- Process
  1. Understand the project
     - Most mobile devices are equipped with different kind of sensors.
     - We can use the data sent from Gyroscope senso and Accelerometer sensor to categorize any motion:
       - 3 numbers from Accelerometer sensor
       - 3 numbers from Gyroscope sensor
     - References
       - A Review on Fall Prediction and Prevention System for Personal Devices: Evaluation and Experimental Results
       - Andorid FallArm Project
         - Sensors
           - Difference Between an Accelerometer and a Gyroscope
  2. Use this heuristic to decide the value of K
     The choice of K equal to the odd number cloest to the square root of the number of instances is an empirical rule-of-thumb popularized by the "Pattern Classification" book by Duda et al.
     - References
       - How can we find the optimum K in K-Nearest Neighbor?
  3. Using KNN to manually calculate the distance and predict the result.
     - This is the tranining data and the test data:

Accelerometer Data			**Gyroscope Data**			**Fall (+), Not (-)**
x	**y**	**z**	**x**	**y**	**z**	+/-
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**	??

  4. Use Python to implement the application of using kNN to predict fall.
     a. Create the code by modifying KNN from scratch
        - Explain the code
     b. Run the code on Colab
        - References
          - Get Start with Colab
     - References
       - Python for KNN
         - - Santhi Sree Nagalla, 2021 Summer
           - Knn.ipynb - Python code using Scikit-learn library
           - Knn_Python.ipynb - Python Program to predict kNN Value
         - Develop k-Nearest Neighbors in Python From Scratch
         - Quan Zhou - Fall, 2019
         - Wijian Xiong - Fall, 2019
         - Juilee Panse - Fall, 2019
  5. Comparing the result from the Python program and the result of manual calculation.
  6. Adding the project to your portofolio
     a. Please use Google Slides to document the project
     b. Please link your presentation on GitHub using this structure

```
Machine Learning
  - Supervised Learning
    + Falling Prediction using KNN
```

  7. Submit
     a. The URLs of the Google Slides and GitHub web pages related to this project.
     b. A PDF file of your Google Slides
- References
  - Java for KNN
    - Java for K-NN
  - R for K-NN
  - FallArm Project
  - kNN algorithm
  - R
    - SparkR
      - SparkR setup
    - RStudio

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

Let’s take an example:

Step 1: Table of Input data -

- Most mobile devices are equipped with different kind of sensors.
- We can use the data sent from Gyroscope and Accelerometer sensors to categorize any motion. (3 numbers from Accelerometer sensor, 3 numbers from Gyroscope sensor)
- This is the training data and the test data:

Accelerometer Data			**Gyroscope Data**			**Fall (+), Not (-)**
x	**y**	**z**	**x**	**y**	**z**	**+/-**
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**	**??**

Step 2:

Suppose we determine K = 8 (we will use 8 nearest neighbors) as parameter of this algorithm.

- Then we calculate the distance between the query-instance and all the training samples. Because we use only quantitative X_i, we can use Euclidean distance.
- Suppose the query instance have coordinates (X_1^q, X_2^q) and the coordinate of training sample is (X_1^t, X_2^t) then square Euclidean distance is $\mathbf{d_{tq}^2 = (X_1^t - X_1^q)^2 + (X_2^t - X_2^q)^2}$

N = 8

K = sqrt N = sqrt 8 = 3

$\mathbf{d_{tq}^2 = (X_1^t - X_1^q)^2 + (X_2^t - X_2^q)^2}$

(Target x - Data x) ^ 2 + (Target y – Data y) ^ 2 + (Target z – Data z) ^ 2

Step 3: Find the distance using the formula in step 2-

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Dist. Acce	Dist. Gyro
x	y	z	x	y	z	+/-		
1	2	3	2	1	3	-	56	50
2	1	3	3	1	2	-	54	54
1	1	2	3	2	2	-	70	45
2	2	3	3	2	1	-	45	56
6	5	7	5	6	7	+	6	0
5	6	6	6	5	7	+	5	2
5	6	7	5	7	6	+	8	2
7	6	7	6	5	6	+	4	3
7	6	5	5	6	7	+		

Since K = 3,

3 smallest numbers for Dist. Acce are 4, 5 and 6.

Since all numbers represent positive, the query instance is plus (+).

3 smallest numbers for Dist. Gyro are 0, 2 and 2.

Since all numbers represent positive, the query instance is plus (+).

Step 4: Find the K-nearest neighbors

We include a training sample as nearest neighbors if the distance of this training sample to the query instance is less than or equal to the K-th smallest distance.

If the distance of the training sample is below the K-th minimum, then we gather the category Y of this nearest neighbors' training samples.

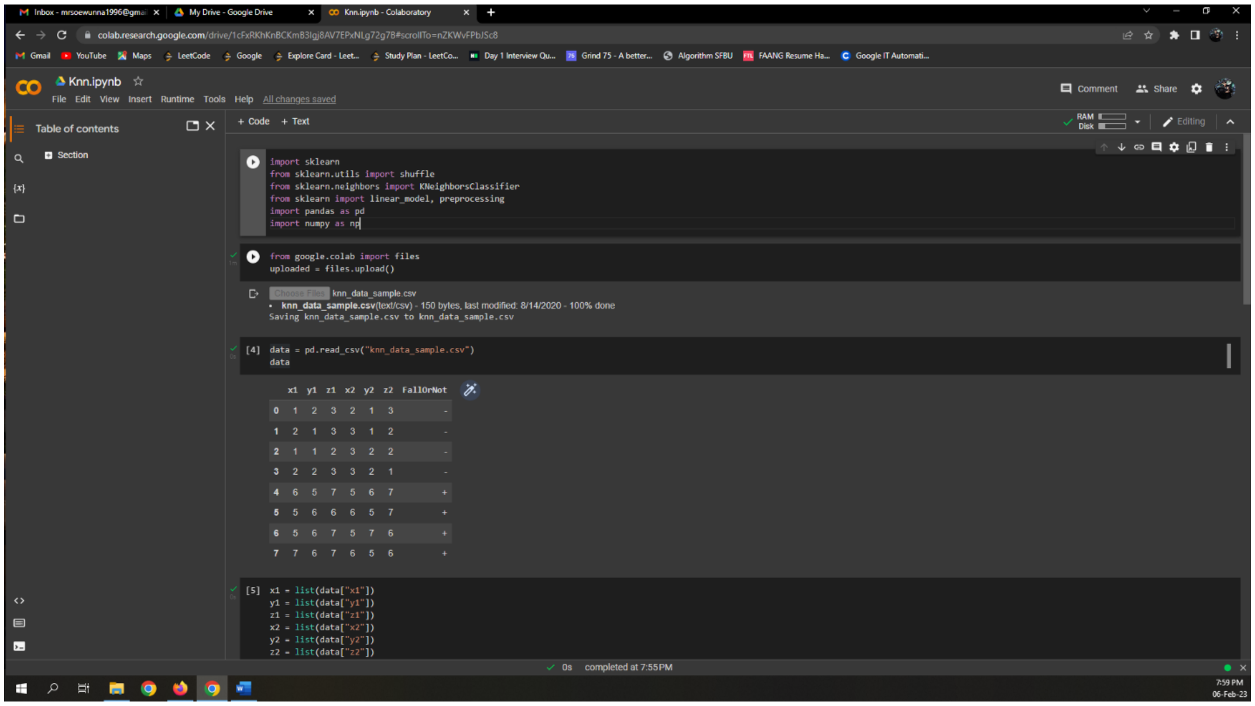
Some special case happens in our example that the 3rd until the 8 th minimum distance happen to be the same.

- In this case we directly use the highest K=8 because choosing arbitrary among the 3rd until the 8th nearest neighbors is unstable. The KNN prediction of the query instance is based on simple majority of the category of nearest neighbors.
- In our example, the data is only binary, thus the majority can be taken as simple as counting the number of '+' and '-' signs.

If the number of plus is greater than minus, we predict the query instance as plus and vice versa.

If the number of plus is equal to minus, we can choose arbitrary or determine as one of the plus or minus.

Running the notebook file in Colab. knn_data_sample.csv is uploaded from local drive.



```
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
```

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

```
knn_data_sample.csv
• knn_data_sample.csv(text/csv) - 150 bytes, last modified: 8/14/2020 - 100% done
Saving knn_data_sample.csv to knn_data_sample.csv
```

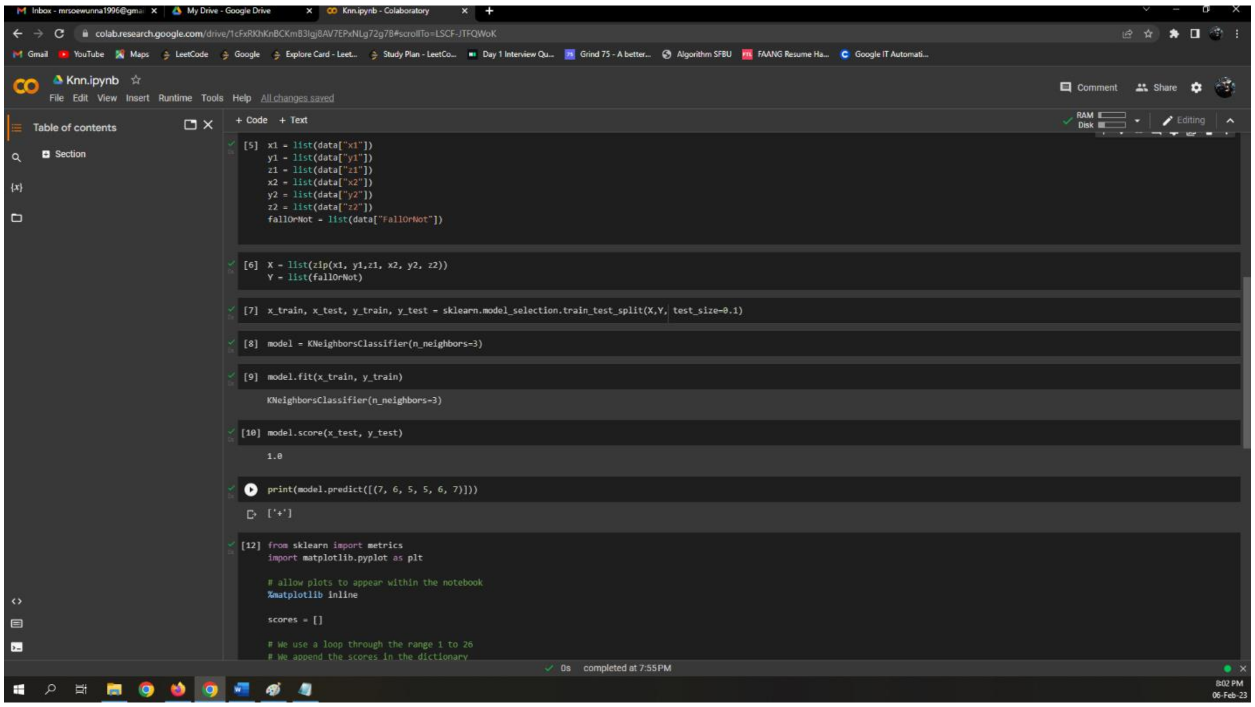
```
[4] data = pd.read_csv("knn_data_sample.csv")
    data
```

	x1	y1	z1	x2	y2	z2	FallOrNot
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	+

```
[5] x1 = list(data["x1"])
    y1 = list(data["y1"])
    z1 = list(data["z1"])
    x2 = list(data["x2"])
    y2 = list(data["y2"])
    z2 = list(data["z2"])
```

```
0s   completed at 7:55PM
```

We got the same result as manual calculation. ['+']



```
[5] x1 = list(data["x1"])
    y1 = list(data["y1"])
    z1 = list(data["z1"])
    x2 = list(data["x2"])
    y2 = list(data["y2"])
    z2 = list(data["z2"])
    fallOrNot = list(data["FallOrNot"])
```

```
[6] X = list(zip(x1, y1,z1, x2, y2, z2))
    Y = list(fallOrNot)
```

```
[7] x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(X,Y, test_size=0.1)
```

```
[8] model = KNeighborsClassifier(n_neighbors=3)
```

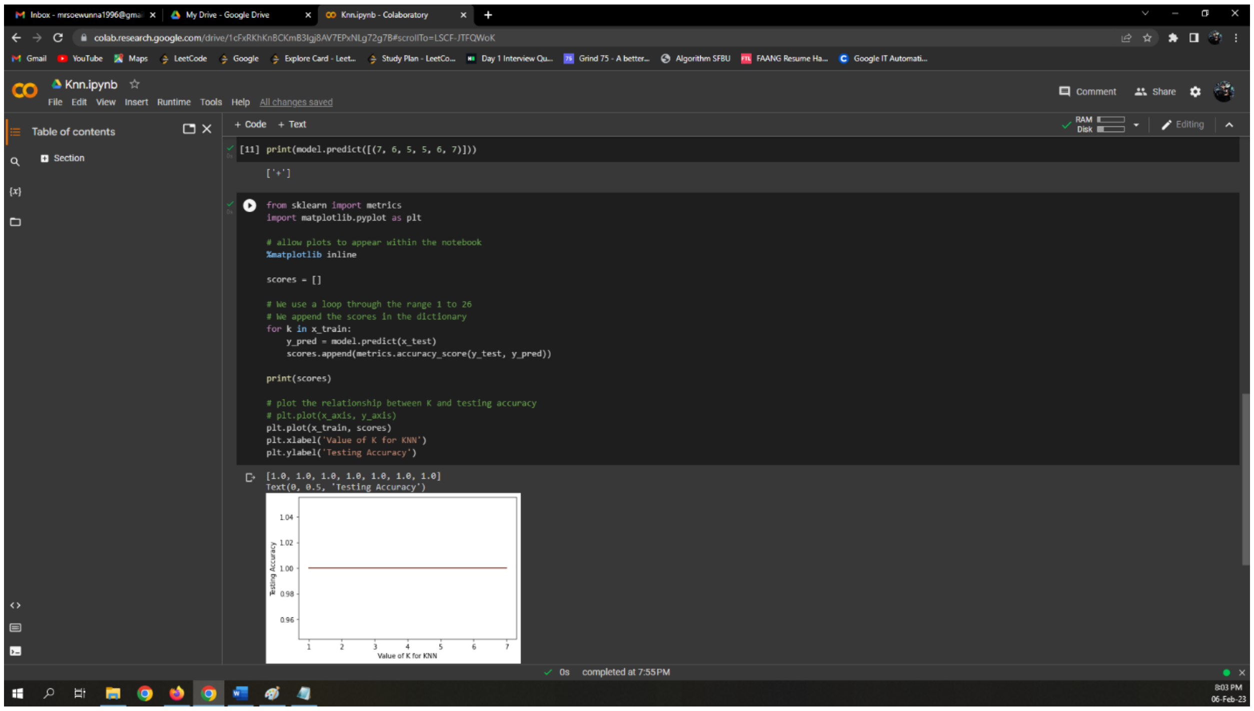
```
[9] model.fit(x_train, y_train)

    KNeighborsClassifier(n_neighbors=3)
```

```
[10] model.score(x_test, y_test)

     1.0
```

```
print(model.predict([(7, 6, 5, 5, 6, 7)]))

['+']
```

```
[12] from sklearn import metrics
     import matplotlib.pyplot as plt

     # allow plots to appear within the notebook
     %matplotlib inline

     scores = []

     # We use a loop through the range 1 to 26
     # We append the scores in the dictionary
```

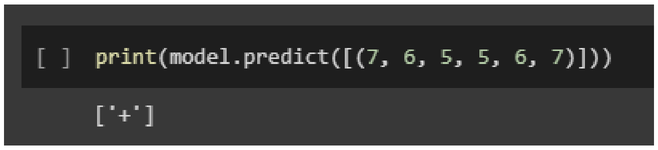
```
0s   completed at 7:55PM
```



```
[11] print(model.predict([(7, 6, 5, 5, 6, 7)]))

['+']
```

```
from sklearn import metrics
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

scores = []

# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in x_train:
    y_pred = model.predict(x_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

print(scores)

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(x_train, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Text(0, 0.5, 'Testing Accuracy')
```



```
[ ] print(model.predict([(7, 6, 5, 5, 6, 7)]))

    ['+']
```