IDC4251C Module 7 Project: kNN with PowerBI

K-nearest neighbors (kNN) algorithm is a type of supervised machine learning algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification problems in industry. The following two properties would define kNN well:

> (1) Lazy learning algorithm – kNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training during classification.
> (2) Non-parametric learning algorithm – kNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Normally kNN is implemented in programming languages like Python, R or MATLAB, but in this exercise we will implement it using DAX in Power BI.

You will start with a blank Power BI workbook and the data file "KNNData.xlsx".

Given the following labeled and categorized products as our training data:

	A	C	D	E
1	Product	Quantity	Price	Category
2	P1	10	7	A
3	P2	21	20	B
4	P3	33	29	C
5	P4	54	41	D
6	P5	45	36	C
7	P6	18	19	A
8	P7	18	20	A
9	P8	29	27	B
10	P9	31	28	C
11	P10	44	35	C
12	P11	21	20	B
13	P12	25	25	B
14	P13	27	26	B
15	P14	17	13	A
16	P15	11	11	A
17	P16	9	5	A
18	P17	45	36	D
19	P18	55	42	D
20	P19	38	29	C
21	P20	31	28	C
22				

And given the following uncategorized products as our testing data:

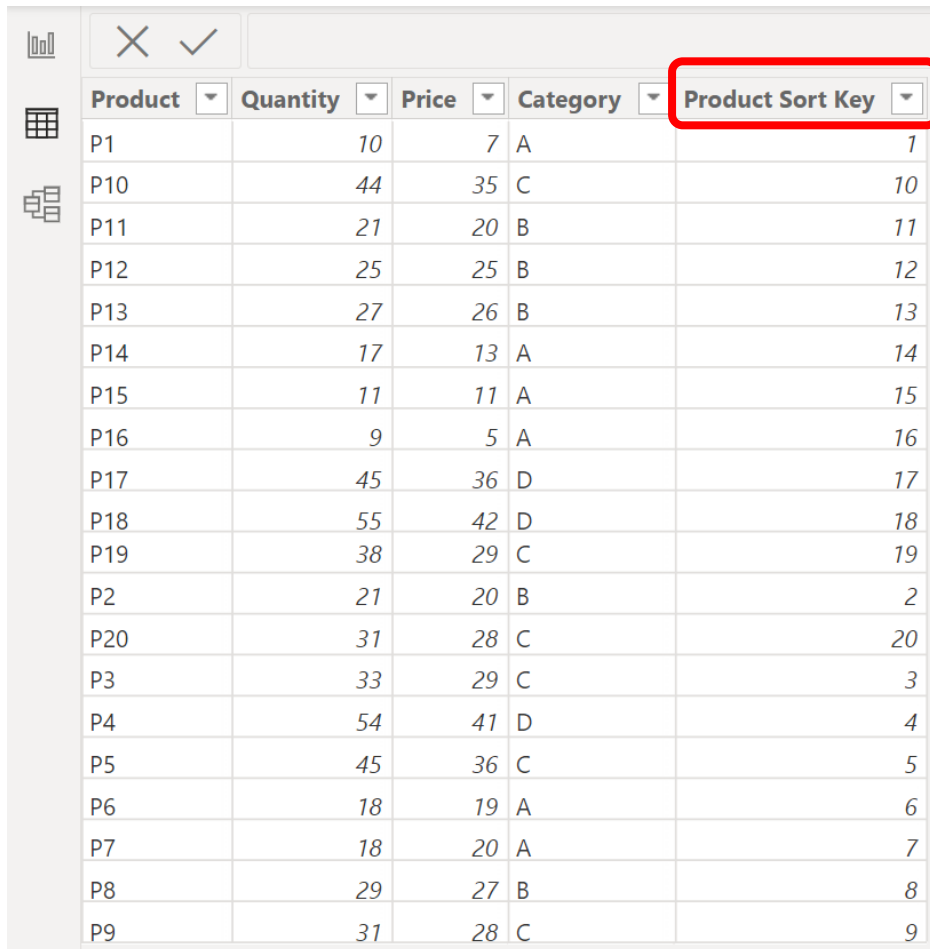|	A	B	C
1	_Product	_Quantity	_Price
2	P21	8	6
3	P22	12	10
4	P23	14	12
5	P24	14	12
6	P25	22	20
7	P26	35	27
8	P27	34	26
9	P28	41	31
10	P29	55	44
11	P30	56	45

Categorize the products in the testing data using the kNN algorithm by following these steps:

1. In a new PowerBI workbook, read the data from the attached Excel spreadsheet. Be sure to include both the Training and Testing sheets.

2. Try sorting the training data in various ways to see if you can determine any patterns in the relationships between the Categories and other data fields.

Notice that we cannot get a numerical sort of the Product column since the values are text. Use DAX to create a column which extracts the numbers from the text so we can sort the Products sequentially:

```
Product Sort Key = RIGHT([Product], LEN([Product]) - 1)
```

| Product	Quantity	Price	Category	Product Sort Key
P1	10	7	A	1
P10	44	35	C	10
P11	21	20	B	11
P12	25	25	B	12
P13	27	26	B	13
P14	17	13	A	14
P15	11	11	A	15
P16	9	5	A	16
P17	45	36	D	17
P18	55	42	D	18
P19	38	29	C	19
P2	21	20	B	2
P20	31	28	C	20
P3	33	29	C	3
P4	54	41	D	4
P5	45	36	C	5
P6	18	19	A	6
P7	18	20	A	7
P8	29	27	B	8
P9	31	28	C	9

3. Let's do a manual calculation with a simple DAX formula and add the result as another column to our training data.

```
Calculated Category =
        IF([Quantity] < 21, "A",
            IF([Quantity] < 31, "B",
                IF([Quantity] < 45, "C", "D")))
```

Product	Quantity	Price	Category	Product Sort Key	Calculated Category
P16	9	5	A	16	A
P1	10	7	A	1	A
P15	11	11	A	15	A
P14	17	13	A	14	A
P7	18	20	A	7	A
P6	18	19	A	6	A
P11	21	20	B	11	B
P2	21	20	B	2	B
P12	25	25	B	12	B
P13	27	26	B	13	B
P8	29	27	B	8	B
P20	31	28	C	20	C
P9	31	28	C	9	C
P3	33	29	C	3	C
P19	38	29	C	19	C
P10	44	35	C	10	C
P17	45	36	D	17	D
P5	45	36	C	5	D
P4	54	41	D	4	D
P18	55	42	D	18	D

The nested If structure in the formula is a little messy but it does the job, kind of. Sort by Quantity (or Price) -- then note that in the second highlighted section in the image above that the calculation for P5 is incorrect due to ambiguity in the categorization of P5 and P17, which are categorized differently with the same data. It will be interesting to see how kNN categorizes these items.

4. Create a scatter plot visualization and see if you can visually detect a pattern (it's pretty obvious, since we have something close to a linear relationship). If the markers were all the same color, would you be able to accurately circle them in groups by category? Notice that some of the horizontal distances between some points in the same category are relative far apart; again, it will be interesting to how kNN deals with this. Also notice that the P5 category C point (45, 36) is obscured by P17.

Category, Quantity and Price

Category ● A ● B ● C ● D
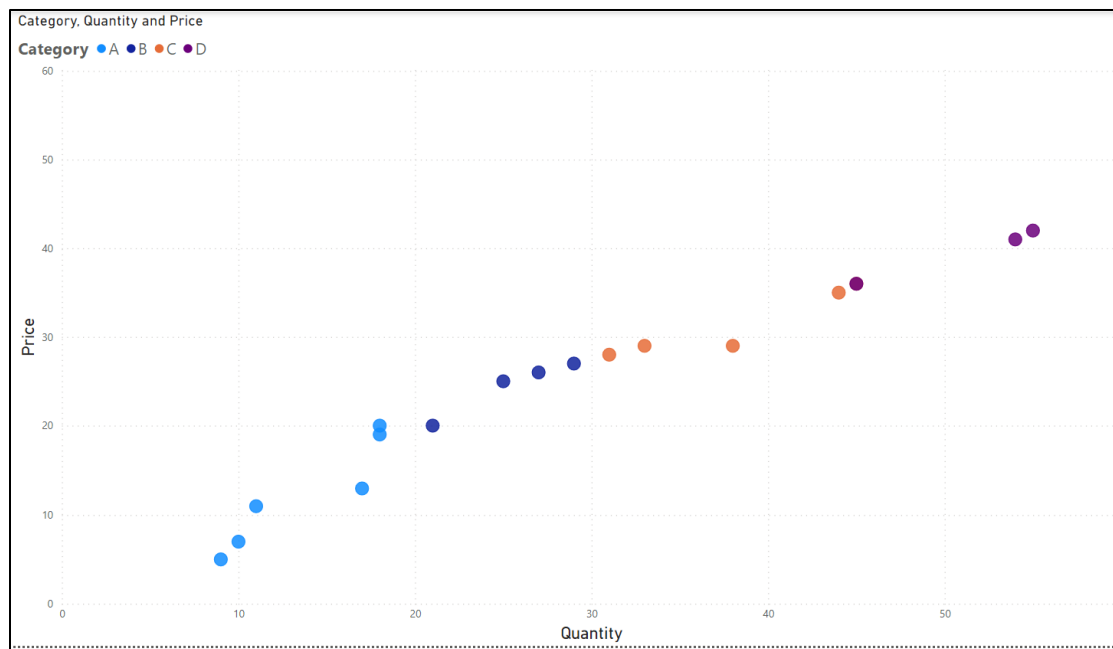
5. Now let's do our kNN analysis. Calculate the distance between each test data point and training set data point and take the (K = 5) values with the smallest distance as the neighboring points of the test point.

As we have read in our textbook, the value of K will have a great influence on the result. If the value is too small, it will increase the complexity of the model and make it prone to overfitting. If it is too large, the classification will be blurred (affected by noisy or irrelevant data).

In this exercise, we are using K = sqrt(N)=5, where N represents the number of samples in the training data set (it's actually somewhere between 4 and 5, but we need an integer). This is frequently the optimal value used in kNN and is a good starting point.

We will create a table (Modeling/New table) to perform these calculations.

Start by creating a table of the cartesian product of the Testing and Training tables; GENERATEALL takes each product in the Testing data and combines it with all products in the Training table.

We then add a column which represents the distance for each row. Here is the formula we are encoding (p and q in this formula are just variables for two points, they are not related to the name of the product or quantity columns).

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

The SQRT function calculates the square root of the sum of the X differences and Y differences squared.

We save the calculation in a variable T1 so we can use it in subsequent steps.

The CEILING function rounds (up) each calculated distance to the nearest 1/100.

```
KNNTable =
VAR T1 =
ADDCOLUMNS(
    GENERATEALL( 'Testing', 'Training' ),
    "Distance",
    CEILING(
        SQRT(
            ([Quantity] - [_Quantity])^2 + ([Price]-[_Price])^2
        ),
        0.01
    )
) RETURN T1
```

_Product	_Quantity	_Price	Product	Quantity	Price	Category	Product Sort Key	Calculated Category	Distance
P21	8	6	P1	10	7	A	1	A	2.24
P21	8	6	P10	44	35	C	10	C	46.23
P21	8	6	P11	21	20	B	11	B	19.11
P21	8	6	P12	25	25	B	12	B	25.5
P21	8	6	P13	27	26	B	13	B	27.59
P21	8	6	P14	17	13	A	14	A	11.41
P21	8	6	P15	11	11	A	15	A	5.84
P21	8	6	P16	9	5	A	16	A	1.42
P21	8	6	P17	45	36	D	17	D	47.64
P21	8	6	P18	55	42	D	18	D	59.21
P21	8	6	P19	38	29	C	19	C	37.81
P21	8	6	P2	21	20	B	2	B	19.11
P21	8	6	P20	31	28	C	20	C	31.83
P21	8	6	P3	33	29	C	3	C	33.98
P21	8	6	P4	54	41	D	4	D	57.81
P21	8	6	P5	45	36	C	5	D	47.64

We are not showing all 200 rows of this table, but you get the idea.

Now let's add another column to our table which represents an indicator of whether the distance is within our value for K of 5.

```
// For points with neighboring distances <= 5, set IskNN to 1
VAR T2 =
ADDCOLUMNS(
```

```
        T1,
        "IskNN",
        IF(
            RANKX(
                FILTER(T1, [_Product] = EARLIER([_Product]) ),
                [Distance], , ASC, Skip
            ) <= 5,
            1, 0
        )
)

RETURN T2
```

_Product	_Quantity	_Price	Product	Quantity	Price	Category	Product Sort Key	Calculated Category	Distance	IskNN
P21	8	6	P1	10	7	A	1	A	2.2	1
P21	8	6	P10	44	35	C	10	C	46.2	0
P21	8	6	P11	21	20	B	11	B	19.1	0
P21	8	6	P12	25	25	B	12	B	25.	0
P21	8	6	P13	27	26	B	13	B	27.5	0
P21	8	6	P14	17	13	A	14	A	11.4	1
P21	8	6	P15	11	11	A	15	A	5.8	1
P21	8	6	P16	9	5	A	16	A	1.4	1
P21	8	6	P17	45	36	D	17	D	47.6	0
P21	8	6	P18	55	42	D	18	D	59.2	0
P21	8	6	P19	38	29	C	19	C	37.8	0
P21	8	6	P2	21	20	B	2	B	19.1	0
P21	8	6	P20	31	28	C	20	C	31.8	0
P21	8	6	P3	33	29	C	3	C	33.9	0
P21	8	6	P4	54	41	D	4	D	57.8	0
P21	8	6	P5	45	36	C	5	D	47.6	0
P21	8	6	P6	18	19	A	6	A	16.4	1
P21	8	6	P7	18	20	A	7	A	17.2	0
P21	8	6	P8	29	27	B	8	B	29.	0
P21	8	6	P9	31	28	C	9	C	31.8	0

The RANKX function is allowing us to use an IF to insert a 1 if the test point falls within the K=5 proximity.

Finally, let's filter the table so we only see the points that meet our criteria:

```
RETURN
FILTER(
    T2,
    [IskNN] = 1
)
```

_Product	_Quantity	_Price	Product	Quantity	Price	Category	Product Sort Key	Calculated Category	Distance	IskNN
P21	8	6	P1	10	7	A	1	A	2.24	1
P21	8	6	P14	17	13	A	14	A	11.41	1
P21	8	6	P15	11	11	A	15	A	5.84	1
P21	8	6	P16	9	5	A	16	A	1.42	1
P21	8	6	P6	18	19	A	6	A	16.41	1
P22	12	10	P1	10	7	A	1	A	3.61	1
P22	12	10	P14	17	13	A	14	A	5.84	1
P22	12	10	P15	11	11	A	15	A	1.42	1
P22	12	10	P16	9	5	A	16	A	5.84	1
P22	12	10	P6	18	19	A	6	A	10.82	1
P23	14	12	P1	10	7	A	1	A	6.41	1
P23	14	12	P14	17	13	A	14	A	3.17	1
P23	14	12	P15	11	11	A	15	A	3.17	1
P23	14	12	P16	9	5	A	16	A	8.61	1
P23	14	12	P6	18	19	A	6	A	8.07	1
P24	14	12	P1	10	7	A	1	A	6.41	1
P24	14	12	P14	17	13	A	14	A	3.17	1
P24	14	12	P15	11	11	A	15	A	3.17	1

6. Now we want to count the number of adjacent points under each category to see which category has more adjacent points. Each record will be assigned to the category with more adjacent points. In the case of a tie we use the category with the smallest distance.

```
KNNResult =
ADDCOLUMNS(
    'KNNTable',
    "NumberOfNearbyPoints",
    COUNTX(
    FILTER(ALL('KNNTable'), [_Product] = EARLIER([_Product]) &&
        [Category] = EARLIER([Category])), [Distance])
)

Column =
VAR __min =
MINX(
    FILTER( 'KNNTable', [__Product] = EARLIER([__Product]) ),
    [Number of Nearby Points]
)
VAR __max =
MAXX(
    FILTER( 'KNNTable', [__Product] = EARLIER([__Product]) ),
    [Number of Nearby Points]
)
VAR x =
MINX(
    FILTER( 'KNNTable', [__Product] = EARLIER([__Product] ) ),
    [Distance]
)
RETURN
IF(
    __max = __min,
    CALCULATE(
```

```
                    MAX([Category]),
                    FILTER(
                        'KNNTable',
                        [Distance] = x && [__Product] = EARLIER( [__Product] )
                    )
                ),
                [Category]
            )

            // Cartesian product of the training data table
            // and the test data table.
```

4. Complete the classification of the test data and return the result table.

```
            Result =
            SUMMARIZE(
                'KNNTable',
                [__Product],
                "Maximum number of nearby points", MAX('KNNTable'[number of nearby
            points])
            )

            Category =
            LOOKUPVALUE(
                'KNNTable'[Column],
                'KNNTable'[__Product], 'Result'[__Product],
                'KNNTable'[number of nearby points], 'Result'[Maximum number of
            nearby points]
            )
```
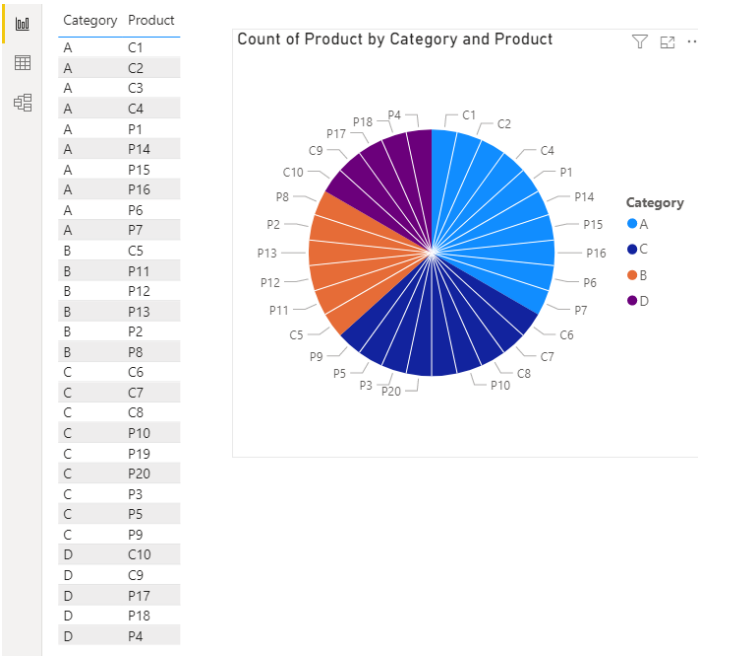
Result

Category	Product
A	C1
A	C2
A	C3
A	C4
A	P1
A	P14
A	P15
A	P16
A	P6
A	P7
B	C5
B	P11
B	P12
B	P13
B	P2
B	P8
C	C6
C	C7
C	C8
C	P10
C	P19
C	P20
C	P3
C	P5
C	P9
D	C10
D	C9
D	P17
D	P18
D	P4

Count of Product by Category and Product

Product	Price	Quantity	Category
P1	7	10	A
P14	13	17	A
P15	11	11	A
P16	5	9	A
P6	19	18	A
P7	20	18	A
P11	20	21	B
P12	25	25	B
P13	26	27	B
P2	20	21	B
P8	27	29	B
P10	35	44	C
P19	29	38	C
P20	28	31	C
P3	29	33	C
P5	36	45	C
P9	28	31	C
P17	36	45	D
P18	42	55	D
P4	41	54	D

__Product	__Price	__Quantity	Category
C1	6	8	A
C2	10	12	A
C3	12	14	A
C4	12	14	A
C5	20	22	B
C7	26	34	C
C6	27	35	C
C8	31	41	C
C9	44	55	D
C10	45	56	D

__Product	Maximum number of nearby points	Category
C1	5	A
C2	5	A
C3	5	A
C4	5	A
C5	3	B
C6	4	C
C7	4	C
C8	4	C
C9	3	D
C10	3	D