CII2M3 INTRODUCTION OF ARTIFICIAL INTELLIGENCE

EVEN SEMESTER SESSION 2021/2022

ASSIGNMENT 3 - LEARNING

Group No : 10

Section: IF-44-INT

Lecturer Name: Edward Ferdian

Group Member:

NAME	STUDENT ID
NUR SABRINA SYAZA BINTI ZAHAR HISHAM	1301213666
NUR FASIHAH AYUNI BINTI MOHD YAHYA	1301213670
NURUL IMAN BINTI NORDIN	1301213669

TABLE OF CONTENT

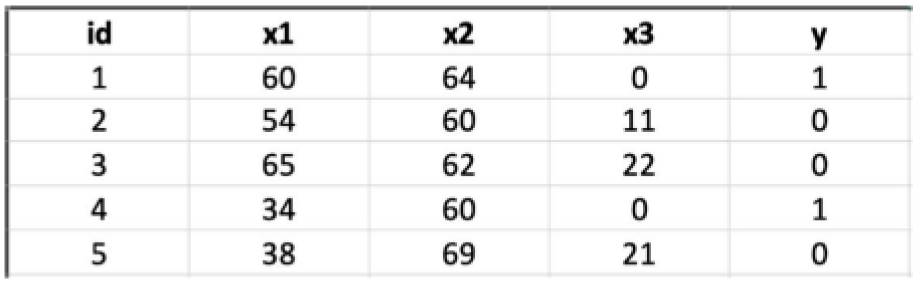
Readme link: https://github.com/Chiko21/Programming-Assignment-3/blob/main/README.md

PowerPoint:

https://docs.google.com/presentation/d/1D5ylZjSp5E-L5y415mnzt8ahkczknYFq4hgPLafLMfQ/edit?usp=sharing

QUESTION:

Given file traintest.xlsx which contains 2 sheets: train and test, consists of a dataset for binary classification problems. Each record/row comprised of row number (id), input features (x1, x2, x3), and class output (y). Input features are integers within a certain range for each different feature. Class output is binary (either 0 or 1).



id	x1	x2	x3	y
1	60	64	0	1
2	54	60	11	0
3	65	62	22	0
4	34	60	0	1
5	38	69	21	0

The train sheet comprised of 296 data, complete with class target output (*y*). Use this sheet to model or training depending on the learning method you use. The test sheet comprised of 10 data, with hidden class output. Please use this sheet to test your model. In the end, the output of your model will be compared with the target or actual classes.

CHAPTER 1
PROBLEM DESCRIPTION

We have been given dataset file “traintest.xlsx” which train sheet comprised of 296 data, complete with class target output (*y*). From the dataset, we have to test the model depending on the learning method that we choose. For our assignment, the learning method that we use to train the model is K-Nearest Neighbor (KNN). Why do we choose kNN? kNN is the most accurate model as it makes highly accurate predictions and it is the simplest learning method compared to other algorithms. From that, by using kNN we have to test the model by implementing it in code. The language that we use to implement the code is Python. To test the model, there are three steps which are training, testing and evolution.

CHAPTER 2
OBSERVATION

2.1 Modeling Method

We are using the Manhattan **distance** formula to calculate the distance between 2 points which is the length of the line segment that connects them. For example, the dataset is composed of 296 attributes and it represents 296 dimensional space.

The formula to calculate the Manhattan distance :

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

2.2 Normalizing Data

Before we process the data, we do the **Normalization** which is we recalculate the data to values between 0 (minimum value) and 1 (maximum value). The normalized values will always lie between 0 and 1, which is a nice feature. If we compute distance for non-normalized data, the effect is it assigns higher weights to the variable with wider ranges of values.

The formula of Normalization:

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

2.3 Split the Data into a Training Set and Test Set

Next, we split the data into training and test set.The data is part of K-Fold Cross Validation. For our model, we split the data to train and test that include all of the data from all column except from column ‘y’ by creating 5 new dataset :

1. From row 1 until row 59 as testing set and the balance is training set
2. From row 60 until row 118 as testing set while the balance is training set
3. From row 119 until row 177 as testing set while the balance is training set
4. From row 178 until row 236 as testing set while the balance is training set
5. From row 237 until row 296 as testing set while the balance is training set

2.4 Validation

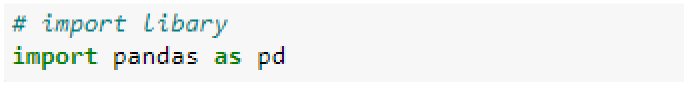
Moreover, before we test the training data, we have to do validation first as we can choose the best K and Fold is suitable to use for testing . For the validation method, it will produce accuracy for every fold which is measured by factor k . The best accuracy will be chosen which has the best fold for testing data.

Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

CHAPTER 3
IMPLEMENTATION

A. Import Library

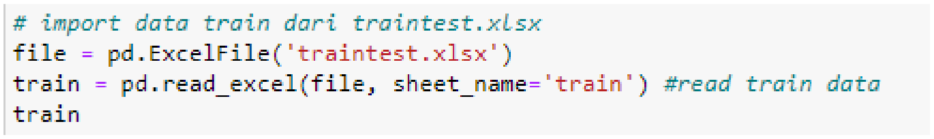
The library that we use is pandas to import and read the data in format .xlsx



```
# import libary
import pandas as pd
```

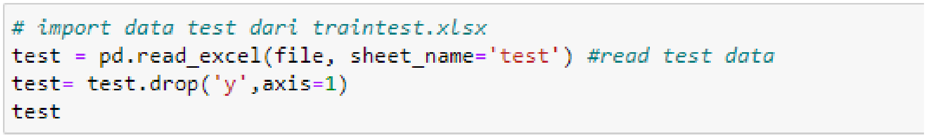
B. Import Data

Import file ‘traintest.xlsx’ and read the column train



```
# import data train dari traintest.xlsx
file = pd.ExcelFile('traintest.xlsx')
train = pd.read_excel(file, sheet_name='train') #read train data
train
```

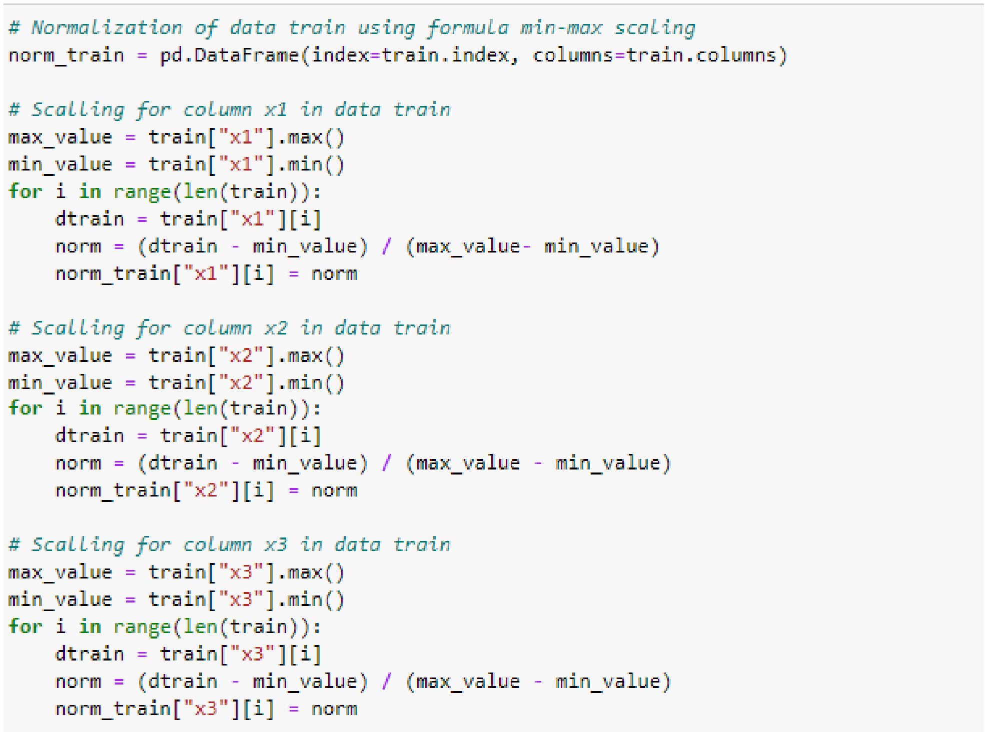
Read column ‘test’



```
# import data test dari traintest.xlsx
test = pd.read_excel(file, sheet_name='test') #read test data
test= test.drop('y',axis=1)
test
```

C. Model Training

Normalization for data train :



```
# Normalization of data train using formula min-max scaling
norm_train = pd.DataFrame(index=train.index, columns=train.columns)

# Scalling for column x1 in data train
max_value = train["x1"].max()
min_value = train["x1"].min()
for i in range(len(train)):
    dtrain = train["x1"][i]
    norm = (dtrain - min_value) / (max_value- min_value)
    norm_train["x1"][i] = norm

# Scalling for column x2 in data train
max_value = train["x2"].max()
min_value = train["x2"].min()
for i in range(len(train)):
    dtrain = train["x2"][i]
    norm = (dtrain - min_value) / (max_value - min_value)
    norm_train["x2"][i] = norm

# Scalling for column x3 in data train
max_value = train["x3"].max()
min_value = train["x3"].min()
for i in range(len(train)):
    dtrain = train["x3"][i]
    norm = (dtrain - min_value) / (max_value - min_value)
    norm_train["x3"][i] = norm
```

```
# the result of normalization for column x1, x2, x3 of data train in float
norm_train['x1'] = norm_train['x1'].astype(float)
norm_train['x2'] = norm_train['x2'].astype(float)
norm_train['x3'] = norm_train['x3'].astype(float)
norm_train['y'] = train['y']
norm_train['id'] = train['id']
norm_train
```

Output:

Data before normalization:

	id	x1	x2	x3	y
0	1	60	64	0	1
1	2	54	60	11	0
2	3	65	62	22	0
3	4	34	60	0	1
4	5	38	69	21	0
...	...	...	...	...	...
291	292	59	64	1	1
292	293	65	67	0	1
293	294	53	65	12	0
294	295	57	64	1	0
295	296	54	59	7	1

296 rows × 5 columns

Data after normalization:

	id	x1	x2	x3	y
0	1	0.566038	0.545455	0.000000	1
1	2	0.452830	0.181818	0.211538	0
2	3	0.660377	0.363636	0.423077	0
3	4	0.075472	0.181818	0.000000	1
4	5	0.150943	1.000000	0.403846	0
...	...	...	...	...	...
291	292	0.547170	0.545455	0.019231	1
292	293	0.660377	0.818182	0.000000	1
293	294	0.433962	0.636364	0.230769	0
294	295	0.509434	0.545455	0.019231	0
295	296	0.452830	0.090909	0.134615	1

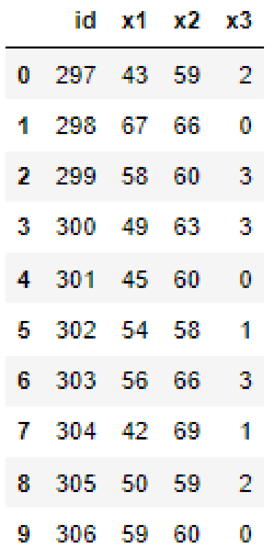
296 rows × 5 columns

Normalization for data test :

```
# Normalization of data test using formula min-max scalling
norm_test = pd.DataFrame(index=test.index, columns=test.columns)

# Scalling for column x1 in data test
max_value = test["x1"].max()
min_value = test["x1"].min()
for i in range(len(test)):
    x = test["x1"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x1"][i] = float(norm)

# Scalling for column x2 in data test
max_value = test["x2"].max()
min_value = test["x2"].min()
for i in range(len(test)):
    x = test["x2"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x2"][i] = float(norm)

# Scalling for column x3 in data test
max_value = test["x3"].max()
min_value = test["x3"].min()
for i in range(len(test)):
    x = test["x3"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x3"][i] = float(norm)

# the result of normalization for column x1, x2, x3 of data test in float
norm_test['x1'] = norm_test['x1'].astype(float)
norm_test['x2'] = norm_test['x2'].astype(float)
norm_test['x3'] = norm_test['x3'].astype(float)
norm_test['id'] = test['id']
norm_test
```

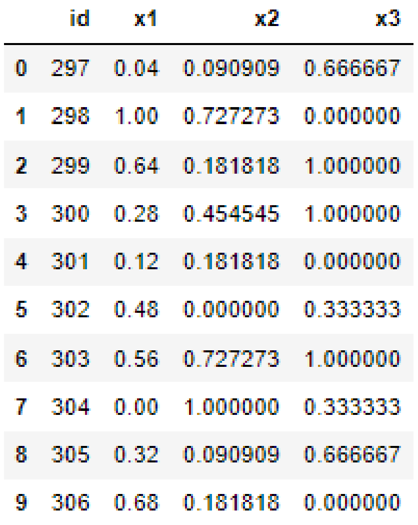
Output:

Data before normalization:



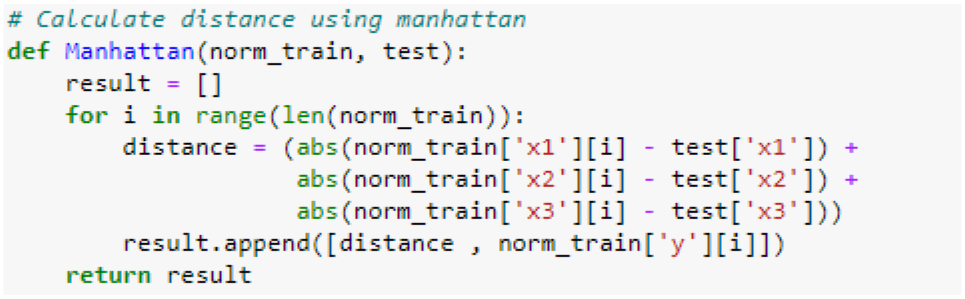
	id	x1	x2	x3
0	297	43	59	2
1	298	67	66	0
2	299	58	60	3
3	300	49	63	3
4	301	45	60	0
5	302	54	58	1
6	303	56	66	3
7	304	42	69	1
8	305	50	59	2
9	306	59	60	0

Data after normalization:



	id	x1	x2	x3
0	297	0.04	0.090909	0.666667
1	298	1.00	0.727273	0.000000
2	299	0.64	0.181818	1.000000
3	300	0.28	0.454545	1.000000
4	301	0.12	0.181818	0.000000
5	302	0.48	0.000000	0.333333
6	303	0.56	0.727273	1.000000
7	304	0.00	1.000000	0.333333
8	305	0.32	0.090909	0.666667
9	306	0.68	0.181818	0.000000

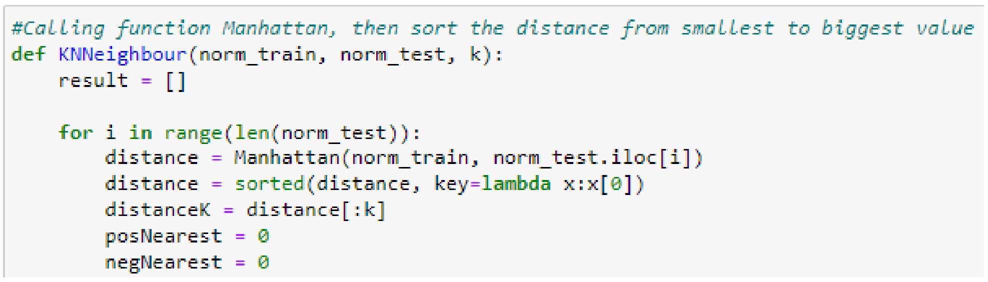
To store the result model training by using Manhattan distance :



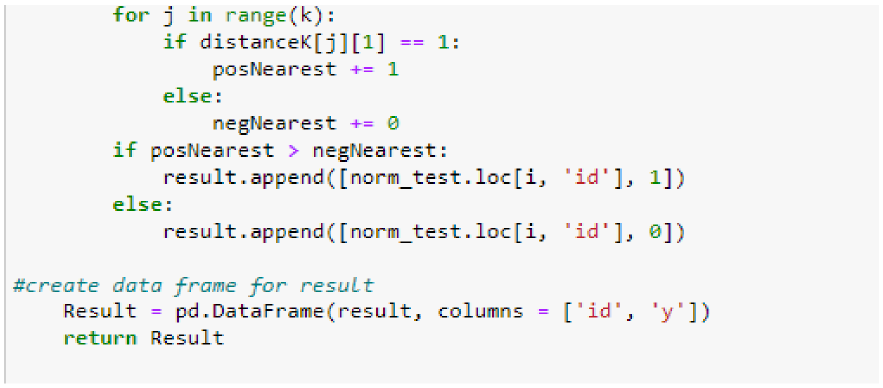
```
# Calculate distance using manhattan
def Manhattan(norm_train, test):
    result = []
    for i in range(len(norm_train)):
        distance = (abs(norm_train['x1'][i] - test['x1']) +
                    abs(norm_train['x2'][i] - test['x2']) +
                    abs(norm_train['x3'][i] - test['x3']))
        result.append([distance , norm_train['y'][i]])
    return result
```

D. Model Testing

This function is to sort the distance from small value to biggest value and to determine ‘y’ in Manhattan 1 or 0.



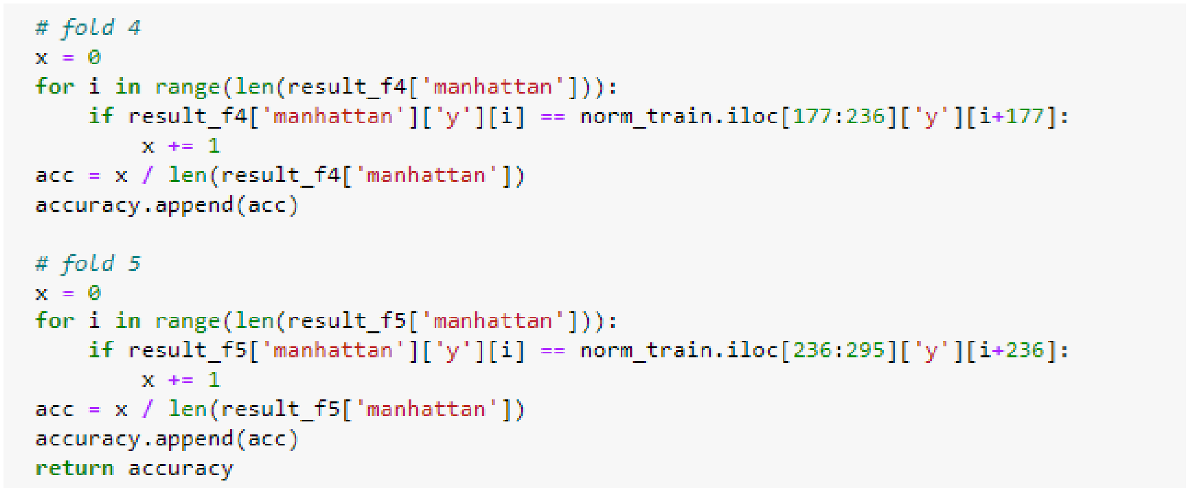
```
#Calling function Manhattan, then sort the distance from smallest to biggest value
def KNNeighbour(norm_train, norm_test, k):
    result = []

    for i in range(len(norm_test)):
        distance = Manhattan(norm_train, norm_test.iloc[i])
        distance = sorted(distance, key=lambda x:x[0])
        distanceK = distance[:k]
        posNearest = 0
        negNearest = 0
```



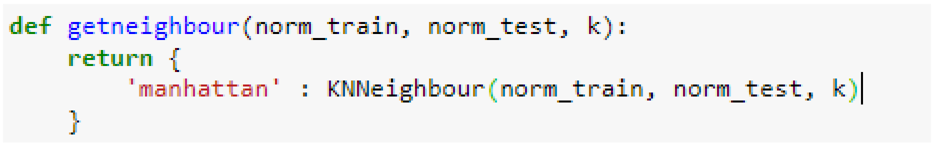
```
        for j in range(k):
            if distanceK[j][1] == 1:
                posNearest += 1
            else:
                negNearest += 0
        if posNearest > negNearest:
            result.append([norm_test.loc[i, 'id'], 1])
        else:
            result.append([norm_test.loc[i, 'id'], 0])

#create data frame for result
    Result = pd.DataFrame(result, columns = ['id', 'y'])
    return Result
```

E. Model Evaluation

```python
def Validation(norm_train, k):

    accuracy = []
    # spilt the normalization of dataset train and dataset test to create 5 new dataset
    ftest1 = norm_train.iloc[:59]
    ftrain1 = norm_train.iloc[59:].reset_index().drop('index', axis = 1)
    ftest2 = norm_train.iloc[59:118].reset_index().drop('index', axis = 1)
    ftrain2 = pd.concat([norm_train.iloc[:59], norm_train.iloc[118:]]).reset_index().drop('index', axis = 1)
    ftest3 = norm_train.iloc[118:177].reset_index().drop('index', axis = 1)
    ftrain3 = pd.concat([norm_train.iloc[:118], norm_train.iloc[177:]]).reset_index().drop('index', axis = 1)
    ftest4 = norm_train.iloc[177:236].reset_index().drop('index', axis = 1)
    ftrain4 = pd.concat([norm_train.iloc[:177], norm_train.iloc[236:]]).reset_index().drop('index', axis = 1)
    ftest5 = norm_train.iloc[236:295].reset_index().drop('index', axis = 1)
    ftrain5 = norm_train.iloc[0:236].reset_index().drop('index', axis = 1)

    result_f1 = getneighbour(ftrain1, ftest1, k)
    result_f2 = getneighbour(ftrain2, ftest2, k)
    result_f3 = getneighbour(ftrain3, ftest3, k)
    result_f4 = getneighbour(ftrain4, ftest4, k)
    result_f5 = getneighbour(ftrain5, ftest5, k)

    # fold 1
    x = 0
    for i in range(len(result_f1['manhattan'])):
        if result_f1['manhattan']['y'][i] == norm_train['y'][i]:
            x += 1
    acc = x / len(result_f1['manhattan'])
    accuracy.append(acc)

    # fold 2
    x = 0
    for i in range(len(result_f2['manhattan'])):
        if result_f2['manhattan']['y'][i] == norm_train.iloc[59:118]['y'][i+59]:
            x += 1
    acc = x / len(result_f2['manhattan'])
    accuracy.append(acc)

    # fold 3
    x = 0
    for i in range(len(result_f3['manhattan'])):
        if result_f3['manhattan']['y'][i] == norm_train.iloc[118:177]['y'][i+118]:
            x += 1
    acc = x / len(result_f3['manhattan'])
    accuracy.append(acc)
```

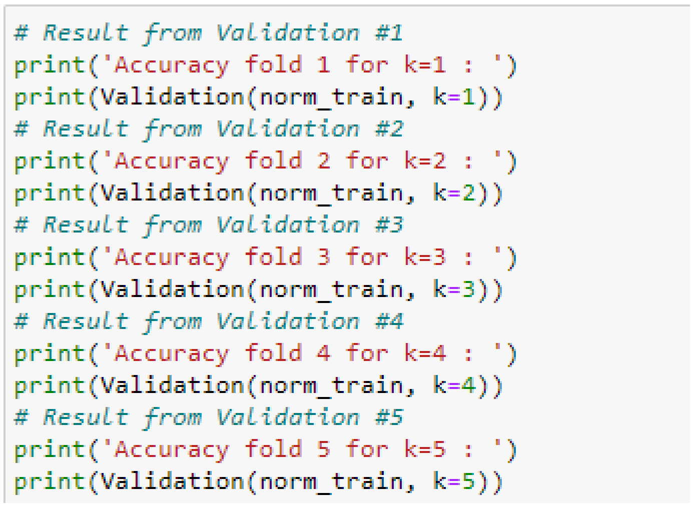


```
# fold 4
x = 0
for i in range(len(result_f4['manhattan'])):
    if result_f4['manhattan']['y'][i] == norm_train.iloc[177:236]['y'][i+177]:
        x += 1
acc = x / len(result_f4['manhattan'])
accuracy.append(acc)

# fold 5
x = 0
for i in range(len(result_f5['manhattan'])):
    if result_f5['manhattan']['y'][i] == norm_train.iloc[236:295]['y'][i+236]:
        x += 1
acc = x / len(result_f5['manhattan'])
accuracy.append(acc)
return accuracy
```



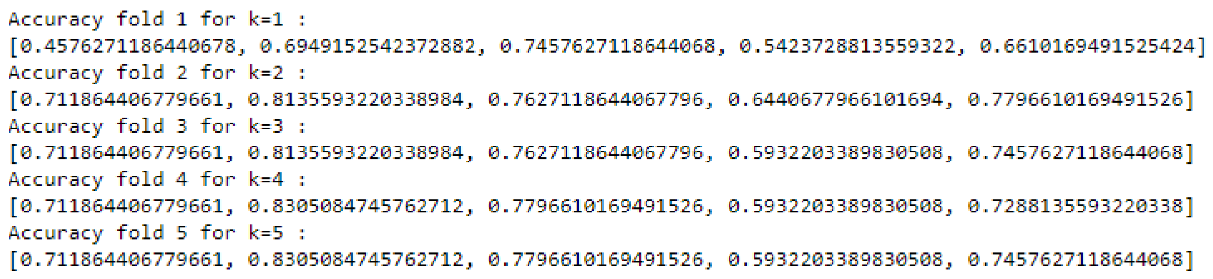
```
def getneighbour(norm_train, norm_test, k):
    return {
        'manhattan' : KNNeighbour(norm_train, norm_test, k)
    }
```

Print the value from validation :



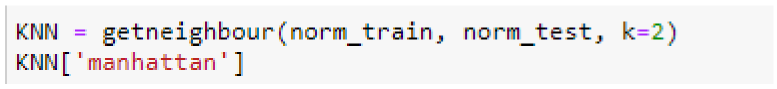
```
# Result from Validation #1
print('Accuracy fold 1 for k=1 : ')
print(Validation(norm_train, k=1))
# Result from Validation #2
print('Accuracy fold 2 for k=2 : ')
print(Validation(norm_train, k=2))
# Result from Validation #3
print('Accuracy fold 3 for k=3 : ')
print(Validation(norm_train, k=3))
# Result from Validation #4
print('Accuracy fold 4 for k=4 : ')
print(Validation(norm_train, k=4))
# Result from Validation #5
print('Accuracy fold 5 for k=5 : ')
print(Validation(norm_train, k=5))
```

Output:



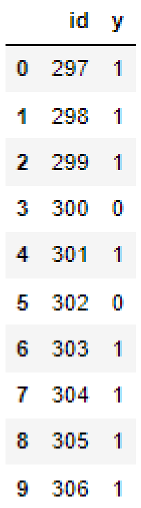
```
Accuracy fold 1 for k=1 :
[0.4576271186440678, 0.6949152542372882, 0.7457627118644068, 0.5423728813559322, 0.6610169491525424]
Accuracy fold 2 for k=2 :
[0.711864406779661, 0.8135593220338984, 0.7627118644067796, 0.6440677966101694, 0.7796610169491526]
Accuracy fold 3 for k=3 :
[0.711864406779661, 0.8135593220338984, 0.7627118644067796, 0.5932203389830508, 0.7457627118644068]
Accuracy fold 4 for k=4 :
[0.711864406779661, 0.8305084745762712, 0.7796610169491526, 0.5932203389830508, 0.7288135593220338]
Accuracy fold 5 for k=5 :
[0.711864406779661, 0.8305084745762712, 0.7796610169491526, 0.5932203389830508, 0.7457627118644068]
```

Testing to best k:



```
KNN = getneighbour(norm_train, norm_test, k=2)
KNN['manhattan']
```

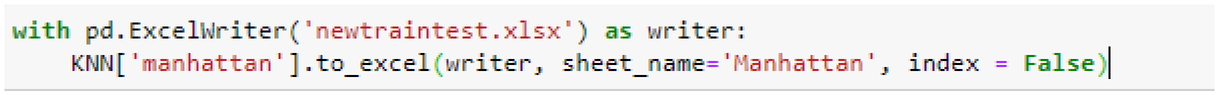
Output:



	id	y
0	297	1
1	298	1
2	299	1
3	300	0
4	301	1
5	302	0
6	303	1
7	304	1
8	305	1
9	306	1

F. Create .xlsx file

We create the .xlsx file to store the value of ‘y’.



```
with pd.ExcelWriter('newtraintest.xlsx') as writer:
    KNN['manhattan'].to_excel(writer, sheet_name='Manhattan', index = False)
```

The Code

```
In [1]: # import libary
        import pandas as pd
        import math
```

Import Data

```
In [2]: # import data train dari traintest.xlsx
        file = pd.ExcelFile('traintest.xlsx')
        train = pd.read_excel(file, sheet_name='train') #read train data
        train
```

Out[2]:

	id	x1	x2	x3	y
0	1	60	64	0	1
1	2	54	60	11	0
2	3	65	62	22	0
3	4	34	60	0	1
4	5	38	69	21	0
...	...	...	...	...	...
291	292	59	64	1	1
292	293	65	67	0	1
293	294	53	65	12	0
294	295	57	64	1	0
295	296	54	59	7	1

296 rows × 5 columns

```
In [3]: # import data test dari traintest.xlsx
        test = pd.read_excel(file, sheet_name='test') #read test data
        test= test.drop('y',axis=1)
        test
```

Out[3]:

	id	x1	x2	x3
0	297	43	59	2
1	298	67	66	0
2	299	58	60	3
3	300	49	63	3
4	301	45	60	0
5	302	54	58	1
6	303	56	66	3
7	304	42	69	1
8	305	50	59	2
9	306	59	60	0

Data Preprocessing

In [4]:

```python
# Normalization of data train using formula min-max scaling
norm_train = pd.DataFrame(index=train.index, columns=train.columns)

# Scalling for column x1 in data train
max_value = train["x1"].max()
min_value = train["x1"].min()
for i in range(len(train)):
    dtrain = train["x1"][i]
    norm = (dtrain - min_value) / (max_value- min_value)
    norm_train["x1"][i] = norm

# Scalling for column x2 in data train
max_value = train["x2"].max()
min_value = train["x2"].min()
for i in range(len(train)):
    dtrain = train["x2"][i]
    norm = (dtrain - min_value) / (max_value - min_value)
    norm_train["x2"][i] = norm

# Scalling for column x3 in data train
max_value = train["x3"].max()
min_value = train["x3"].min()
for i in range(len(train)):
    dtrain = train["x3"][i]
    norm = (dtrain - min_value) / (max_value - min_value)
    norm_train["x3"][i] = norm
```

```python
# the result of normalization for column x1, x2, x3 of data train in float
norm_train['x1'] = norm_train['x1'].astype(float)
norm_train['x2'] = norm_train['x2'].astype(float)
norm_train['x3'] = norm_train['x3'].astype(float)
norm_train['y'] = train['y']
norm_train['id'] = train['id']
norm_train
```

Out[4]:

	id	x1	x2	x3	y
0	1	0.566038	0.545455	0.000000	1
1	2	0.452830	0.181818	0.211538	0
2	3	0.660377	0.363636	0.423077	0
3	4	0.075472	0.181818	0.000000	1
4	5	0.150943	1.000000	0.403846	0
...	...	...	...	...	...
291	292	0.547170	0.545455	0.019231	1
292	293	0.660377	0.818182	0.000000	1
293	294	0.433962	0.636364	0.230769	0
294	295	0.509434	0.545455	0.019231	0
295	296	0.452830	0.090909	0.134615	1

296 rows × 5 columns

In [5]:
```
# Normalization of data test using formula min-max scaling
norm_test = pd.DataFrame(index=test.index, columns=test.columns)

# Scalling for column x1 in data test
max_value = test["x1"].max()
min_value = test["x1"].min()
for i in range(len(test)):
    x = test["x1"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x1"][i] = float(norm)

# Scalling for column x2 in data test
max_value = test["x2"].max()
min_value = test["x2"].min()
for i in range(len(test)):
    x = test["x2"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x2"][i] = float(norm)

# Scalling for column x3 in data test
max_value = test["x3"].max()
min_value = test["x3"].min()
for i in range(len(test)):
    x = test["x3"][i]
    norm = (x - min_value) / (max_value - min_value)
    norm_test["x3"][i] = float(norm)

# the result of normalization for column x1, x2, x3 of data test in float
norm_test['x1'] = norm_test['x1'].astype(float)
norm_test['x2'] = norm_test['x2'].astype(float)
norm_test['x3'] = norm_test['x3'].astype(float)
norm_test['id'] = test['id']
norm_test
```

Out[5]:

	id	x1	x2	x3
0	297	0.04	0.090909	0.666667
1	298	1.00	0.727273	0.000000
2	299	0.64	0.181818	1.000000
3	300	0.28	0.454545	1.000000
4	301	0.12	0.181818	0.000000
5	302	0.48	0.000000	0.333333
6	303	0.56	0.727273	1.000000
7	304	0.00	1.000000	0.333333
8	305	0.32	0.090909	0.666667
9	306	0.68	0.181818	0.000000

Calculate distance

using **Euclidean Distance**

In [6]:
```
# Calculate distance using manhattan
def Manhattan(norm_train, test):
    result = []
    for i in range(len(norm_train)):
        distance = (abs(norm_train['x1'][i] - test['x1']) +
                    abs(norm_train['x2'][i] - test['x2']) +
                    abs(norm_train['x3'][i] - test['x3']))
        result.append([distance , norm_train['y'][i]])
    return result
```

In [7]:
```
#Calling function Manhattan, then sort the distance from smallest to biggest value
def KNNeighbour(norm_train, norm_test, k):
    result = []

    for i in range(len(norm_test)):
        distance = Manhattan(norm_train, norm_test.iloc[i])
        distance = sorted(distance, key=lambda x:x[0])
        distanceK = distance[:k]
        posNearest = 0
        negNearest = 0
        for j in range(k):
            if distanceK[j][1] == 1:
                posNearest += 1
            else:
                negNearest += 0
        if posNearest > negNearest:
            result.append([norm_test.loc[i, 'id'], 1])
        else:
            result.append([norm_test.loc[i, 'id'], 0])

#create data frame for result
    Result = pd.DataFrame(result, columns = ['id', 'y'])
    return Result
```

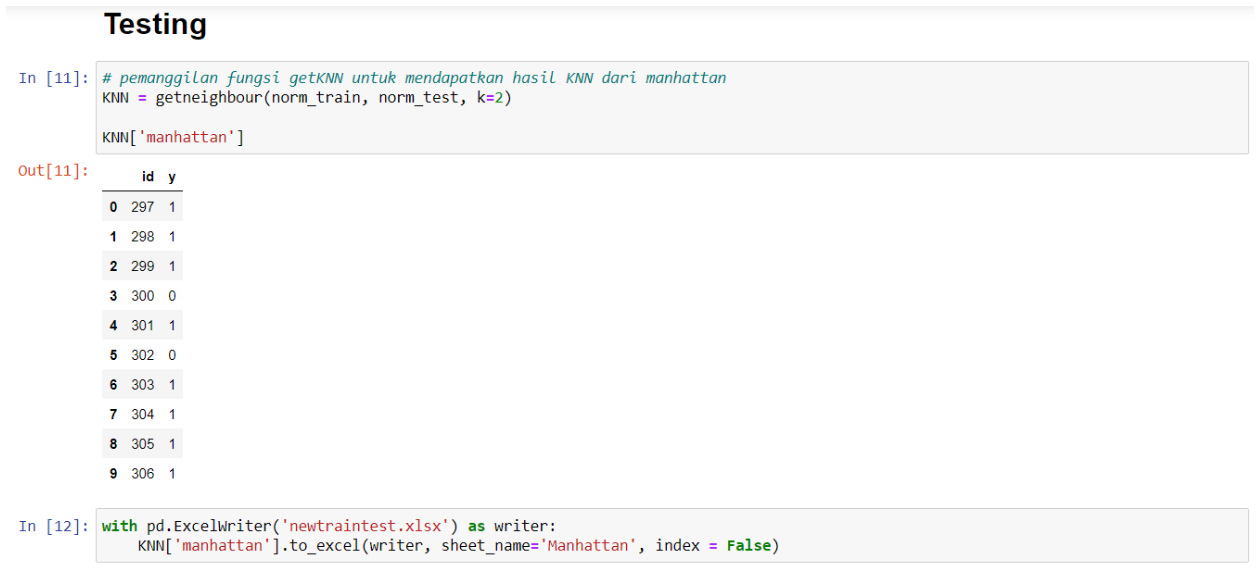
In [8]:
```
def getneighbour(norm_train, norm_test, k):
    return {
        'manhattan' : KNNeighbour(norm_train, norm_test, k)
    }
```

Evaluation

In [9]:
```
def Validation(norm_train, k):

    accuracy = []
    # spilt the normalization of dataset train and dataset test to create 5 new dataset
    ftest1 = norm_train.iloc[:59]
    ftrain1 = norm_train.iloc[59:].reset_index().drop('index', axis = 1)
    ftest2 = norm_train.iloc[59:118].reset_index().drop('index', axis = 1)
    ftrain2 = pd.concat([norm_train.iloc[:59], norm_train.iloc[118:]]).reset_index().drop('index', axis = 1)
    ftest3 = norm_train.iloc[118:177].reset_index().drop('index', axis = 1)
    ftrain3 = pd.concat([norm_train.iloc[:118], norm_train.iloc[177:]]).reset_index().drop('index', axis = 1)
    ftest4 = norm_train.iloc[177:236].reset_index().drop('index', axis = 1)
    ftrain4 = pd.concat([norm_train.iloc[:177], norm_train.iloc[236:]]).reset_index().drop('index', axis = 1)
    ftest5 = norm_train.iloc[236:295].reset_index().drop('index', axis = 1)
    ftrain5 = norm_train.iloc[0:236].reset_index().drop('index', axis = 1)

    result_f1 = getneighbour(ftrain1, ftest1, k)
    result_f2 = getneighbour(ftrain2, ftest2, k)
    result_f3 = getneighbour(ftrain3, ftest3, k)
    result_f4 = getneighbour(ftrain4, ftest4, k)
    result_f5 = getneighbour(ftrain5, ftest5, k)

    # fold 1
    x = 0
    for i in range(len(result_f1['manhattan'])):
        if result_f1['manhattan']['y'][i] == norm_train['y'][i]:
            x += 1
    acc = x / len(result_f1['manhattan'])
    accuracy.append(acc)
    # fold 2
    x = 0
    for i in range(len(result_f2['manhattan'])):
        if result_f2['manhattan']['y'][i] == norm_train.iloc[59:118]['y'][i+59]:
            x += 1
    acc = x / len(result_f2['manhattan'])
    accuracy.append(acc)

    # fold 3
    x = 0
    for i in range(len(result_f3['manhattan'])):
        if result_f3['manhattan']['y'][i] == norm_train.iloc[118:177]['y'][i+118]:
            x += 1
    acc = x / len(result_f3['manhattan'])
    accuracy.append(acc)

    # fold 4
    x = 0
    for i in range(len(result_f4['manhattan'])):
        if result_f4['manhattan']['y'][i] == norm_train.iloc[177:236]['y'][i+177]:
            x += 1
    acc = x / len(result_f4['manhattan'])
    accuracy.append(acc)

    # fold 5
    x = 0
    for i in range(len(result_f5['manhattan'])):
        if result_f5['manhattan']['y'][i] == norm_train.iloc[236:295]['y'][i+236]:
            x += 1
    acc = x / len(result_f5['manhattan'])
    accuracy.append(acc)
    return accuracy
```

In [10]:
```
# Result from Validation #1
print('Accuracy fold 1: ')
print(Validation(norm_train, k=1))
# Result from Validation #2
print('Accuracy fold 2: ')
print(Validation(norm_train, k=2))
# Result from Validation #3
print('Accuracy fold 3: ')
print(Validation(norm_train, k=3))
# Result from Validation #4
print('Accuracy fold 4: ')
print(Validation(norm_train, k=4))
# Result from Validation #5
print('Accuracy fold 5: ')
print(Validation(norm_train, k=5))
```

```
Accuracy fold 1:
[0.4576271186440678, 0.6949152542372882, 0.7457627118644068, 0.5423728813559322, 0.6610169491525424]
Accuracy fold 2:
[0.711864406779661, 0.8135593220338984, 0.7627118644067796, 0.6440677966101694, 0.7796610169491526]
Accuracy fold 3:
[0.711864406779661, 0.8135593220338984, 0.7627118644067796, 0.5932203389830508, 0.7457627118644068]
Accuracy fold 4:
[0.711864406779661, 0.8305084745762712, 0.7796610169491526, 0.5932203389830508, 0.7288135593220338]
Accuracy fold 5:
[0.711864406779661, 0.8305084745762712, 0.7796610169491526, 0.5932203389830508, 0.7457627118644068]
```



Testing

```
In [11]: # pemanggilan fungsi getKNN untuk mendapatkan hasil KNN dari manhattan
         KNN = getneighbour(norm_train, norm_test, k=2)

         KNN['manhattan']
```

```
Out[11]:
      id  y
   0  297 1
   1  298 1
   2  299 1
   3  300 0
   4  301 1
   5  302 0
   6  303 1
   7  304 1
   8  305 1
   9  306 1
```

```
In [12]: with pd.ExcelWriter('newtraintest.xlsx') as writer:
             KNN['manhattan'].to_excel(writer, sheet_name='Manhattan', index = False)
```

TEAM MEMBERS CONTRIBUTION

NAME	TASKS
NUR SABRINA SYAZA BINTI ZAHAR HISHAM (1301213666)	- Program Source Code - Report
NUR FASIHAH AYUNI BINTI MOHD YAHYA (1301213670)	- Program Source Code - Report
NURUL IMAN BINTI NORDIN (1301213669)	- Report - Readme.txt