |

In [4]:

```
shirt_df.info()
# 18 entries, 0 to 17
# columns (total 3 columns):
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
```

```
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Height (in cms)  18 non-null     int64 
 1   Weight (in kgs)  18 non-null     int64 
 2   T Shirt Size     18 non-null     object
dtypes: int64(2), object(1)
memory usage: 560.0+ bytes
```

In [5]:
```python
shirt_df.describe()
```

Out[5]:

| | Height (in cms) | Weight (in kgs) |
|---|---|---|
| **count** | 18.00000 | 18.000000 |
| **mean** | 164.00000 | 62.333333 |
| **std** | 4.32503 | 2.634611 |
| **min** | 158.00000 | 58.000000 |
| **25%** | 160.00000 | 60.250000 |
| **50%** | 164.00000 | 62.500000 |
| **75%** | 168.00000 | 64.000000 |
| **max** | 170.00000 | 68.000000 |

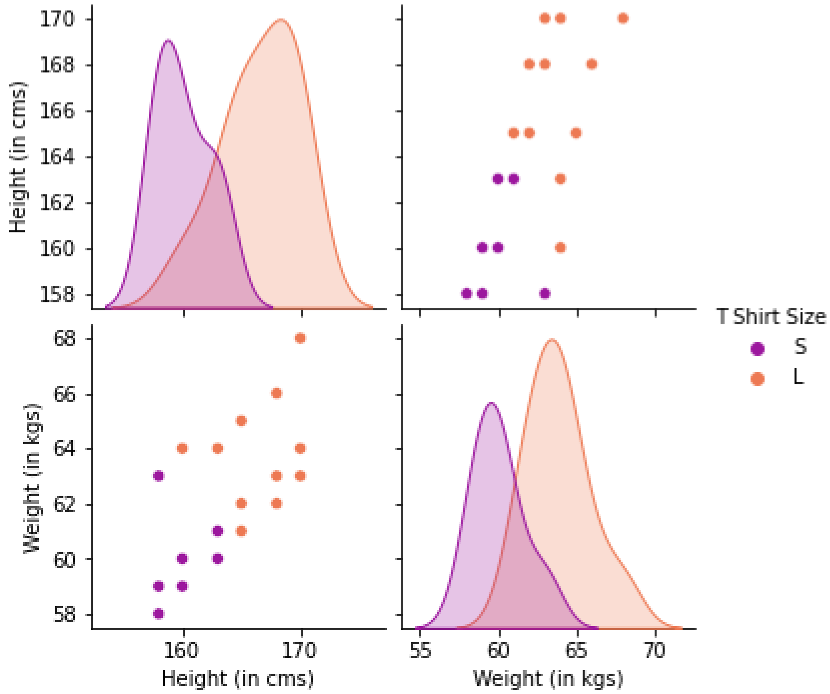
## EDA

In [6]:
```python
sns.heatmap(shirt_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
# No missing values
# No data imputation is needed
```

Out[6]:
```
<AxesSubplot:>
```

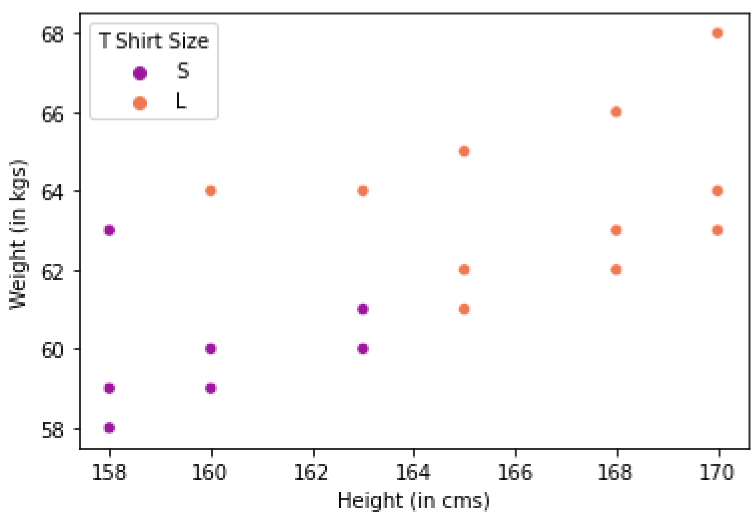

In [7]:
```python
sns.pairplot(data = shirt_df, hue = 'T Shirt Size', palette = 'plasma')
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x1f184921310>
```




```
In [8]: sns.scatterplot(data = shirt_df, x = 'Height (in cms)', y = 'Weight (in kgs)' , hue = 'T
```

```
Out[8]: <AxesSubplot:xlabel='Height (in cms)', ylabel='Weight (in kgs)'>
```




# Train Test Split

```
In [143…  X = shirt_df.drop('T Shirt Size', axis = 1).values
          y = shirt_df['T Shirt Size'].values
```

```
In [144…  X.shape
```

```
Out[144]: (18, 2)
```

## Label Encoding

```
In [145…  # we need labelencoder for the y values to convert te values from L, S to 1, 0
          y
```

```
Out[145]: array(['S', 'S', 'S', 'S', 'S', 'S', 'S', 'L', 'L', 'L', 'L', 'L', 'L',
                 'L', 'L', 'L', 'L', 'L'], dtype=object)
```

In [146…
```
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
```

In [147…
```
y
```

Out[147]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [148…
```
# will do a standraization for the variables!
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [149…
```
from sklearn.model_selection import train_test_split
```

In [151…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state =
```

# Fit/train the model

In [164…
```
#p : int, default=2 , Power parameter for the Minkowski metric.
#metric : str or callable, default='minkowski'. The default metric is minkowski
#, and with p=2 is equivalent to the standard Euclidean metric.
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=2, metric = 'minkowski', p=2)
```

In [165…
```
classifier.fit(X_train, y_train)
```

Out[165]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=2)
```

In [166…
```
predictions = classifier.predict(X_test)
predictions
```

Out[166]:
```
array([0, 1, 0, 0, 0])
```

# Evaluation for this model

In [167…
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [168…
```
print('*************Classification Report*************************')
print(classification_report(y_test, predictions))
print('***************Confusion Matrix***************************')
print(confusion_matrix(y_test, predictions))
```

```
*************Classification Report*************************
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         2
           1       1.00      0.33      0.50         3

    accuracy                           0.60         5
   macro avg       0.75      0.67      0.58         5
weighted avg       0.80      0.60      0.57         5

***************Confusion Matrix***************************
```

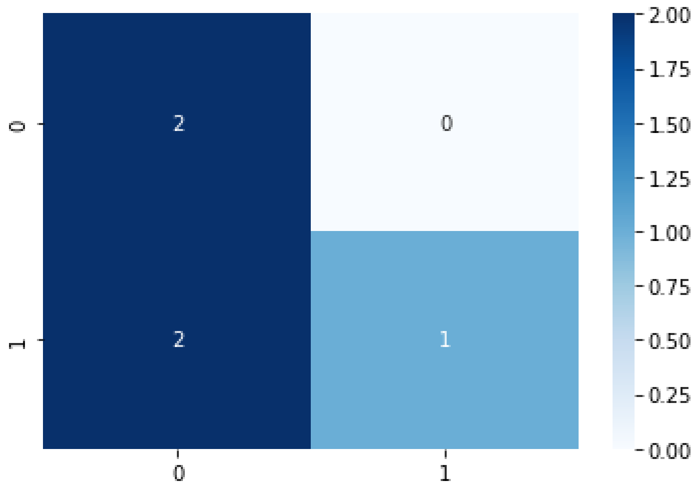
```
[[2 0]
 [2 1]]
```

In [169…

```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot = True, fmt = "d", cmap ='Blues')
```

Out[169]:

```
<AxesSubplot:>
```




# Improving the model by choosing a good K value

Let's go ahead and use the elbow method to pick a good K Value:

## Elbow Method

In [170…

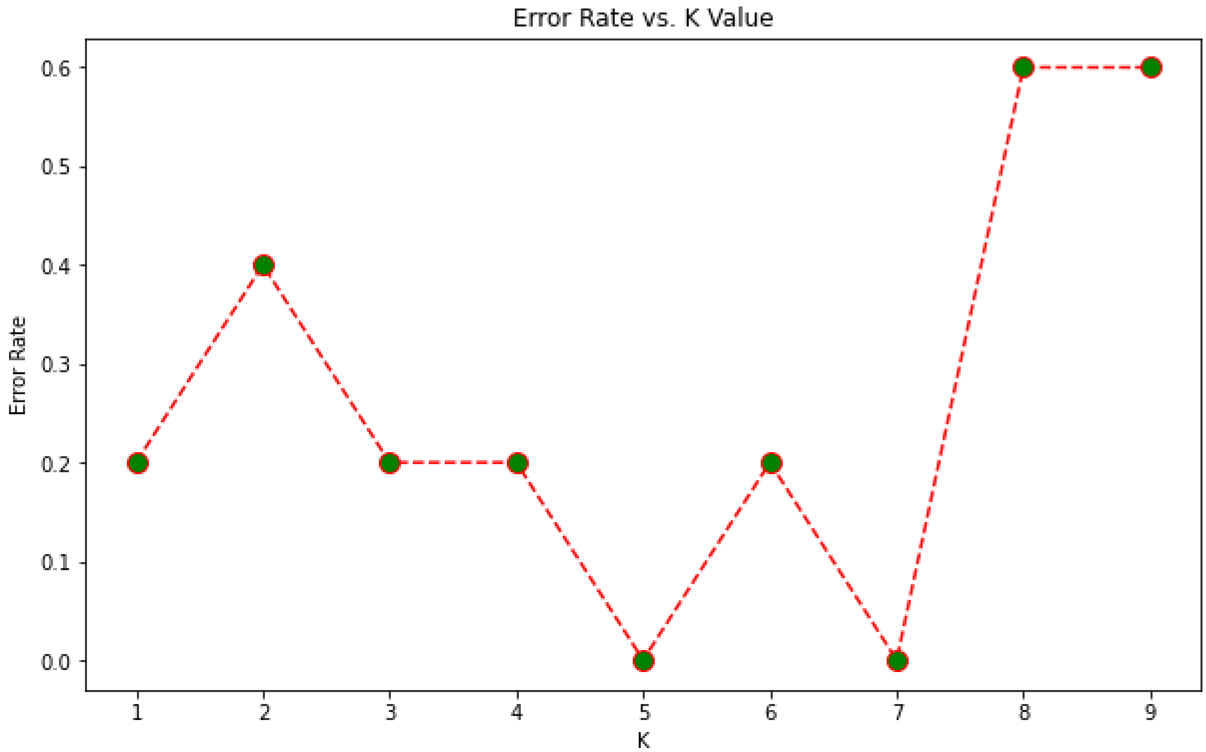
```
error_rate = []

# Will take some time
for i in range(1,10):

    classifier = KNeighborsClassifier(n_neighbors=i)
    classifier.fit(X_train,y_train)
    pred_i = classifier.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [171…

```
plt.figure(figsize=(10,6))
plt.plot(range(1,10),error_rate,color='red', linestyle='dashed', marker='o',
         markerfacecolor='green', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[171]:

```
Text(0, 0.5, 'Error Rate')
```




Here we can see that that at K= 5 & K=7 the error rate just tends to hover around 0.16-0.17 Let's retrain the model with that and check the classification report!

In [172…

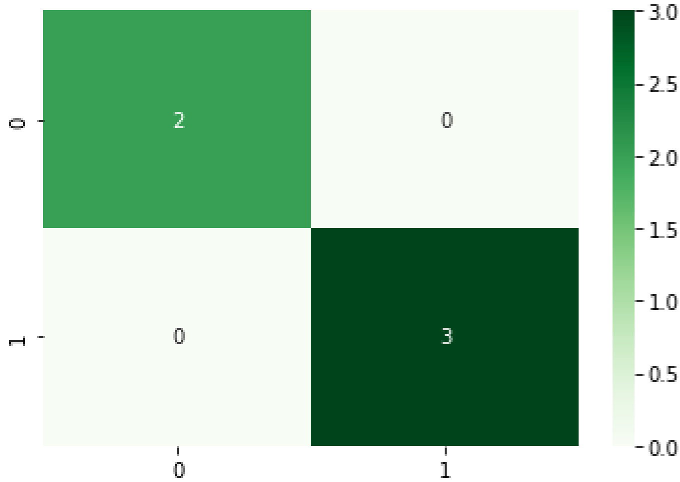
```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=5
classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(X_train,y_train)
predictions = classifier.predict(X_test)

print('WITH K=5')
print('\n')
print('*************Classification Report************************')
print(classification_report(y_test, predictions))
print('***************Confusion Matrix***************************')
print(confusion_matrix(y_test, predictions))
```

```
WITH K=5


*************Classification Report************************
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         3

    accuracy                           1.00         5
   macro avg       1.00      1.00      1.00         5
weighted avg       1.00      1.00      1.00         5

***************Confusion Matrix***************************
[[2 0]
 [0 3]]
```

In [173…

```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot = True, fmt = "d", cmap ='Greens')
```

Out[173]: <AxesSubplot:>


2
0
0
3
0
1
0
1
3.0
2.5
2.0
1.5
1.0
0.5
0.0