Process

1. *Understand the project*
2. *Use this heuristic to decide the value of K*
   n = 8
   K = sqrt(8) = 2.828427 = 3

3. *Using KNN to manually calculate the distance and predict the result*

Accelerometer Data			Gyroscope Data			Fall(+), Not(-)
x	y	z	x	y	z	(+) or (-)
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+
7	**6**	**5**	**5**	**6**	**7**	**??**

Calculate the distance from each data set to the predicted data set:

(Apred - Ax)^2+(Apred - Ay)^2+(Apred - Az)^2+(Gpred - Gx)^2+(Gpred - Gy)^2+(Gpred - Gz)^2

Accelerometer Data			Gyroscope Data			Distance to predict point	Fall(+), Not(-)
x	y	z	x	y	z		(+) or (-)
1	2	3	2	1	3	**10.29563014**	-
2	1	3	3	1	2	**10.39230485**	-
1	1	2	3	2	2	**10.72380529**	-
2	2	3	3	2	1	**10.04987562**	-
6	5	7	5	6	7	**2.449489743**	+
5	6	6	6	5	7	**2.645751311**	+
5	6	7	5	7	6	**3.16227766**	+
7	6	7	6	5	6	**2.645751311**	+
7	**6**	**5**	**5**	**6**	**7**		**??**

The highlighted data sets are ones selected for the prediction which are closest to the predict data set. All three samples selected are in the Fall(+) group .

So with the predict dataset provided, it is predicted to fall(+).

4. *Use Python to implement the application of using KNN to predict fall.*

➔ KNN with original code

```python
import math
# calculate the Euclidean distance between two vectors
#     Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return math.sqrt(distance)
```

```python
# Locate the most similar neighbors
def get_neighbors(train, test_row, k_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  print("List of calsulated distance from predict data:")
  for i in range(len(distances)):
    print(distances[i])
  neighbors = list()
  for i in range(k_neighbors):
    neighbors.append(distances[i][0])
  return neighbors
```

```python
# Make a classification prediction with neighbors
# - test_row is row 0
# - k_neighbors is 3
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row, num_neighbors)
  print("\nNeighbors selected are: ")
  for i in range(len(neighbors)):
    print(neighbors[i])

  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction
```

```
# Test distance function
dataset = [[7,6,5,5,6,7,0],
  [1,2,3,2,1,3,0],
  [2,1,3,3,1,2,0],
  [1,1,2,3,2,2,0],
  [2,2,3,3,2,1,0],
  [6,5,7,5,6,7,1],
  [5,6,6,6,5,7,1],
  [5,6,7,5,7,6,1],
  [7,6,7,6,5,6,1]]

# row 0 (i.e., dataset[0]) is the one to be predicted
prediction = predict_classification(dataset[1:], dataset[0], 3)

# - Display predict result
print('\n')
'Prediction result is Fall(+)' if prediction==1 else '\nPrediction result is Not Fall(-)'

```

```
List of calsulated distance from predict data:
([6, 5, 7, 5, 6, 7, 1], 2.449489742783178)
([5, 6, 6, 6, 5, 7, 1], 2.6457513110645907)
([7, 6, 7, 6, 5, 6, 1], 2.6457513110645907)
([5, 6, 7, 5, 7, 6, 1], 3.1622776601683795)
([2, 2, 3, 3, 2, 1, 0], 10.04987562112089)
([1, 2, 3, 2, 1, 3, 0], 10.295630140987)
([2, 1, 3, 3, 1, 2, 0], 10.392304845413264)
([1, 1, 2, 3, 2, 2, 0], 10.723805294763608)

Neighbors selected are:
[6, 5, 7, 5, 6, 7, 1]
[5, 6, 6, 6, 5, 7, 1]
[7, 6, 7, 6, 5, 6, 1]


'Prediction result is Fall(+)'
```

➔ KNN with sklearn

```
[15]
import sklearn
from sklearn.utils import shuffle
from sklearn.neighbors import KNeighborsClassifier
from sklearn import linear_model, preprocessing
import pandas as pd
import numpy as np

```

```
[16]

from google.colab import files
uploaded = files.upload()
```

Choose Files knn_data.csv

- **knn_data.csv**(text/csv) - 151 bytes, last modified: 2/6/2023 - 100% done

```
Saving knn_data.csv to knn_data.csv
```

```
[18]  1 data = pd.read_csv("knn_data.csv")
      2 data
```

	Ax	Ay	Az	Gx	Gy	Gz	Fall or Not
0	1	2	3	2	1	3	-
1	2	1	3	3	1	2	-
2	1	1	2	3	2	2	-
3	2	2	3	3	2	1	-
4	6	5	7	5	6	7	+
5	5	6	6	6	5	7	+
6	5	6	7	5	7	6	+
7	7	6	7	6	5	6	+

```
[28]  1 Ax= list(data["Ax"])
      2 Ay = list(data["Ay"])
      3 Az = list(data["Az"])
      4 Gx = list(data["Gx"])
      5 Gy = list(data["Gy"])
      6 Gz = list(data["Gz"])
      7 fallOrNot = list(data["Fall or Not"])
      8
      9 X = list(zip(Ax, Ay,Az, Gx, Gy, Gz))
     10 Y = list(fallOrNot)
```

```
[29]  1 x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(X,Y, test_size=0.1)
      2
      3 model = KNeighborsClassifier(n_neighbors=3)
      4 model.fit(x_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
[30]  1 model.score(x_test, y_test)
```

```
1.0
```

```
  1 print("Predicted result is ", model.predict([(7, 6, 5, 5, 6, 7)]))
```

```
Predicted result is  ['+']
```

5. *Comparing the result from the Python program and the result of manual calculation.*

All methods conclude with the prediction of Fall(+)