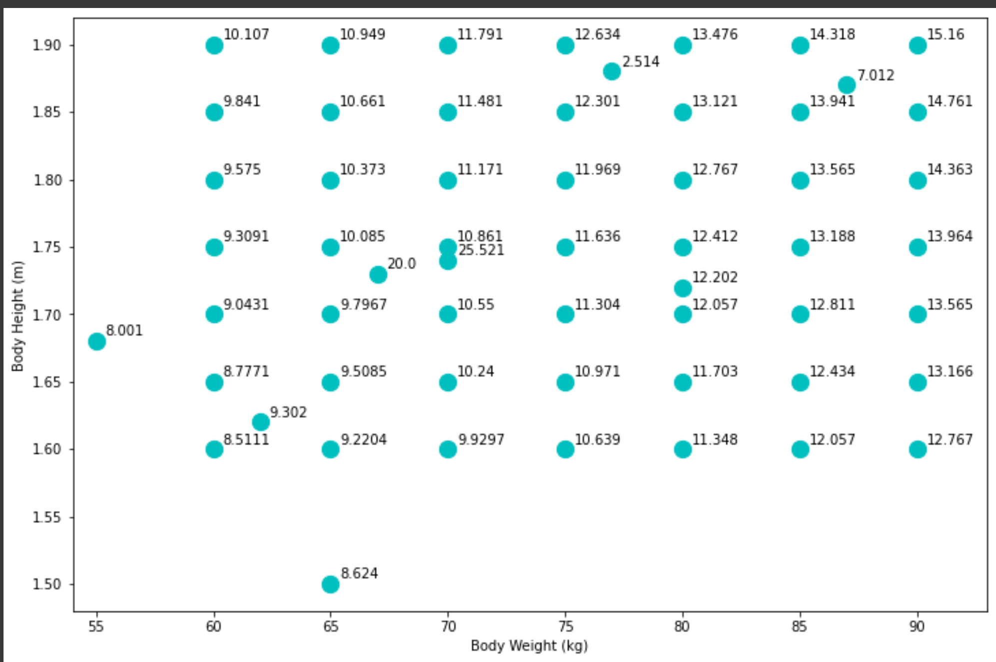
```
# Importing the library files...
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Fetching the Dataset for training...
path = "https://raw.githubusercontent.com/shakil1819/CSE442-Machine-Learning-Sessional/main/Week%201%2B2/Knee-Torque-ZDataSet.
df = pd.read_csv(path)
df.tail()
```

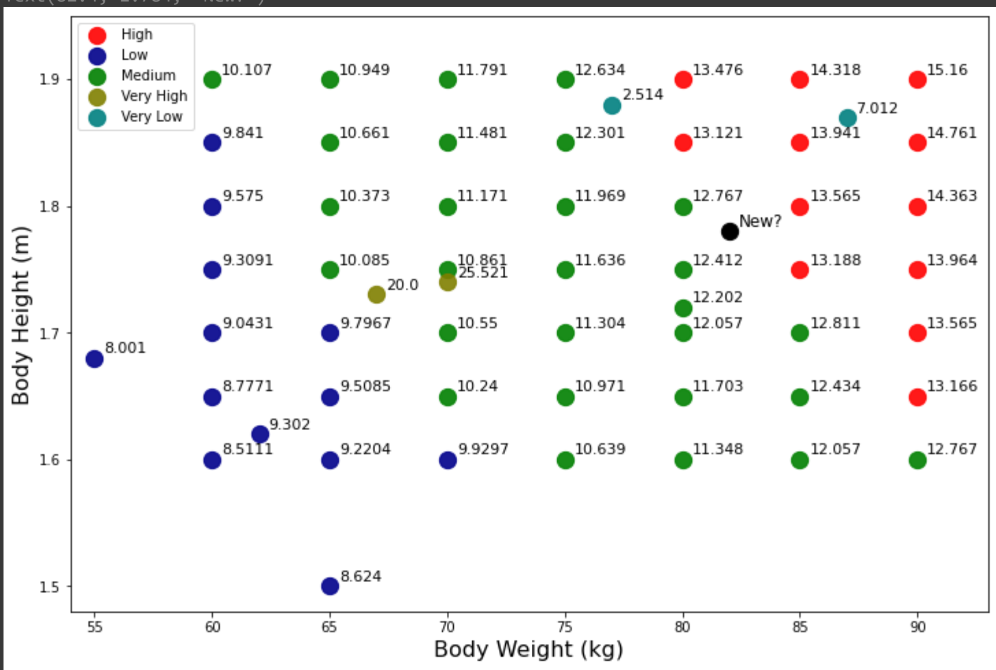
	Body Weight (kg)	Body Height (m)	Weight-Height Ratio (R)	Internal Moment (M)	Required Torque (N-m)	Torque Category
52	62	1.62	NaN	NaN	9.302	Low
53	65	1.50	NaN	NaN	8.624	Low
54	87	1.87	NaN	NaN	7.012	Very Low
55	77	1.88	NaN	NaN	2.514	Very Low
56	67	1.73	NaN	NaN	20.000	Very High

```
# Plotting the Body Weight-Height graph...
# Help link: https://matplotlib.org/stable/tutorials/introductory/pyplot.html
#            https://www.w3schools.com/python/matplotlib_plotting.asp
# https://www.python-graph-gallery.com/custom-legend-with-matplotlib
plt.figure(figsize=(12, 8))
plt.scatter(df["Body Weight (kg)"], df["Body Height (m)"], s=150, c='c')
plt.axis([54, 93, 1.48, 1.92]) # plt.axis([x-min, x-max, y-min, y-max])

plt.xlabel("Body Weight (kg)")
plt.ylabel("Body Height (m)")

w = df["Body Weight (kg)"].to_numpy()
h = df["Body Height (m)"].to_numpy()
i=0
# df.to_numpy() to convert the DataFrame to a NumPy array...
for v in df["Required Torque (N-m)"].to_numpy(): #df.iloc[:, 4].values:
  #plt.text(df["Body Weight (kg)"], df["Body Height (m)"], "1")
  plt.text(w[i]+.4, h[i]+.004, v)
  i=i+1

```



```
T_CATEGORY = df['Torque Category'].values
T_CATEGORY_ = np.unique(T_CATEGORY)
COLORS = ["#FF0000", "#00008b", "#008000", "#808000", "#008080"]

fig, ax = plt.subplots(figsize=(12,8))
plt.axis([54, 93, 1.48, 1.95]) # plt.axis([x-min, x-max, y-min, y-max])
for catagory, color in zip(T_CATEGORY_, COLORS):
    idxs = np.where(T_CATEGORY == catagory)
    # No legend will be generated if we do not pass label=catagory...
    ax.scatter(
        w[idxs], h[idxs], label=catagory, s=150, color=color, alpha=0.9
    )
ax.legend()

font = {'family': 'sans serif', 'color':  'black',
        'weight': 'normal', 'style' : 'normal', 'size': 11,
        }
i=0
for v in df["Required Torque (N-m)"].to_numpy(): #df.iloc[:, 4].values:
  plt.text(w[i]+.4, h[i]+.004, v, fontdict=font)
  i=i+1

# https://matplotlib.org/stable/tutorials/text/text_props.html
font = {'family': 'sans serif', 'color':  'black',
        'weight': 'normal', 'style' : 'normal', 'size': 16,
        }
ax.set_xlabel("Body Weight (kg)", fontdict=font, fontsize=16)
ax.set_ylabel("Body Height (m)", fontdict=font, fontsize=16)

plt.scatter(82, 1.78, s=150, c='#000000')
plt.text(82+.4, 1.78+.004, "New?", fontname='sans serif', fontsize=12)
```

```
Text(82.4, 1.784, 'New?')
```




```
# df # df will sohow all the records with values...
# df.isnull() # This will soow all the records with Ture or False (bool) values...
df.isnull().sum() # [df.isna().sum()] Shows the features with NULL values count...
```

```
Body Weight (kg)           0
Body Height (m)            0
Weight-Height Ratio (R)    8
Internal Moment (M)        8
Required Torque (N-m)      0
Torque Category            0
dtype: int64
```

```
# ===================================
# Dropping the NULL records...
# ===================================
# df1 = df
# avg1 = df['Weight-Height Ratio (R)'].sum()/df['Weight-Height Ratio (R)'].count()
# avg2 = df['Internal Moment (M)'].sum()/df['Internal Moment (M)'].count()

#### df.dropna() or df.fillna(100) or df.replace(np.nan, 0) or df.interpolate() ...
# df1['Weight-Height Ratio (R)'] = df['Weight-Height Ratio (R)'].replace(np.nan, avg1)
# df1['Internal Moment (M)'] = df['Internal Moment (M)'].replace(np.nan, avg2)

df1 = df.dropna()

df1.tail(10)
```

	Body Weight (kg)	Body Height (m)	Weight-Height Ratio (R)	Internal Moment (M)	Required Torque (N-m)	Torque Category
39	85	1.80	47.222	1.3841	13.565	High
40	85	1.85	45.946	1.4226	13.941	High
41	85	1.90	44.737	1.4610	14.318	High
42	90	1.60	56.250	1.3027	12.767	Medium
43	90	1.65	54.545	1.3434	13.166	High
44	90	1.70	52.941	1.3841	13.565	High
45	90	1.75	51.429	1.4249	13.964	High
46	90	1.80	50.000	1.4656	14.363	High
47	90	1.85	48.649	1.5063	14.761	High
48	90	1.90	47.368	1.5470	15.160	High

```
y = df1['Required Torque (N-m)']
X = df1.drop(['Weight-Height Ratio (R)', 'Internal Moment (M)',
              'Required Torque (N-m)', 'Torque Category'], axis=1)
X.describe()
```

	Body Weight (kg)	Body Height (m)
count	49.00000	49.000000
mean	75.00000	1.750000
std	10.10363	0.101036
min	60.00000	1.600000
25%	65.00000	1.650000
50%	75.00000	1.750000
75%	85.00000	1.850000
max	90.00000	1.900000

```
# Splitting Train and Test data (75% & 25%, respectively) ...
from sklearn.model_selection import train_test_split
# SEED = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

len(X)       # 57
len(X_train) # 42
len(X_test)  # 15

X_test
```

	Body Weight (kg)	Body Height (m)
30	80	1.70
5	60	1.85
43	90	1.65
21	75	1.60

```
X_test2 = X_test[:1]
X_test2
```

	Body Weight (kg)	Body Height (m)
30	80	1.7
18	70	1.80

```
# Feature Scaling for KNN Regression...
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit only on X_train...
scaler.fit(X_train)

X_t = X_test
X_t2 = X_test2

# Scale both X_train and X_test...
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_test2 = scaler.transform(X_test2)
```

```
X_test2
```

```
array([[ 0.54403307, -0.66422963]])
```

```
col_names=['Body Weight (kg)', 'Body Height (m)'] # , 'Weight-Height Ratio (R)', 'Internal Moment (M)']
scaled_df = pd.DataFrame(X_train, columns=col_names)
scaled_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Body Weight (kg)	36.0	4.502571e-16	1.014185	-1.464704	-0.96252	0.041849	1.046217	1.548402
Body Height (m)	36.0	2.800229e-15	1.014185	-1.703893	-0.66423	-0.144398	0.895266	1.415098

```
# Training and Predicting K-NN Regression...
from sklearn.neighbors import KNeighborsRegressor
K = 5
regressor = KNeighborsRegressor(n_neighbors=K)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
print(X_test), print(y_test), print(y_pred)
```

```
array([[ 0.54403307, -0.66422963],
       [-1.46470441,  0.89526602],
       [ 1.54840181, -1.18406151],
       [ 0.0418487 , -1.7038934 ],
       [-1.46470441,  1.41509791],
       [ 1.04621744, -1.7038934 ],
       [-0.46033567, -0.66422963],
       [ 0.54403307, -1.18406151],
       [-0.46033567,  0.37543414],
       [-0.96252004,  0.89526602],
       [ 1.54840181, -0.66422963],
       [ 1.04621744, -1.18406151],
       [ 0.54403307, -1.7038934 ]])
```

```
y_pred2 = regressor.predict(X_test2)
y_pred2
```

```
array([12.2702])
```

```
col_names=['Required Torque (N-m)']
scaled_y_pred = pd.DataFrame(y_pred2, columns=col_names)
scaled_y_pred
```

Required Torque (N-m)

```
X_t2
```

	Body Weight (kg)	Body Height (m)
30	80	1.7

```
#col_names=['Body Weight (kg)', 'Body Height (m)', 'Weight-Height Ratio (R)', 'Internal Moment (M)']
pred_df = X_t2 # pd.DataFrame(X_test2, columns=col_names)
pred_df
```

	Body Weight (kg)	Body Height (m)
30	80	1.7

```
pred_df['Required Torque (N-m)'] = y_pred2
pred_df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

	Body Weight (kg)	Body Height (m)	Required Torque (N-m)
30	80	1.7	12.2702

```
# Evaluating the Algorithm for KNN Regression...
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f'mae: {mae}')
print(f'mse: {mse}')
print(f'rmse: {rmse}')
```

```
mae: 0.2714353846153841
mse: 0.10830680212307671
rmse: 0.32909998803262924
```

```
# R2 can be calculated...
regressor.score(X_test, y_test)
```

```
0.9105616219325816
```

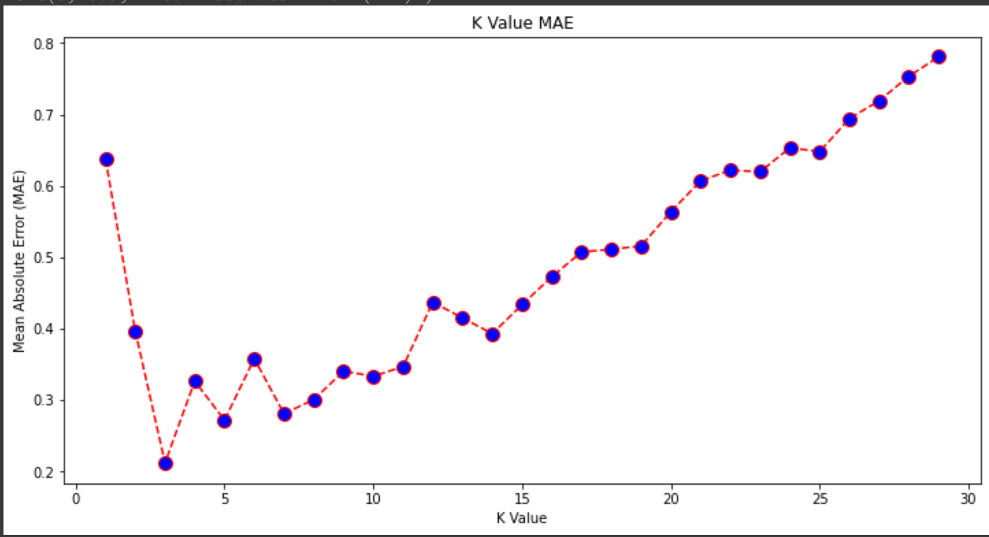
```
y.describe()
```

```
count    49.000000
mean     11.636300
std       1.707826
min       8.511100
25%      10.240000
50%      11.636000
75%      12.811000
max      15.160000
Name: Required Torque (N-m), dtype: float64
```

```
# ========================================
# Finding the Best K for KNN Regression...
# ========================================
error = []

# Calculating MAE error for K values between 1 and 29
for i in range(1, 30):
    knn = KNeighborsRegressor(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    mae = mean_absolute_error(y_test, pred_i)
    error.append(mae)
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), error, color='red',
         linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)

plt.title('K Value MAE')
plt.xlabel('K Value')
plt.ylabel('Mean Absolute Error (MAE)')
```

```
Text(0, 0.5, 'Mean Absolute Error (MAE)')
```




```python
# ===================================
# Predicting with minimum error position as K...
# ===================================
import numpy as np

# print(min(error))
# print(np.array(error).argmin())

print(min(error[2:]))          # 0.14875230769230813
print(np.array(error[2:]).argmin()+2) # 4
K_neigh = np.array(error[2:]).argmin()+2

knn_reg = KNeighborsRegressor(n_neighbors=4)
knn_reg.fit(X_train, y_train)
y_pred_reg = knn_reg.predict(X_test)
r2_reg = knn_reg.score(X_test, y_test)

mae_reg = mean_absolute_error(y_test, y_pred_reg)
mse_reg = mean_squared_error(y_test, y_pred_reg)
rmse_reg = mean_squared_error(y_test, y_pred_reg, squared=False)
print(f'r2: {r2_reg}, \nmae: {mae_reg} \nmse: {mse_reg} \nrmse: {rmse_reg}')
```

```
0.21149487179487172
2
r2: 0.8485152716705489,
mae: 0.3261134615384617
mse: 0.18344279995192306
rmse: 0.42830222968357645
```

```python
# ==============================================
# Classification using K-Nearest Neighbors with Scikit-Learn...
# ==============================================

dfc = df.dropna()

# Creating 4 categories and assigning them to a NewColumn...
# dfc["NewColumn"] = pd.qcut(dfc["ClassificatinColumn"], 4, retbins=False,
#                labels=[1, 2, 3, 4])
y = dfc['Torque Category']
X = dfc.drop(['Weight-Height Ratio (R)', 'Internal Moment (M)',
              'Required Torque (N-m)', 'Torque Category'], axis = 1)

```

```python
# Splitting Data into Train and Test Sets...
from sklearn.model_selection import train_test_split

# SEED = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25) #, random_state=SEED)
```

```python
# Feature Scaling for Classification...
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# Training and Predicting for Classification...
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier() # n_neighbors = 5
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
y_pred
```

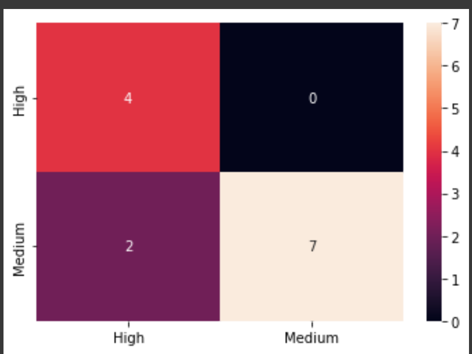
```
array(['High', 'High', 'High', 'Medium', 'Medium', 'Medium', 'Medium',
       'Medium', 'High', 'High', 'High', 'Medium', 'Medium'], dtype=object)
```

```python
# Evaluating KNN for Classification...
acc =  classifier.score(X_test, y_test)
print(acc) # 0.8461538461538461
```

```
0.8461538461538461
```

```python
# Visualize using a heatmap....
from sklearn.metrics import classification_report, confusion_matrix
#importing Seaborn's to use the heatmap
import seaborn as sns

# Adding classes names for better interpretation
classes_names = ['High','Medium']
cm = pd.DataFrame(confusion_matrix(y_test, y_pred),
                  columns=classes_names, index = classes_names)

# Seaborn's heatmap to better visualize the confusion matrix
sns.heatmap(cm, annot=True, fmt='d');

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        High       0.67      1.00      0.80         4
      Medium       1.00      0.78      0.88         9

    accuracy                           0.85        13
   macro avg       0.83      0.89      0.84        13
weighted avg       0.90      0.85      0.85        13
```

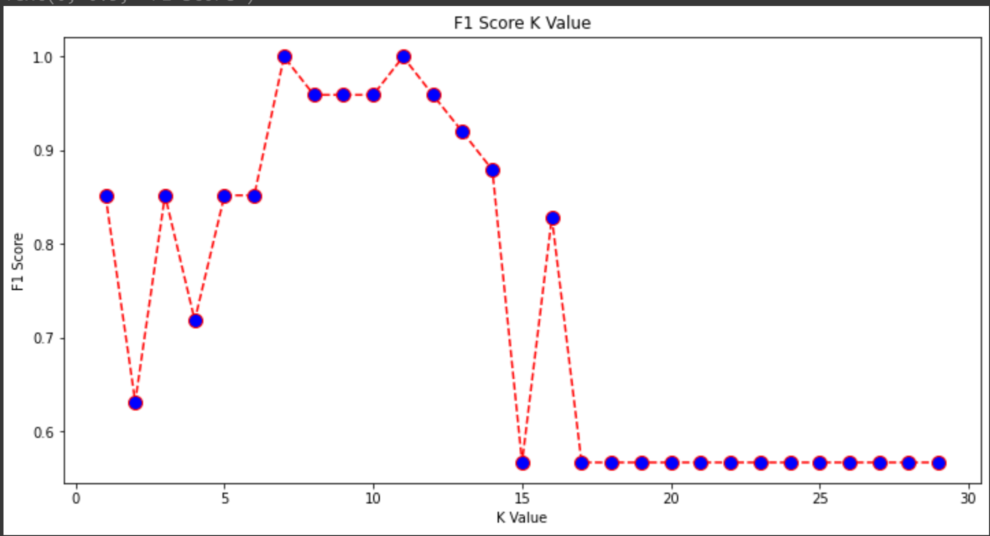



```python
print(f'Test: {np.array(y_test)} \n')
print(f'Pred: {y_pred}')
```

```
Test: ['High' 'Medium' 'Medium' 'Medium' 'Medium' 'Medium' 'Medium' 'Medium'
 'High' 'High' 'High' 'Medium' 'Medium']

Pred: ['High' 'High' 'High' 'Medium' 'Medium' 'Medium' 'Medium' 'Medium' 'High'
 'High' 'High' 'Medium' 'Medium']
```

```python
# =============================================
# Finding the Best K for KNN Classification...
# =============================================
from sklearn.metrics import f1_score

f1s = []

# Calculating f1 score for K values between 1 and 30
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    # average='weighted' to calculate a weighted average for the classes
    f1s.append(f1_score(y_test, pred_i, average='weighted'))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), f1s, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('F1 Score K Value')
plt.xlabel('K Value')
plt.ylabel('F1 Score')
```

```
Text(0, 0.5, 'F1 Score')
```




```python
np.array(f1s)
```

```
array([0.85192308, 0.63116371, 0.85192308, 0.71828172, 0.85192308,
       0.85192308, 1.        , 0.95927602, 0.95927602, 0.95927602,
       1.        , 0.95927602, 0.91960671, 0.87912088, 0.56643357,
       0.82820513, 0.56643357, 0.56643357, 0.56643357, 0.56643357,
       0.56643357, 0.56643357, 0.56643357, 0.56643357, 0.56643357,
       0.56643357, 0.56643357, 0.56643357, 0.56643357])
```

```python
# The f1-score is the highest when the value of the K is 7 or 11...
# Retrain the classifier with 7 neighbors...
classifier7 = KNeighborsClassifier(n_neighbors=7)
classifier7.fit(X_train, y_train)
y_pred7 = classifier7.predict(X_test)
print(classification_report(y_test, y_pred7))
```

```
              precision    recall  f1-score   support

        High       1.00      1.00      1.00         4
      Medium       1.00      1.00      1.00         9

    accuracy                           1.00        13
   macro avg       1.00      1.00      1.00        13
weighted avg       1.00      1.00      1.00        13
```

```python
# Retrain the classifier with 11 neighbors...
classifier11 = KNeighborsClassifier(n_neighbors=11)
classifier11.fit(X_train, y_train)
y_pred11 = classifier11.predict(X_test)
print(classification_report(y_test, y_pred11))
```

```
              precision    recall  f1-score   support

        High       1.00      1.00      1.00         4
      Medium       1.00      1.00      1.00         9

    accuracy                           1.00        13
```

```
    macro avg       1.00      1.00      1.00        13
 weighted avg       1.00      1.00      1.00        13
```

```python
# Retrain the classifier with 5 neighbors...
classifier5 = KNeighborsClassifier(n_neighbors=5)
classifier5.fit(X_train, y_train)
y_pred5 = classifier5.predict(X_test)
print(classification_report(y_test, y_pred5))
```

```
              precision    recall  f1-score   support

        High       0.67      1.00      0.80         4
      Medium       1.00      0.78      0.88         9

    accuracy                           0.85        13
   macro avg       0.83      0.89      0.84        13
weighted avg       0.90      0.85      0.85        13
```

```python
# ============================================================
# Implementing KNN for Outlier Detection with Scikit-Learn...
# ============================================================
from sklearn.neighbors import NearestNeighbors

nbrs = NearestNeighbors(n_neighbors = 5)
nbrs.fit(X_train)
# Distances and indexes of the 5 neighbors...
distances, indexes = nbrs.kneighbors(X_train)

# 5 distances for each data point (distance between itself and 5 neighbors)...
distances[:3], distances.shape
```

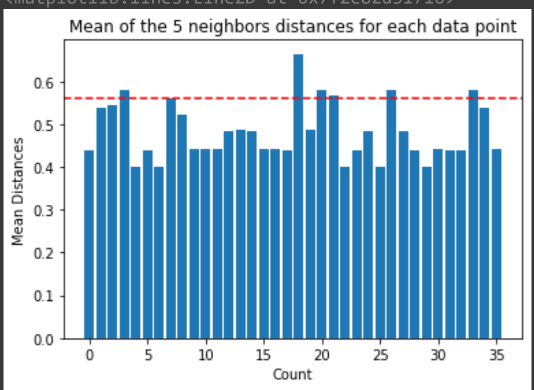
```
(array([[0.        , 0.48989795, 0.48989795, 0.51034318, 0.70742502],
        [0.        , 0.48989795, 0.51034318, 0.70742502, 0.9797959 ],
        [0.        , 0.48989795, 0.51034318, 0.70742502, 1.02068636]]),
 (36, 5))
```

```python
# Look at the neighbors' indexes for 3 rows...
indexes[:3], indexes[:3].shape
```

```
(array([[ 0, 23, 27,  4, 25],
        [ 1, 23, 10, 25,  0],
        [ 2, 24, 11,  6, 30]]), (3, 5))
```

```python
# calculate the mean of the 5 distances and plot a graph that counts
# each row on the X-axis and displays each mean distance on the Y-axis...
dist_means = distances.mean(axis=1)
plt.bar(np.array(range(0, 36)), dist_means)
plt.title('Mean of the 5 neighbors distances for each data point')
plt.xlabel('Count')
plt.ylabel('Mean Distances')

plt.axhline(y = 0.56, color = 'r', linestyle = '--')
```

```
<matplotlib.lines.Line2D at 0x7f2e82d31710>
```




```python
import numpy as np

# Visually determine cutoff values > 0.56
outlier_index = np.where(dist_means > 0.56)
outlier_index
```

```
(array([ 3,  7, 18, 20, 21, 26, 33]),)
```

```
# Filter outlier values (locate them in the dataframe)...
outlier_values = df1.iloc[outlier_index]
outlier_values
```

	Body Weight (kg)	Body Height (m)	Weight-Height Ratio (R)	Internal Moment (M)	Required Torque (N-m)	Torque Category
3	60	1.75	34.286	0.94990	9.3091	Low