Datavarehus Øving 5

Alexander Fredheim
TDT4300 - Datavarehus og Datagruvedrift
NTNU

March 25, 2020

1 Datawarehousing

a)

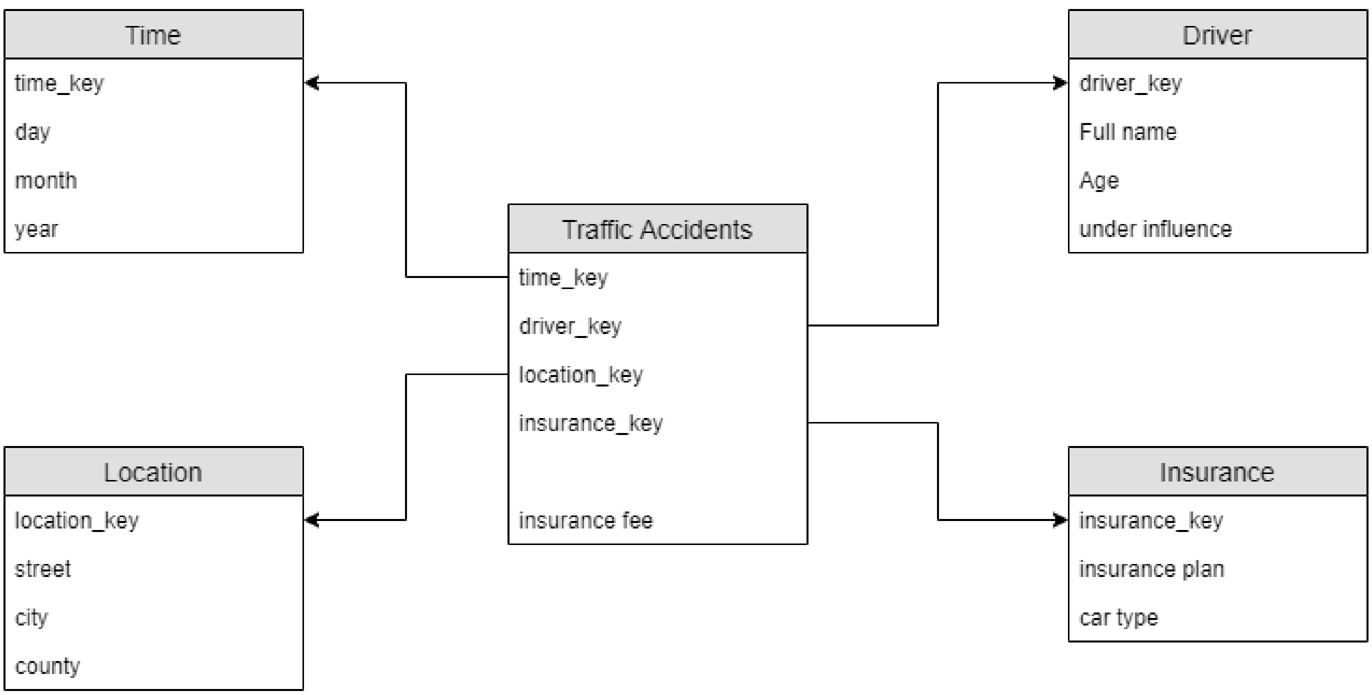

Figure 1: Snowflake scheme

Assumptions:

1. insurance fee is based on factors from both the Driver and Insurance dimensions

b)

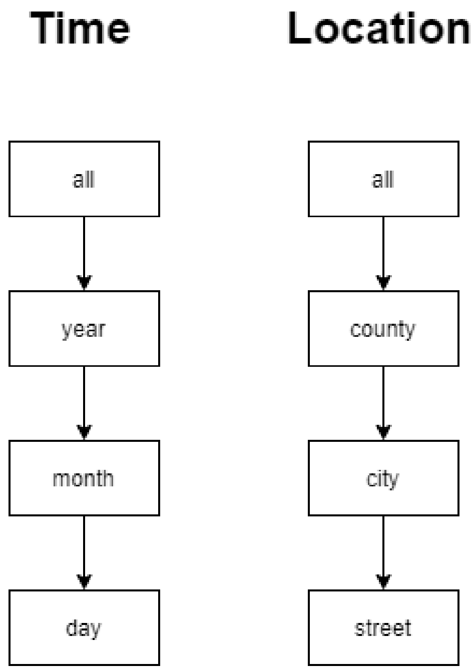

Figure 2: Hierachies

2 Association Rules

2.1 Finding frequent element sets

We are here going to generate association rules by using the Apriori algorithm, given minimum support = 0.5 and mimumum condifence = 0.8.
Our minimum support for our initial table is 2

1	A,B,C
2	A,C
3	A,D
4	B,E,F

Table 1: Market basket transactions

Itemset	**Supportcount**
{A}	3
{B}	2
{C}	2
{D}	1
{E}	1
{F}	1

Table 2: initial 1-element sets

Itemset	**Supportcount**
{A}	3
{B}	2
{C}	2

Table 3: frequent 1-element sets above minimum support (2)

Itemset	**Supportcount**
{A, B}	1
{A, C}	2
{B, C}	1

Table 4: 2-element sets generated by frequent 1-element sets

Itemset	**Supportcount**
{A, C}	2

Table 5: frequent 2-element sets above minimum support (2)

2.2 Generating association rules

From 2.1 we see that the {A, C} is the highest frequent element set we can generate. We look at the association rules we can generate from this set and remember that the minimum confidence is 0.8

Rules	**Confidence**
{A, C} =>{B}	0.5
{A, C} =>{D}	0
{A, C} =>{E}	0
{A, C} =>{F}	0

Table 6: Association rules generated from {A, C} and their confidence

We see in table 6 that all rules generated from {A, C} does not meet our confidence threshold. We can therefore prune all possible rules containing B, D, E F on the right hand side. This leaves us with zero association rules possible to generate given a minimum support of 0.5 and a minimum confidence of 0.8

3 Decision trees

Customer ID	**Age**	**Income**	**Student**	**Credit-worthiness**	**PC on Credit**
1	Young	High	No	Pass	No
2	Young	High	No	High	No
3	Middle	High	No	Pass	Yes
4	Old	Medium	No	Pass	Yes
5	Old	Low	No	Pass	Yes
6	Old	Low	Yes	High	No
7	Middle	Low	Yes	High	Yes
8	Young	Medium	No	Pass	No
9	Young	Low	Yes	Pass	Yes
10	Old	Medium	Yes	Pass	Yes
11	Young	Medium	Yes	High	Yes
12	Middle	Medium	No	High	Yes
13	Middle	High	Yes	Pass	Yes
14	Old	Medium	No	High	No
15	Middle	Medium	Yes	Pass	No
16	Middle	Medium	Yes	High	Yes
17	Young	Low	Yes	High	Yes
18	Old	High	No	Pass	No
19	Old	Low	No	High	No
20	Young	Medium	Yes	High	Yes

Table 7: Dataset for generating the decision tree

3.1 Gini index for the whole training set

```
credit_no = 8
credit_yes = 12
total = 20

gini_index = 1 - (credit_yes/total)**2 - (credit_no/total)**2
print(gini_index) #prints 0.48
```

Listing 1: code calculating gini index for the whole dataset

As we can see from the code, the gini index for the entire training set is 0.48

3.2 Gini index for each attribute

```
# List attributes by in order by customer ID and by trait
pc_on_credit = [0,0,1,1,1,0,1,0,1,1,1,1,1,0,0,1,1,0,0,1]
customerid = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
age = [0,0,1,2,2,2,1,0,0,2,0,1,1,2,1,1,0,2,2,0]
income = [2,2,2,1,0,0,0,1,0,1,1,1,2,1,1,1,0,2,0,1]
student = [0,0,0,0,0,1,1,0,1,1,1,0,1,0,1,1,1,0,0,1]
cw = [0,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,1,0,1,1]

# Small check
assert len(pc_on_credit) == len(customerid)
assert len(pc_on_credit) == len(age)
assert len(pc_on_credit) == len(income)
assert len(pc_on_credit) == len(student)
assert len(pc_on_credit) == len(cw)

def classify_attribute(classification, attribute, split_IDS=None):
    """
    takes each trait from the partitions of an attribute and assign each trait how many
    of each classifications are present
    :param classification:
    :param attribute:
    :return: dictionary for each trait of the attribute with counts of each classification
    """
    # Condition that makes this method work on a subset of the whole dataset
    if split_IDS is not None:
        new_classification = []
        new_attribute = []

        # Extract the desired row indexes
        for i in range(1,len(classification)+1):
            if i in split_IDS:
                new_classification.append(classification[i-1])
                new_attribute.append(attribute[i-1])

        # Reassign the variables containing the parameters
        classification = new_classification
        attribute = new_attribute

    attr_classifications = {}
    for class_, partition in zip(classification, attribute):
        if partition not in attr_classifications:
            attr_classifications[partition] = [0,0]
        if class_ == 0:
            attr_classifications[partition][0] += 1
        else:
            attr_classifications[partition][1] += 1
    return attr_classifications

# partition attributes so that we can calculate gini score for each partition and gini index
customer_traits = classify_attribute(pc_on_credit, customerid)
```

```python
age_traits = classify_attribute(pc_on_credit, age)
income_traits = classify_attribute(pc_on_credit, income)
student_traits = classify_attribute(pc_on_credit, student)
cw_traits = classify_attribute(pc_on_credit, cw)

# Calculate gini for a trait, a partition of an attribute
def gini(trait):
    gini_score = 1
    for count in trait:
        gini_score -= (count/sum(trait))**2
    return gini_score

# Calculate gini index, also called weighted average gini
def gini_index(traits, total_size):
    gini_index = 0
    for trait in traits.values():
        gini_index += sum(trait)/total_size * gini(trait)
    return gini_index

# Assign gini index to variables
gini_customer_id = gini_index(customer_traits, len(pc_on_credit))
gini_age = gini_index(age_traits, len(pc_on_credit))
gini_income = gini_index(income_traits, len(pc_on_credit))
gini_student = gini_index(student_traits, len(pc_on_credit))
gini_cw = gini_index(cw_traits, len(pc_on_credit))

# Print gini index for every attribute
print(gini_customer_id)  # Prints 0.0
print(gini_age)  # Prints 0.426
print(gini_income)  # Prints 0.453
print(gini_student) # Prints 0.400
print(gini_cw) # Prints 0.48
```

Listing 2: code calculating gini index for all attributes

The attributes and their corresponding gini index are listed below

Attribute	**Gini index**
Customer ID	0.0
Age	0.426
Income	0.453
Student	0.400
Credit-Worthiness	0.48

Table 8: Attributes and their corresponding gini index

3.3 Splitting on attributes

In the following cases we must decide which attributes to split in what order and whether the customer should get a discount or not

3.3.1 Case 1

Customer 21: A young student with medium income and "high" creditworthiness.

First we want to split on student, as this has the lowest gini index that gives us a reasonable amount of splits

Customer ID	Age	Income	Student	Credit-worthiness	PC on Credit
6	Old	Low	Yes	High	No
7	Middle	Low	Yes	High	Yes
9	Young	Low	Yes	Pass	Yes
10	Old	Medium	Yes	Pass	Yes
11	Young	Medium	Yes	High	Yes
13	Middle	High	Yes	Pass	Yes
15	Middle	Medium	Yes	Pass	No
16	Middle	Medium	Yes	High	Yes
17	Young	Low	Yes	High	Yes
20	Young	Medium	Yes	High	Yes

Table 9: Dataset after splitting on student and selecting yes

The column we are interested in is "PC on Credit", and we can see here that we intuitively gain a lot of information about if the given customer should get the PC on credit or not.

With the given split shown in table **??** we can now recalculate gini index for Age, Income and Credit-worthiness

```
# The customer IDs after splitting on student and selecting "yes"
customerid_stud_split_yes = [6, 7, 9, 10, 11, 13, 15, 16, 17, 20]

# Partition attributes based on earlier split
age_traits2 = classify_attribute(pc_on_credit, age, customerid_stud_split_yes)
income_traits2 = classify_attribute(pc_on_credit, income, customerid_stud_split_yes)
cw_traits2 = classify_attribute(pc_on_credit, cw, customerid_stud_split_yes)

# Assign gini index to variables, after splitting on student and selecting "yes"
gini_age2 = gini_index(age_traits2, len(customerid_stud_split_yes))
gini_income2 = gini_index(income_traits2, len(customerid_stud_split_yes))
gini_cw2 = gini_index(cw_traits2, len(customerid_stud_split_yes))

# Print gini index for the three remaining possible splits
print(gini_age2)  # Prints 0.25
print(gini_income2)  # Prints 0.310
print(gini_cw2)  # Prints 0.316
```

Listing 3: code calculating gini index for the attributes Age, Income and Credit-Worthiness after split on student and selecting yes

Recalculating the gini index after splitting on Student, gives Age as the next best alternative for splitting. This gives the following table:

Customer ID	**Age**	**Income**	**Student**	**Credit-worthiness**	**PC on Credit**
9	Young	Low	Yes	Pass	Yes
11	Young	Medium	Yes	High	Yes
17	Young	Low	Yes	High	Yes
20	Young	Medium	Yes	High	Yes

Table 10: Dataset after splitting on student, splitting on Age and selecting "Young"

Here we can see that every customer with these traits and attributes gets a PC on credit. Or in other terms, all other existing customers in this split has gotten a PC on credit. Therefore, customer 21 should also get a PC on credit

3.3.2 Case 2

Customer 22: A young non-student with low income and "pass" creditworthiness.

With the complete data set (not considering customer 21) as our initial starting point, we start our split at student again, and select "non-student", this yields the following table:

Customer ID	**Age**	**Income**	**Student**	**Credit-worthiness**	**PC on Credit**
1	Young	High	No	Pass	No
2	Young	High	No	High	No
3	Middle	High	No	Pass	Yes
4	Old	Medium	No	Pass	Yes
5	Old	Low	No	Pass	Yes
8	Young	Medium	No	Pass	No
12	Middle	Medium	No	High	Yes
14	Old	Medium	No	High	No
18	Old	High	No	Pass	No
19	Old	Low	No	High	No

Table 11: Dataset after splitting on student and selecting no

like in 3.3.1 we will now calculate gini index for the remaing splits

```python
# The customer IDs after splitting on student and selecting "no"
customerid_stud_split_no = [1, 2, 3, 4, 5, 8, 12, 14, 18, 19]

# Partition attributes based on earlier split
age_traits3 = classify_attribute(pc_on_credit, age, customerid_stud_split_no)
income_traits3 = classify_attribute(pc_on_credit, income, customerid_stud_split_no)
cw_traits3 = classify_attribute(pc_on_credit, cw, customerid_stud_split_no)

# Assign gini index to variables, after splitting on student and selecting "no"
gini_age3 = gini_index(age_traits3, len(customerid_stud_split_no))
gini_income3 = gini_index(income_traits3, len(customerid_stud_split_no))
gini_cw3 = gini_index(cw_traits3, len(customerid_stud_split_no))

# Print gini index for the three remaining possible splits
print(gini_age3)  # Prints 0.24
print(gini_income3)  # Prints 0.45
print(gini_cw3)  # Prints 0.45
```

Listing 4: code calculating gini index for the attributes Age, Income and Credit-Worthiness after split on student and selecting yes

We again get Age as the best possible split, which now yields the the following table:

Customer ID	Age	Income	Student	Credit-worthiness	PC on Credit
1	Young	High	No	Pass	No
2	Young	High	No	High	No
8	Young	Medium	No	Pass	No

Table 12: Dataset after splitting on student, splitting on Age and selecting ”young”

Here we can see that every customer with these traits and attributes **does not** get a PC on credit. In other terms, all other existing customers in this split has been denied a PC on credit. Therefore, customer 22 should not get a PC on credit

4 Data types

Some definitions:

Qualitative:
Nominal means "relating to names." The values of a nominal attribute are symbols or names of things. Each value represents some kind of category, code, or state, and so nominal attributes are also referred to as categorical. The values do not have any meaningful order.

An ordinal attribute is an attribute with possible values that have a meaningful order or ranking among them, but the magnitude between successive values is not known.

Quantitative:
A numeric attribute is quantitative; that is, it is a measurable quantity, represented in integer or real values. Numeric attributes can be interval-scaled or ratio-scaled.

Interval-scaled attributes are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, 0, or negative. Thus, in addition to providing a ranking of values, such attributes allow us to compare and quantify the difference between values.

A ratio-scaled attribute is a numeric attribute with an inherent zero-point. That is, if a measurement is ratio-scaled, we can speak of a value as being a multiple (or ratio) of another value. In addition, the values are ordered, and we can also compute the difference between values, as well as the mean, median, and mode.

a) attribute classifications: binary, qualitative, ordinal

   Reasoning: Time can ONLY be EITHER AM or PM, under the assumption of a 12 hour time system. I argue that this attribute is ordinal as there is correlation between AM and PM.

b) attribute classifications: continuous, quantitative, ratio

   Reasoning: Continuous because you in theory can measure infinitely many small nuances of light. Ratio because light meters use a fixed size scale, often 1-10 to describe darkness/brigthness of, say, a picture

c) attribute classifications: discrete, quantitative, ordinal

   Reasoning: Discrete, as we will get some fixed amount of discrete, concretely distinguishable, descriptions. Ordinal because you most likely will be able to nuance the magnitude of light based on the descriptions

d) attribute classifications: continuous, quantitative, ratio

   Reasoning: Continuous because you in theory can measure infinitely many small nuances between 0 and 360 degrees. Ratio because we have a fixed size scale, where you could say that 30 degrees is twice as much as 15 degrees, clear correlation between measurements

e) attribute classifications: discrete, qualitative, ordinal

   Reasoning: Discrete, as we have three discrete values. Ordinal because the magnitude between the values is common knowledge: gold >silver >bronze

f) attribute classifications: continuous, quantitative, ratio

   Reasoning: Continuous because you in theory can measure infinitely many small nuances of height above sea level. Ratio because we have a fixed size ratio, where you could say that 20 meters above sea level is twice as much as 10 meters above sea level

g) attribute classifications: discrete, quantitative, ratio

   Reasoning: Discrete because we have a discrete amount of people in a hospital, a person in this context is atomic, you cannot count "half" a person. Ratio because we have a fixed size ratio, where you could say that 30 people is twice as many as 15 people.

h) attribute classifications: discrete, qualitative, nominal

   Reasoning: Discrete, as we will get some fixed amount of discrete, concretely distinguishable, ISBN numbers. Nominal because there is no meaningful ranking between different ISBN numbers.

i) attribute classifications: discrete, qualitative, ordinal

   Reasoning: Discrete, as we have three discrete values. Ordinal as there is a meaningful ordering to the values, ranking from least able to pass light, to most able.

j) attribute classifications: discrete, qualitative, ordinal

   Reasoning: Discrete, as we will have a finite amount of discrete ranks. Ordinal as there is a meaningful ordering to the values, often ranking in a power-position type hierachy.

k) attribute classifications: continuous, quantitative, ratio

   Reasoning: Continuous because you in theory can measure infinitely many small nuances of distance from the center of campus. Ratio because something that is 40 meters from campus is twice as far away as something that is 20 meters from campus.

l) attribute classifications: continuous, quantitative, ratio

   Reasoning: Continuous because you in theory can measure infinitely many small nuances of grams per cubic centimeter. Ratio because something that has 60 grams per cubic centimeter is twice as dense as something that has 30 grams per cubic centimeter.

m) attribute classifications: discrete, qualitative, nominal

   Reasoning: Discrete, as we will have a finite amount of coat check number. Nominal as the number are merely identifiers, and don't really have any meaning between them.

5 Auto correlation

Question: Which of the following quantities is likely to show more temporal autocorrelation: daily rainfall or daily temperature? Why?

Daily temperature will have a higher autocorrelation because the change of temperature in degrees on a day to day basis is much lower than the change of rainfall in mm.

6 Noise and Outliers

a) Outliers can be interesting if it is a valid datapoint (i.e not noise). A common example is unusual credit card activity (outlier data) indicating possibility of credit card fraud. Noise is somewhat defined as data that does not fit the model or corrupt data, and is therefore often not desirable

b) Yes, if the data is corrupt enough or unrelated data somehow makes it into our analysis it can be tagged as an outlier, even tho it in reality is just noise.

c) No, sometimes the noise is just smaller deviations from the model, which is not enough to be seen as an outlier.

d) No, sometimes valid data really deviates from the rest of the data set, then it is an outlier but not noise.

e) Yes, with enough noise typical value can seem unusual due to a skewed dataset. In the same manner some unusual value can seem typical.

7 Similarity Measures

I use python to calculate the similarities, but implement the functions manually to make it clear how to calculate each similarity

```python
# Calculates cosine similarity between two vectors, given as two lists in input
def cosine_similarity(vector1, vector2):
    dot_product = 0
    square_A = 0
    square_B = 0
    # Calculate dot product and square values
    for val1, val2 in zip(vector1,vector2):
        dot_product += val1*val2
        square_A += val1**2
        square_B += val2**2
    square_A = math.sqrt(square_A)
    square_B = math.sqrt(square_B)
    return dot_product / (square_A*square_B)


# Calculates standard deviation, given the mean (average)
def standard_deviation(vector, mean):
    sd = 0
    for val in vector:
        sd += (val - mean)**2
```

```
    return math.sqrt(sd/(len(vector)-1))


# Calculates correlation between two vectors, given as two lists in input
def correlation(vector1, vector2):
    avg_x = sum(vector1)/len(vector1)
    avg_y = sum(vector2)/len(vector2)
    sd_x = standard_deviation(vector1, avg_x)
    sd_y = standard_deviation(vector2, avg_y)


    corr = 0
    for x_i, y_i in zip(vector1, vector2):
        z_xi = (x_i - avg_x)
        z_yi = (y_i - avg_y)
        corr += z_xi * z_yi

    return corr/((len(vector1)-1)*sd_x*sd_y)


# Returns euclidean distance
def euclidean_distance(vector1, vector2):
    dist = 0
    for x_i, y_i in zip(vector1, vector2):
        dist += (x_i - y_i)**2
    return math.sqrt(dist)


# Returns Jaccard coefficient (inputs must be binary)
def jaccard_coeff(vector1, vector2):
    one_matches = 0
    non_zeros = 0
    for x_i, y_i in zip(vector1, vector2):
        if x_i and y_i == 1:
            one_matches += 1
            non_zeros += 1
        elif x_i and y_i == 0:
            continue
        else:
            non_zeros += 1
    return one_matches/non_zeros


# Defining vectors
a_x, a_y = [1,1,1,1], [2,2,2,2]
b_x, b_y = [0,1,0,1], [1,0,1,0]
c_x, c_y = [0,-1,0,1], [1,0,-1,0]
d_x, d_y = [1,1,0,1,0,1], [1,1,1,0,0,1]
e_x, e_y = [2,-1,0,2,0,-3], [-1,1,-1,0,0,-1]

# a)
cosine_sim_a = cosine_similarity(a_x, a_y)
# corr_a = correlation(a_x, a_y)
euclid_a = euclidean_distance(a_x, a_y)

# passes check
assert cosine_sim_a == np.dot(a_x, a_y)/(np.linalg.norm(a_x)*np.linalg.norm(a_y))

print("a)")
print(cosine_sim_a)  # Prints 1.0
# correlation for a) is undefined
print(euclid_a)  # Prints 2.0
print("----------")

# b)
cosine_sim_b = cosine_similarity(b_x, b_y)
corr_b = correlation(b_x, b_y)
euclid_b = euclidean_distance(b_x, b_y)
```

```
jaccard_b = jaccard_coeff(b_x, b_y)

print("b)")
print(cosine_sim_b)  # Prints 0.0
print(corr_b)  # Prints -1.0
print(euclid_b)  # Prints 2.0
print(jaccard_b)  # Prints 0.0
print("----------")

# c)
cosine_sim_c = cosine_similarity(c_x, c_y)
corr_c = correlation(c_x, c_y)
euclid_c = euclidean_distance(c_x, c_y)

print("c)")
print(cosine_sim_c)  # Prints 0.0
print(corr_c)  # Prints 0.0
print(euclid_c)  # Prints 2.0
print("---------")

# d)
cosine_sim_d = cosine_similarity(d_x, d_y)
corr_d = correlation(d_x, d_y)
jaccard_d = jaccard_coeff(d_x, d_y)

print("d)")
print(cosine_sim_d)  # Prints 0.75
print(corr_d)  # Prints 0.25
print(jaccard_d)  # Prints 0.6
print("-------------")

# e)
cosine_sim_e = cosine_similarity(e_x, e_y)
corr_e = correlation(e_x, e_y)

print("e)")
print(cosine_sim_e)  # Prints 0.0
print(corr_e)  # Prints -5.73316704659901e-17
```
Listing 5: code calculating the different similarity measures required in the task

Summarized, here are the similarities for each task

Similarity type	**Similarity**
a)	
Cosine	1.0
Correlation	undefined
Euclidean	2.0
b)	
Cosine	0.0
Correlation	-1.0
Euclidean	2.0
Jaccard	0.0
c)	
Cosine	0.0
Correlation	0.0
Euclidean	2.0
d)	
Cosine	0.75
Correlation	0.25
Jaccard	0.6
d)	
Cosine	0.0
Correlation	0.0

Table 13: answers to section 7