Informatics 1: Data & Analysis
Session 2014/2015, Semester 2

Exam Report

This is a report on the end-of-year examination for the course *Informatics 1: Data & Analysis*, held on Thursday 7 May 2015. It reviews the exam itself, with comments on possible solutions and the answers submitted by the students who took the exam. Please note the following:

- This is not a set of "model" answers. It does contain solutions, which can be used to check your own answers; but there are also discussions of different possible answers, key points, possible errors, and comments on the ways people approached each question on the day.
- Not all the questions have a single "right" answer. There can be multiple correct ways to write a database query, explain a concept, or construct an example. This report includes some variants on answers, but still cannot cover every possible correct alternative.
- Studying past exam questions is one way to learn more about a subject, but it is quite limited and not enough on its own. Even when exams routinely follow a fixed structure, the questions change and will include new content not used in any previous paper. Successful performance depends on a good understanding of the material in the course.

The exam consisted of three questions, each with several subquestions. The rest of this report is a brief summary of performance on the exam and then the full text of each question followed by comments on possible solutions and the answers given by students.

Where you find errors in these notes, please send them to me at Ian.Stark@ed.ac.uk

Ian Stark
2015-07-07

Summary

Performance on the exam this year was excellent: 263 candidates submitted scripts to be marked, with a very high proportion passing and a substantial number achieving Grade A, scoring 70% or higher. For each question there were multiple student answers — not all the same — achieving full marks.

Some of the more challenging sections in the exam involved taking in a moderately complex situation and then using that understanding to construct a formal model. In 1(b), for example, the model was an ER diagram. Many students did this really well and very cleanly, with an ER diagram that correctly used a weak entity to capture the constraints on rooms. In contrast, many models for the publisher's relational database of 2(d) were crucially flawed by having the reference from one table to another the wrong way around.

Answers involving mathematical calculation were done very well by almost everyone, especially the cosine similarity computation of 3(d). Responses to "bookwork" questions asking for standard definitions or examples, such as 3(a), were much less consistent. Even though this "bookwork" depends mainly on recall of fixed material, higher-scoring answers were often also those showing an in-depth understanding of the topic.

The following pages expand on this in a detailed breakdown of responses to each question.

Question 1 [*This question is worth a total of 30 marks.*]

For a large database system with multiple simultaneous users, it is often considered important for the implementation to guarantee the *ACID* properties for every database transaction. These can help ensure reliable and robust service.

(a) Each of the letters A, C, I and D stands for a different property. For each property:

(i) Give its name;

(ii) Explain in a sentence or two what it means for a transaction.

[*10 marks*]

(b) A student accommodation service looks after several residential halls, and wishes to integrate management of these into a single database. One small part of this concerns recording the allocation of students to rooms. You have the following information, arising from an initial requirements analysis.

- Each student has a universal username (UUN) that can be used to identify them in the database. This is a unique 8-character alphanumeric code issued by the University.
- Other than that, the database must record every student's name and their year of study.
- Students are allocated to rooms. Each room has a fixed capacity — usually one or two, indicating that the room is shared.
- No student can be allocated to more than one room; but some students in the system might not yet have any room assigned.
- There are several halls, each with its own name and unique Building ID.
- Each room belongs to one hall, and is identified within that hall by its number. There is no coordination in room numbering between different halls: in particular, two rooms in different halls might have the same number.

Draw an entity-relationship (ER) diagram that represents this information. Make sure that you capture the constraints described on the relationships involved, and designate appropriate primary keys for all entities.

[*20 marks*]

Notes on Question 1

(a) **A: Atomicity** All-or-nothing: a transaction either runs to completion, or fails and leaves the database unchanged.

This may involve a *rollback* mechanism to undo a partially-complete transaction.

C: Consistency Applying a transaction in a valid state of the database will always give a valid result state.

Where there are constraints to be maintained in a database, then this may require rolling back a transaction if it breaks one of these.

I: Isolation Concurrent transactions have the same effect as sequential ones: the outcome is as if they were done in order.

Transactions may, in fact, run at the same time: but should never see each other's intermediate state.

D: Durability Once a transaction is committed, it will not be rolled back.

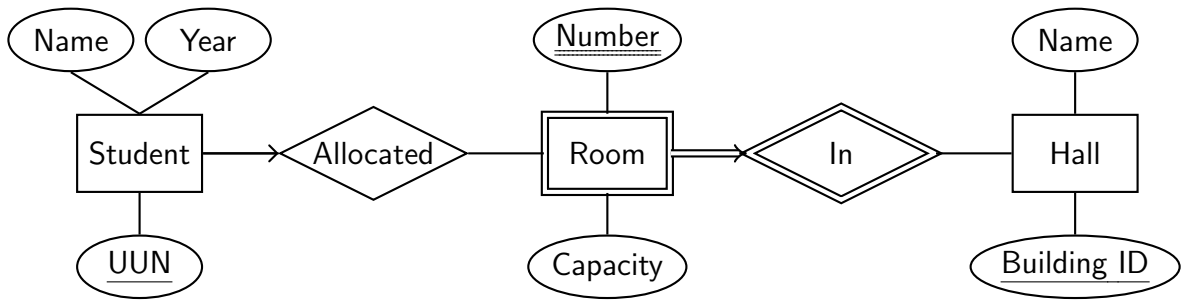
This can require work at many levels, from non-volatile memory and uninterruptible power supplies to distributed commit protocols.

In each case the first sentence alone is a sufficient description.

This topic hadn't appeared in any previous exam question. While that's not unexpected it did on this occasion cause problems for a number of students. Nonetheless, there were very many good answers: ranging from perfect recitation of the ACID properties to several strong part-answers based on a good general understanding of database requirements.

The ACID properties are a key benchmark for assessing database systems — not just with SQL, but also for the more recent NoSQL databases where it's important to know exactly which properties are still guaranteed, and which are being relaxed in a tradeoff with performance or scale.

(b) The ER diagram below represents the scenario described.



All the constraints mentioned in the requirements are captured here.

- The student UUN serves as primary key for the Student entity.
- Student has a key constraint in the Allocated relationship, represented by an arrowhead, indicating that each student can be allocated to at most one room.
- There is no participation constraint on Student (and hence no double line) as the requirements explicitly mention that some students may be recorded in the system but with no specific room assignment.

- Room is a *weak entity*, as the attributes provided, Number and Capacity, may not be enough to uniquely identify a room among all those in different halls. However, In provides an *identifying relationship* for each room, with the relevant Hall serving as *identifying owner*.

  This is shown in the ER diagram by the double outline on Room and In and the double underline on the room Number attribute — this Number and the related Building ID of the relevant Hall make up a composite key for the Room entity.

  The arrowhead and double line joining entity Room to the relationship In show a key constraint and a total participation constraint, respectively: each room must be in exactly one hall. This is essential for the Hall entity to serve as identifying owner for the weak Room entity.

- The Building entity has two attributes, with the unique Building ID chosen as primary key. Note that both Building and Student have a Name attribute, but there is no conflict in this.

Answers to this part of the question were generally very good, with particular attention to the weak entity Room and a small amount of legitimate variation. Even so the following errors each appeared a number of times.

- Not including a Capacity attribute, but instead setting up subclass entities for single and shared rooms. Unfortunately, that doesn't distinguish between shared rooms accommodating two, three or more students — so the diagram will still need a Capacity attribute to record this.
- Misplaced arrows. Arrows always point from an entity to a relationship, so the arrowhead is always at a relationship. See, for example, the key constraint that each student may be allocated to at most one room: this has an arrowhead on the line from entity Student to relationship Allocated; with nothing on the line going into Room.
- Omitting the identifying relationship. Whenever there is a weak entity shown by a double outline, like Room here, there must be an identifying relationship also shown by a double outline. Some answers missed this out.
- Wrong positioning of Number attribute. The room number is an attribute of the room, not of the identifying relationship.
- Unrequested additional constraints. Some answers included a total participation constraint as a double line from Hall to In. This would require that every hall has at least one room recorded in it. Although that's a plausible extra constraint on the system, it isn't in the specification provided. Moreover, it might not always hold: for example, when a hall undergoes major maintenance it could temporarily have no usable rooms recorded.

Question 2 [*This question is worth a total of 40 marks.*]

The following XML document captures some information about a book publisher's catalogue: in this case, the non-existent publisher *Animal Press*.

```
<?xml version="1.0"?>
<!DOCTYPE publisher SYSTEM "publisher.dtd">
<publisher>
    <name>Animal Press</name>
    <imprint>
        <title>Tapir Tomes</title>
        <tag>TT</tag>
        <book code="717899" type="hardback">
            <title>
                Beyond the Trees
            </title>
        </book>
        <book code="833908">
            <title>
                Never Before
            </title>
        </book>
    </imprint>
</publisher>
```

A full catalogue would list many different *imprints* of the publisher — different brand names they use to sell books — with for each imprint a short unique *tag* and a list of many books. Every book has a unique code, and is either *hardback* or *paperback*, with the default being paperback if not specified.

(a) Draw the tree of the *XPath data model* for this XML document [*10 marks*]

(b) Write out a DTD which describes this document and any other similar publisher's catalogue, suitable for the "publisher.dtd" file referenced. [*10 marks*]

(c) Write XPath expressions to obtain the following information from such a document.

(i) A list of all the imprint tags.

(ii) The title of the book with code 823095.

(iii) The title of every imprint that includes at least one hardback book.

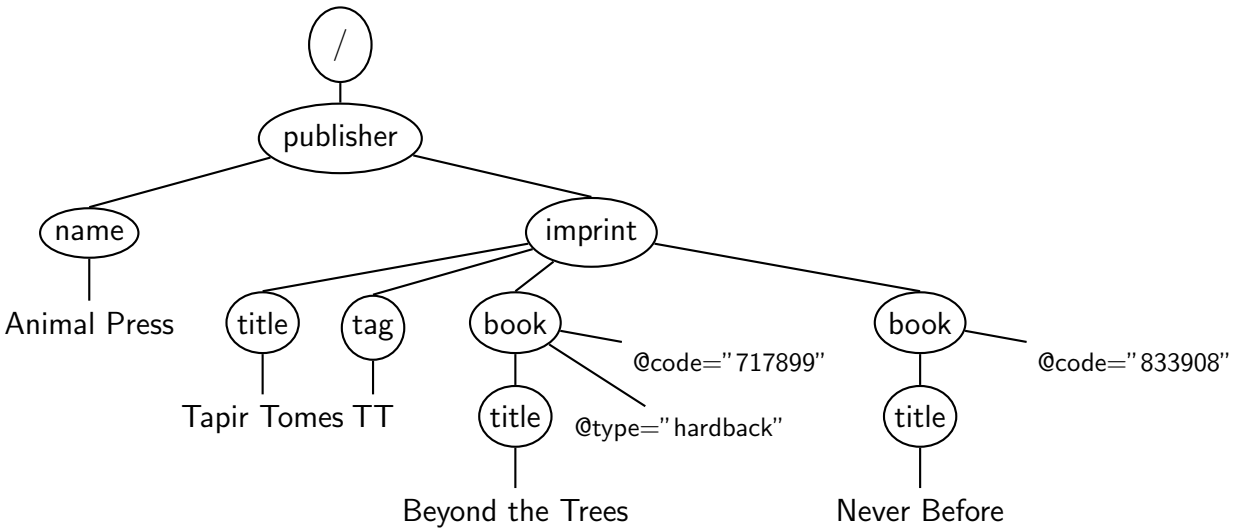
[*6 marks*]

(d) The *Animal Press* themselves keep this information in a relational database with two linked tables: Imprint and Book. Write suitable schemas for these tables in the SQL Data Declaration Language. [*8 marks*]

(e) Based on your schemas, write SQL queries to find out the information required for each item in part (c) above. [*6 marks*]

Notes on Question 2

(a) The following tree is the XPath data model of the XML file given.



The left-to-right ordering of element nodes matters: for example, the "name" node must be to the left of the "imprint" node. Ordering of attribute nodes like "@type" and "@code" is arbitrary. The important thing here is not that ordered or unordered is "better", but that you need to be able to work correctly with each as required.

Notice that there is a single root node "/", above publisher. Text in the leaf nodes should not be in quotation marks.

Remember that there is a 1-1 translation between the textual syntax of XML and the tree model of XPath. This means that the tree should not add, remove or change anything that is in the XML text. Several students incorrectly made their own modifications — for example, changing the "title" node to an "imprint−title" node, putting in "..." for further nodes, or adding an extra attribute to indicate that "Never Before" is a paperback book. All of these are errors, as the resulting tree does not match the XML text given.

This also happens sometimes with questions posed the other way round: from a tree, give the XML text. Again, it's an error to change the names of nodes or alter the content.

(b) The following is a suitable contents for the "publisher.dtd" file.

```
<!ELEMENT publisher (name,imprint+) >
<!ELEMENT imprint (title,tag,book+) >
<!ELEMENT book (title) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT tag (#PCDATA) >
<!ATTLIST book code CDATA #REQUIRED>
<!ATTLIST book type (hardback|paperback) "paperback" >
```

The ordering of lines doesn't matter: the DTD language is purely declarative, and each individual declaration can refer freely to any other.

Notice that there is only one declaration of the "title" element, even though this is used in two different settings (as the title of an imprint, and of individual books). Repeating the declaration is an error.

There are some legitimate variations on these DTD declarations: for example, using "(name,imprint∗)" to indicate that a publisher element might have zero or more imprints rather than one or more. However, using just "(name,imprint)" is an error, as that limits to structure to only one imprint — the question specifically states that a publisher will have many different imprints.

Several students wrote someting like "(title |tag|book+)" for the imprint element. Using a vertical bar "|" here is entirely wrong, as it denotes alternatives (either a title *or* a tag *or* a bool+). The correct connective is the comma "," to denote each item is required in order.

The **ATTLIST** declaration for the type of a book uses an explicit choice to indicate that every book is either hardback or paperback; and that if no value is provided then the default is paperback.

It's possible to put both book attributes into a single **ATTLIST** line:

<!**ATTLIST** book code **CDATA** #**REQUIRED** type (hardback|paperback) "paperback">

(c) (i) Here are three possible variations on an answer:

- //tag/text()
- /publisher/imprint/tag/text()
- //imprint/tag/text()

Note that the final text() test is necessary to pick out the text itself at the leaf of the tree, rather than the element node that is parent to it.

(ii) A couple of possible answers:

- //book[@code="823095"]/title/text()
- //title [../ @code="823095"]/text()

The first of these picks out books with the right code, and then finds the title text; the second gathers titles, and checks the code of their parent book node. Both give the same result.

(iii) Here are a range of possible solutions.

- //imprint[.//@type="hardback"]/title/text()
- //imprint[book/@type="hardback"]/title/text()
  These two pick out imprints containing a book that is a hardback, and then return the imprint title.
- //book[@type="hardback"]/../title/text()
- //book[@type="hardback"]/ancestor::imprint/title/text()
  These two solutions select all hardback books, then navigate up the tree to find out which imprint they come from, and then its title.
- //imprint[count(book[@type="hardback"])>0]/title/text()
  This solution came from a student, and was completely unexpected. I didn't cover the XPath "count()" function in the course, but it's entirely legitimate and, if used correctly, makes for a reasonable answer to the question. Here it appears in a predicate to identify those imprint nodes which contain more than zero book nodes with attribute type of "hardback".

(d) The table declarations below capture the information held in the XML document.

```
create table Imprint (
    title  varchar(60) not null,
    tag    varchar(6),
    primary key (tag)
)

create table Book (
    title      varchar(120) not null,
    code       varchar(6),
    type       varchar(10),
    imprint    varchar(6) not null,
    primary key (code),
    foreign key (imprint) references Imprint(tag)
)
```

The exact size of each varchar is not too important. Fields from the primary key are always **not null**, so don't need that declared; however, it's also fine to declare them **not null** explicitly. The foreign key imprint of Book should be **not null**, as each book must belong to some imprint. I've also made the title of both books and imprints **not null**.

I've listed the type of a book as a varchar field, which I expect to take the value "hardback" or "paperback" as in the XML. It would be possible to code this as a boolean, but strictly that's a deviation from the original XML.

Some students renamed the two title fields to book_title and imprint_title. That's not necessary: the ambiguity can always be resolved by referring to them in SQL with the qualified names Book.title and Imprint.title.

Several students made the error of having a book field as a foreign key in the Imprint table. This is entirely wrong: each imprint can contain several books, so there is no single book to point at from the record of a specific imprint. It is sensible to have a foreign key imprint in the Book table, though, so that for each book we can identify the unique imprint to which it belongs.

I've used the short imprint tag as primary key for the Imprint table. It would be possible to use the title field as primary key instead, as these should also be unique. The foreign key in Book would then also be the imprint title, and the solution to the last part of (e) becomes slightly simpler.

(e) The following SQL queries fetch the information required.

(i) **select** tag **from** Imprint

(ii) **select** title **from** Book **where** code = '823095'

(iii)
```
select distinct Imprint.title
from Imprint, Book
where Book.imprint=Imprint.tag and Book.type='hardback'
```

Notice the use of **distinct** here to avoid duplication where an imprint contains more than one hardback book. Also, that the **select** statement uses the full-qualified field name Imprint.title to avoid confusion with the Book.title field.

A few students gave an answer similar to this but with the error of missing out either one of the two tables (**from** Imprint, Book) or the crucial test that links them together (Book.imprint=Imprint.tag).

It's possible to use an explicit **join** here instead.

```
select distinct Imprint.title
from Imprint join Book on Book.imprint=Imprint.tag
where Book.type='hardback'
```

This is exactly equivalent to the query given earlier.

If the Imprint table declaration uses title as its primary key rather than tag, this question becomes a little simpler as each Book record will already include the complete imprint title as a foreign key field.

```
select distinct Book.imprint
from Book where Book.type='hardback'
```

It's possible to argue about whether there might be efficiency problems in using a lengthy text field like the imprint title as a primary key. On the whole, though, these discussions are only sensible in the context of a specific, known, database engine; which may well have perfectly good handling of such key fields through hashes or other methods.

Question 3 [*This question is worth a total of 30 marks.*]

(a) Give brief explanations of the following four kinds of data scale, making clear how each differs from the others: *categorical*, *ordinal*, *interval* and *ratio*. For each one, give an example of data that is measured using that kind of scale. [*8 marks*]

(b) A network testing tool measures the response time for one hundred and one successive test probes of the same remote address: $t_1, t_2, \ldots, t_{101}$. For example, $t_1 = 24.32$ and $t_{101} = 43.60$, both in milliseconds. Give formulas for calculating the *mean* and *standard deviation* for these response times.

Is it meaningful to compute the *median* response time? If so, then say how you would do it. If not, then explain why not.

Is it meaningful to compute the *modal* response time? If so, then say how you would do it. If not, then explain why not. [*6 marks*]

(c) A trading standards officer selects at random for testing five bags of peanuts from a large crate in a shop storeroom. All are sold as "250g" and in fact are found to weigh the following amounts: 250g, 255g, 252g, 258g, 255g.

Estimate the mean and standard deviation of the weights of all the bags of peanuts in the large crate. Use a calculator and write down all your working. [*6 marks*]

(d) Here is a table of how often certain key words appear in an online index to some leaflets held in an archive of 20th-century printed advertisements.

	health	oranges	Vitamin C	natural
Leaflet P	13	0	3	20
Leaflet Q	8	12	9	4
Leaflet R	0	10	0	14
Keywords	1	1	1	1

The *cosine similarity* method for documents ranking uses a *vector space model* to describe documents.

Given two 4-dimensional vectors (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4), with angle A between them, write out a general formula for computing $\cos(A)$ in terms of $\{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}$.

Assuming no other information is available about the leaflets in the archive, use cosine similarity to rank these leaflets in a suggested order for their relevance to the given keywords "health", "oranges", "Vitamin C" and "natural". [*10 marks*]

Notes on Question 3

(a) For this question it's important to include both the description of the scale and an example of its use.

- A *categorical* scale divides data into different named categories. There is no ordering or numerical content. For example, classifying words as different parts of speech — noun, verb, adjective.
- An *ordinal* scale gives a recognised ordering between data items, but there is no arithmetic content. Numbers may still be used to record the ordering, but there is no way to perform arithmetic with them. For example, classifying T-shirts by size as Small, Medium or Large; or dividing degrees as 1st, 2.1, 2.2 and 3rd class.
- An *interval* scale assigns numeric values to data, but where these values are relative to each other. Values can be compared, averaged, and subtracted; but not added together or multiplied. For example, times of day, or temperatures on the Celsius scale.
- A *ratio* scale uses numeric values which have an absolute notion of zero; this means they can sensibly be added, and multiplied by real numbers. For example, weight, height, or length of objects; or temperatures on the Kelvin scale.

Some incorrect answers to this question confused the measurement scale and the item being measured. It's not the individual item that's important here — a particular word, or a certain T-shirt — it's the scale on which they are measured: being classified as a noun/verb/etc. or small/medium/large.

(b) A few students misunderstood "response time" for a test probe: the response time is the length of time between sending out the test probe and receiving a reply. Expanding on the data in the question, for example, the first five values in milliseconds might be $t_1 = 24.32, t_2 = 16.08, t_3 = 50.11, t_4 = 30.42$ and $t_5 = 17.64$.

Mean and standard deviation of observed response times can be calculated as follows.

$$\text{Mean } \mu = \frac{1}{101}\sum_{i=1}^{101} t_i \;=\; \frac{t_1 + t_2 + \cdots + t_{101}}{101}$$

$$\text{Standard deviation } \sigma = \sqrt{\frac{1}{101}\sum_{i=1}^{101}(t_i - \mu)^2} \;=\; \sqrt{\left(\frac{1}{101}\sum_{i=1}^{101} t_i^2\right) - \mu^2}$$

In each case either of the two presentations is fine. However, it's important here to adapt the general formula to the specific situation of the question: in this case, 101 values $t_1, \ldots, t_{101}$, rather than just "N" values "$x_1, \ldots, x_N$".

Calculating the median response time is meaningful: half of the tests take less time than this to return a response, and half take more. Unlike the mean it isn't strongly affected by single extreme results, such as if a single probe takes 1000 milliseconds to return.

Computing the media is done by sorting all 101 response times in order and choosing the value in the middle, the 51st. It's important here to sort the times first — you can't just take the value t_{51} of the 51st test probe.

Calculating the modal response time is not meaningful. The mode is defined as the most common value; but here the times are real numbers and no two will ever be exactly equal.

In this example they have been rounded to two decimal places, but even if that means that two values appear to be equal it is an accident of rounding and will depend on the number of decimal places chosen.

(c) Notice that the question is about the mean and standard deviation of the weights of all the bags of peanuts in the crate. We don't know exactly what those are, as we have only a sample of five bags. However, because they were chosen at random it's reasonable to use the weights of those five to make estimates about the weights of all the bags.

To estimate the mean weight of all bags in the crate, we can use the mean weight of the five bags in the sample.

$$\mu = \frac{250 + 255 + 252 + 258 + 255}{5} = \frac{1270}{5} = 254$$

Thus our estimate m of the mean weight of all bags in the crate is 254g.

To estimate the standard deviation in weight across all the bags we cannot use the standard deviation σ_n of the sample as that will be slightly too small. Instead we apply the Bessel correction and compute σ_{n-1}:

$$\begin{aligned}
\sigma_{n-1} &= \sqrt{\frac{(250-254)^2 + (255-254)^2 + (252-254)^2 + (258-254)^2 + (255-254)^2}{(5-1)}} \\
&= \sqrt{\frac{4^2 + 1^2 + 2^2 + 4^2 + 1^2}{4}} \\
&= \sqrt{\frac{38}{4}} \\
&= 3.1
\end{aligned}$$

This gives an estimate s of 3.1g for the standard deviation of weights in the crate.

It's possible to give more digits of precision 3.08220700148448822512509619072 71. . . but this doesn't really add useful information as the original data is only given to three significant figures

Note that the sample is small enough that this is clearly different to the standard deviation σ_n of weight among the five bags, which is only 2.8g.

(d) This is the formula for computing the cosine of the angle between two 4-vectors.

$$\cos(A) = \frac{x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4}{\sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2}\sqrt{y_1^2 + y_2^2 + y_3^2 + y_4^2}}$$

As with the calculation of mean in part (b), although this uses a general formula it's important to apply it to the question as stated. Some students gave incorrect answers with entirely different vectors and variables, such as $\vec{u}.\vec{v}/|\vec{u}||\vec{v}|$, rather than $x_1, y_1, \ldots$ from the question.

For the three leaflets listed, the appropriate calculation is the cosine between each report and the set of keywords

$$\begin{aligned}
\cos(\text{Leaflet P}, \text{Keywords}) &= \frac{13 + 3 + 20}{\sqrt{4}\sqrt{13^2 + 3^2 + 20^2}} &&= \frac{36}{2\sqrt{578}} = 0.75 \\
\cos(\text{Leaflet Q}, \text{Keywords}) &= \frac{8 + 12 + 9 + 4}{\sqrt{4}\sqrt{8^2 + 12^2 + 9^2 + 4^2}} &&= \frac{33}{2\sqrt{305}} = 0.95 \\
\cos(\text{Leaflet R}, \text{Keywords}) &= \frac{10 + 14}{\sqrt{4}\sqrt{10^2 + 14^2}} &&= \frac{24}{2\sqrt{296}} = 0.70
\end{aligned}$$

Leaflet Q, with the largest cosine to the vector of keywords, and hence the smallest vector angle, is ranked highest in relevance.

Ranking all three documents in this way gives us the following:

- Leaflet Q;
- Leaflet P;
- Leaflet R

ordered from most relevant to least relevant.

A very large number of students gave correct answers to this. A few tripped up in the very last section: where the question says to rank the leaflets, some identified only the highest ranked leaflet.