# TMA 02, Question 1

**Name**: [Pablo Toledo]

**PI**: [C4451553]

1a) Storing data for Open Sitters in a file-based system would have a number of disadvantages over a relational database approach, including:

1. A tight coupling between the Open Sitters application and its data, leading to poor maintainability

The application would have to contain the file / data definition, meaning that any changes in the data structure (such as new fields or constraints) would require a change in the Open Sitters application as well as any other application which relied on this data.

1. No support for concurrent users or transactions

As this system will be used by many users at the same time, proper concurrency support would be needed to provide adequate response times, but also to guard against simultaneous updates interfering with each other (e.g. lost updates, or reading uncommitted data). Transaction support would also be required to ensure that operations involving more than 1 update either complete successfully or not at all, therefore guaranteeing the consistency of the data. As file based data systems have no support for either concurrency or transactions, the application would have to implement these separately, incurring additional time and money compared to a modern DBMS. Further, this mechanism would have to be implemented separately in any other application which wanted to access this data.

1. Lack of access control leading to potential legal issues

File based data systems store data in sequential files on the file system. Any user with access to the file system could in theory view, modify or delete the contents of the data (or write a program that did so). Without proper access control which grants the appropriate access level (read, write, delete) to authorized persons, implementing a file-based system is problematic from a legal point of view as it could fall foul of data protection legislation which puts responsibility on the data controller to protect personal data of individuals (e.g. families or babysitters) far as possible.

1b) A possible schema-less data structure for use in mongoDB could be as follows:

{

"id": "1234567890ABCDEF",

"primaryCarer": {

"title": "Mrs",

"firstName": "Jane",

"surname": "Doe",

"mobile": "07812345678",

"landline": "023 8045 6789",

"email": "jane.doe@example.com"

},

"secondaryCarer": {

"title": "Mr",

"firstName": "Jon",

"surname": "Doe",

"mobile": "07887654321",

"email": "jon.doe@example.com"

},

"children": [

{

"fullName": "Johnny Doe",

"dateOfBirth": "01/01/2014"

},

{

"fullName": "Lucy Doe",

"dateOfBirth": "01/01/2015"

},

{

"fullName": "Jerry Doe",

"dateOfBirth": "01/01/2016"

}

]

}

1c)

**FAMILY(FamilyId, Title1, FirstName1, Surname1, Mobile1, Landline1, Email1, Title2, FirstName2, Surname2, Mobile2, Email2)**

FamilyId is the primary key

**CHILD(ChildId, FamilyId, FullName, DateOfBirth)**

ChildId is the primary key

FamilyId references FAMILY

Family has a one-to-many relationship with the Child table (one row in the family table can have 0 or more rows in the child table). This relationship can be implemented using the DDL (data definition language) for the particular RDBMS we choose, and would involve declaring the FamilyId in the Child table as a foreign key which referenced the Family table.

In the Family table, Title1, FirstName1, Surname1, Mobile1, Landline1 and Email1 (corresponding to all the fields for the primary carer) are all mandatory text fields. Title1 and Title2 are also text fields and should be one of [Mr, Mrs, Ms].

Title2, FirstName2, Surname2, Mobile2 and Email2 are mandatory if any one of them is populated, otherwise they are optional. This ensures that if the details of a secondary carer are entered they are completed properly.

In the Child table, FamilyId, FullName and DateOfBirth are all mandatory.

FullName is a text field, and DateOfBirth would be a date field.

The above scheme assumes each child belongs to exactly 1 family, and has at least 1, and at most 2 carers (in reality there may be more, for example, mother, father and grandmother). It also assumes that a secondary carer won’t have a landline (perhaps because the assumption is that the primary and secondary carer live in the same house), but will have a mobile. Families with no children are also possible, as will be the case when a family has registered (and has an entry in the Family table) but not yet added any children to the Child table.

1d) The key differences between the document-based model and relational model for Open Sitters are:

* In the document-based model, data about families and children is stored in documents which consist of key-value pairs. A value may be atomic (e.g. ‘id’), or it may be another sub-document (e.g. ‘primaryCarer’). In a relational database, the data would be stored in Family and Child tables, each table consisting of attribute columns (such as ‘FirstName’ and ‘DateOfBirth’) and rows which represented an instance of a family or child.
* The document-based model is schema-less, meaning fields (such as ‘landline’ or ‘skypeId’ could be added, removed or the data structure changed at a later date (to allow for a list of carers for example, instead of just 2). The relational model on the other hand prescribes the schema, with the assumption that the Family and Child fields, data constraints and relationships will not change.
* The document-based model works with ‘aggregate data’, that is un-normalized data contained in a single document (in this case, the aggregate contains the family, carer and child data in a single document). In the relational model the family and child data is normalized and stored in different tables, linked by primary and foreign keys.
* In the document database solution, family and child data can easily be shared across multiple servers when the data volume increases (‘scaling out’). In a relational database, data is assumed to reside in a single database on one server, meaning a larger more powerful server will be needed when the data volume increases (‘scaling up’).

1e)

DBMS systems ensure that data integrity is maintained through supporting the concept of a transaction, where each transaction consists of a number of steps. In this case the booking process might consists of the following steps:

1. Add new row to Booking table
2. Read booking count in Family
3. Increment booking count and update booking count in Family table

If, due to an application or hardware error, only the first 2 steps execute, then the booking count for a new booking will be left in an inconsistent state. Transaction support allows for steps 1 to 3 to be wrapped in a transaction so that these steps are treated as a single unit of work (atomically). If the transaction to update the booking succeeds, then all the component steps of the transaction have completed and the database is left in a consistent state (i.e. the booking count is correct when a new booking is added). If the transaction fails, then any steps carried out are ‘rolled back’, leaving the database in the state before the transaction started but crucially still consistent (i.e. the new booking row will be removed). The transaction can then be attempted again at a later time. Transaction support therefore allows the application to update the database in such a way that an inconsistent state should never arise. Similarly, if any hardware error occurs during a transaction (e.g. a power outage), the transaction taking place at the time of the failure can be rolled back to restore the data to a consistent state.

1f)

The following features of relational databases allow the database designer to control the type and format of the data which is persisted:

**Column Typing**

In relational databases, the DDL (data definition language) requires the database designer to specify the data type for each column in a table. This forces the designer to be precise about what data type of data should be used to represent an attribute, and what size or precision should be used. From the outset this leads to a higher degree of data consistency, as the DDL has essentially specified a contract with the users that a particular attribute should always be represented in a certain way. For example, the Open Sitters relational database scheme specifies that ‘DateOfBirth’ should always be of type Date. Using the DDL to type this column as a date guarantees that only valid dates can be stored (without any kind of conversion or data-cleansing required), and allows easy sorting and processing of children using date of birth. Similarly, while an integer might appear a good choice of type for ‘Mobile1’ and ‘Mobile2’, no processing on these fields will be required, and using a string instead of an integer allows the user to enter dashes or spaces if they wish (as well as preventing any leading zero from being lost). Thus column typing is a powerful feature of RDBMS which guarantees that only the correct type of data is stored in a relation.

**Integrity Constraints**

Integrity constraints refer to the ability to use the DDL in relational databases to define rules about the data stored in each column. For example, in the Open Sitters relational database it would be sensible to define a constraint to check that ‘Title’ is one of the allowed values [‘Mr’, ‘Mrs’, ‘Ms’] (CHECK constraint). Similarly, we would want to enforce the entry of a child’s date of birth (NULL constraint), and check that the ‘FamilyId’ for a child corresponded to a row in the Family table (FOREIGN KEY constraint). If a user tried to enter a carer with an invalid ‘Title’ value, or a new child without a ‘DateOfBirth’ value, or the application tried to write a new child row with an invalid ‘FamilyId’ value, then the corresponding integrity constraint would fail and the invalid data would be rejected. This powerful feature of RDBMS goes a long way to ensuring only valid, consistent data is stored in the database.

**Default Values**

Relational databases also support defaulting of column values in the DDL. This can be another useful means of ensuring data consistency, as an attribute can be guaranteed to hold an initial value unless explicitly set. For example, we might make the assumption in the Open Sitters relational data model that the primary carer is the Mother, and the secondary carer the Father. In this case, providing a default value of ‘Mrs’ for the primary carer ‘Title1’ field, and a default value of ‘Mr’ for the secondary carer ‘Title2’ field. If this is true 80% of the time, then data entry has been simplified for 80% of cases. Default values are especially valuable for non-mandatory fields. For instance, if an optional field ‘Alergies’ was added to the Child table, we could default it to ‘None’ so that it was not accidentally left blank.

**Advantage of using a schema-free database**

While column typing, integrity constraints and default values are all powerful mechanisms to ensure data consistency, they also bring with them a certain inflexibility which can a disadvantage. The initial relational schema may be wrong, or the business requirements may change over time which leaves the schema obsolete. For example, while initially it may seem reasonable that you can only have a maximum of 2 carers for a child, it might come to light over time that this is unnecessarily restrictive. If there are more than 2 carers (e.g. Mother, Father, Grandmother) then the relational schema does not allow us to capture this. Similarly, if there is only 1 carer (e.g. single-parent), the secondary carer fields are irrelevant in the database and have to be set to null. While this could of course be solved in the relational world by having a one-to-many relationship between family and carers, such a change would be costly to carry out once the Open Sitters application has gone live. A Schema-less database allows us the freedom to have 1, 2, 3 or any number of carers (perhaps storing these in a ‘carers’ subdocument) without having to change the application or database. Similarly, if Open Sitters decides that Skype is the best way for staff to communicate with carers and babysitters, skype details could easily be added to a carer / babysitter document without the need to modify the database schema, making changing requirements quicker and easier to manage.

**Disadvantage of using a schema-free database**

The flexibility of a schema-less database comes at a price. Using a document to store family, child and carer details is more flexible. Inserting, updating and retrieving a family document is fast compared to using a RDMBS. These operations work well because the unit of work is the ‘data aggregate’ itself. i.e. the family document. However, querying the data along other lines is more complex, and resource-intensive. For example, the finance may want monthly reports of booking broken down by region. Because the data-aggregate is family-oriented, such a query would involve processing each document in turn, and parsing it to extract and examine the bookings. The formation of the query is harder than the SQL equivalent, and more resource-intensive to run. If the Open Sitters application requires a large number of such reports and the ability to run ad-hoc queries then the benefits of flexibility allowed by a schema-less database may be quickly outweighed.

2a)

From skimming the BBC referendum results page and the electoral commissions referendum report, some different ways used in which the referendum result were reported are:

* The number of leave, remain and undeclared votes cast nationally and regionally (England, Northern Island, Scotland and Wales)
* The percentages of leave, remain and undeclared votes cast, broken down:
  + Nationally
  + Regionally
  + By county
* Pie charts showing the % of leave / remain votes out of the total votes cast
* Maps showing counties with more than a 50% leave vote in one colour (often blue), and less than 50% leave vote in another colour (often yellow)

2c)

**Report into Voting Patterns by region in the EU Referendum**

**Aims and objectives**

The aim of this brief investigation was to:

* Become familiar with the EU referendum data available from the electoral commission
* Identify any voting patterns or trends at the national level
* Identify further questions worthy of exploration
* Identify other data sets which could be useful in further investigations

**Background**

The motivation for this investigation was to become familiar with the EU referendum data available from the electoral commission, and understand its meaning and context, and identify further questions/areas for investigation. Those questions could involve analyzing data at the voting area level, and/or correlating the electoral commission data with other data sets.

**Sources of data**

The data sources for this investigation were:

EU referendum report by electoral commission:

<http://www.electoralcommission.org.uk/__data/assets/pdf_file/0008/215279/2016-EU-referendum-report.pdf>

My investigation can be reproduced with the following python notebook:

C4451553\_TMA02\_Question2b.ipynb

**Analysis pipeline**

The analysis proceeded as follows:

1. Familiarized myself with data sources (bbc referendum page, csv data)
2. Imported EU referendum csv data into a panda dataframe, and examining its shape and format
3. Clarified meaning of various columns relating to spoiled votes
4. Sanity checked the data
5. Pivoted data to summarize the voting areas by region
6. Plotted the total electorate for each region, in order to get a feel for how much they differed in size
7. Examined the number of votes cast, and votes uncast in each region to see how this differed from the size of the electorate
8. Examined percentages of votes cast and votes uncast
9. Investigated the possible link between turnout percentage and leave vote %

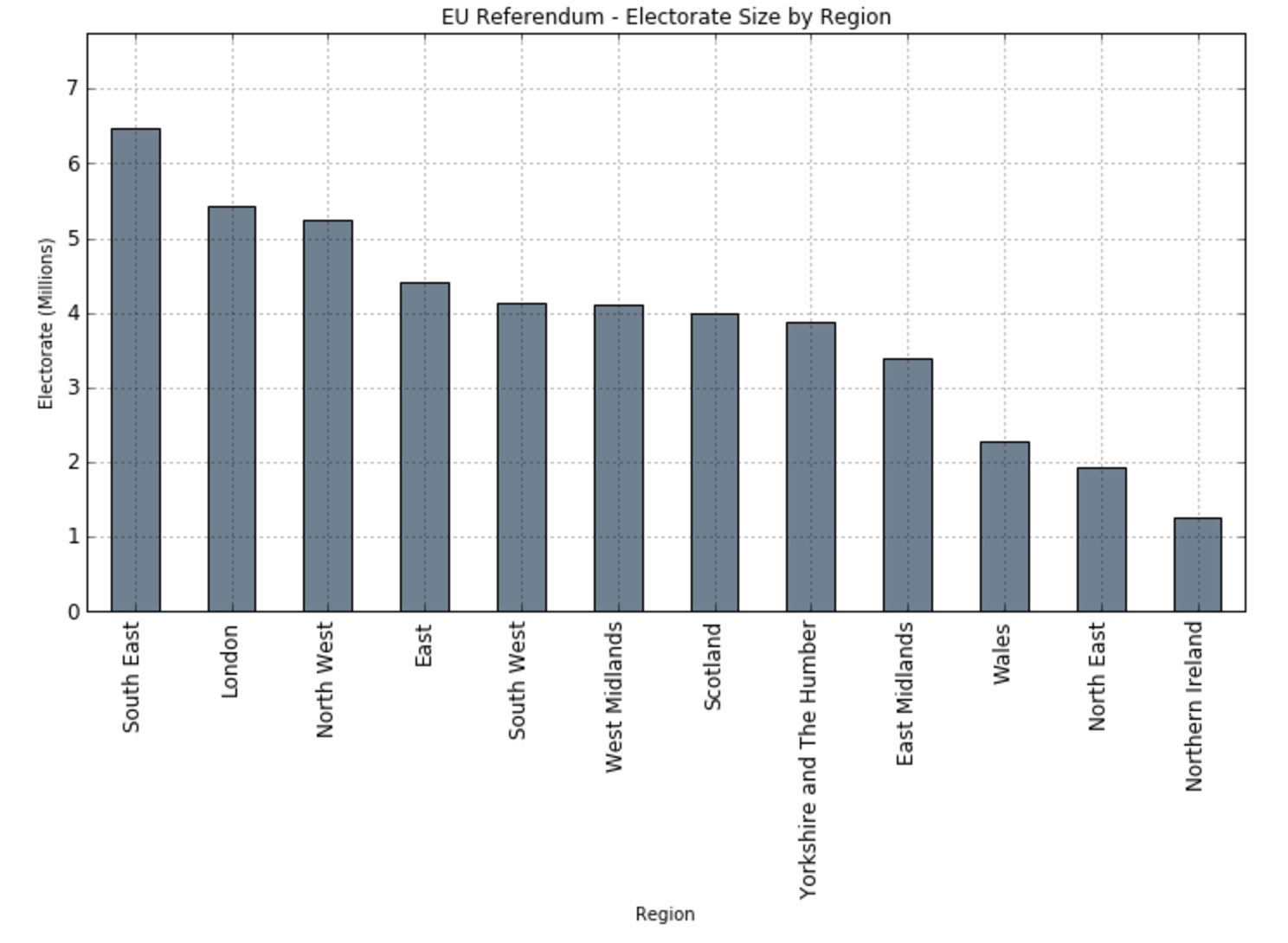
All my analysis is documented in the C4451553\_TMA02\_Question2b.ipynb

Python notebook.

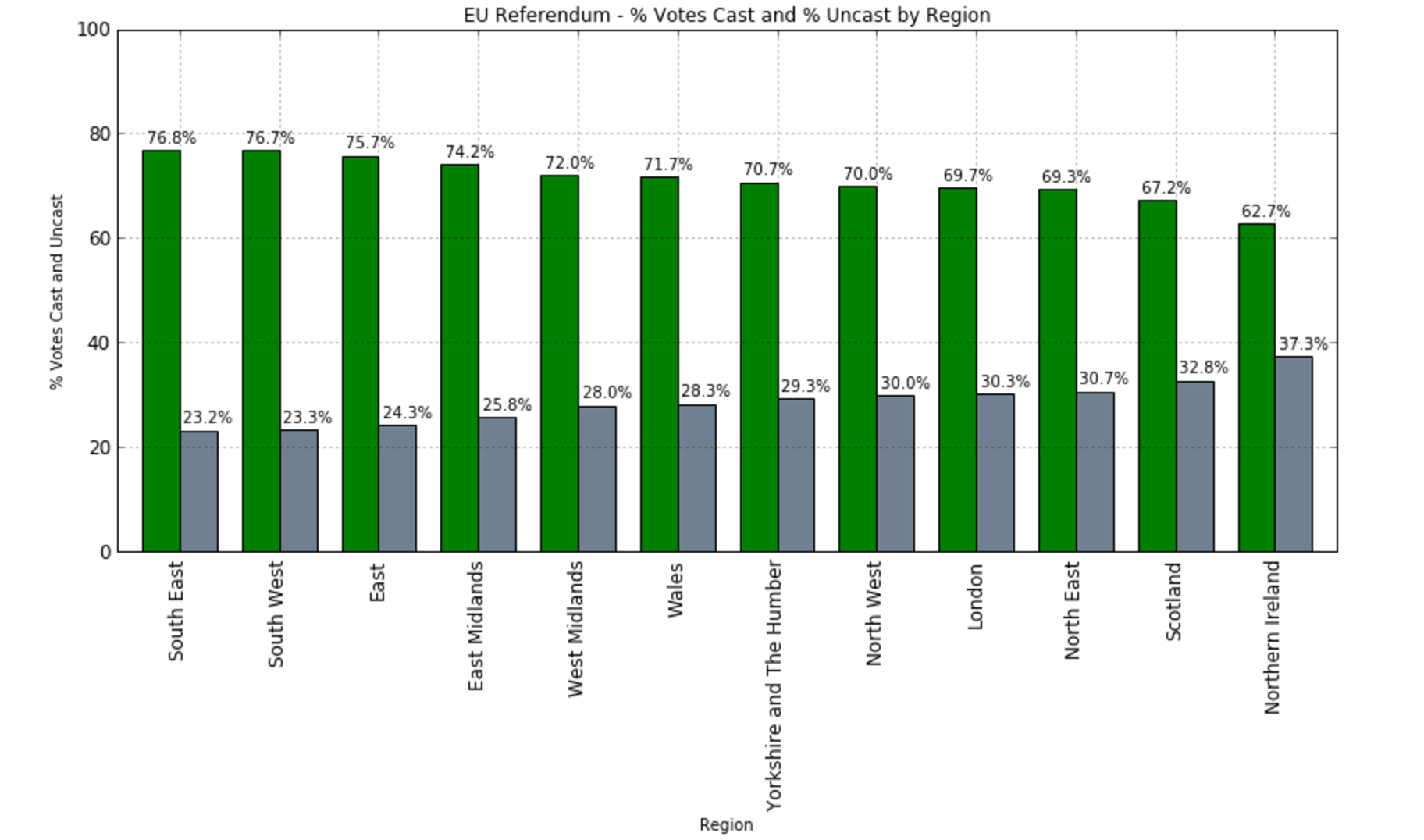
**Findings**

In this analysis my findings were as follows:

1. There was a large difference in the size of the electorate (i.e. eligible voters) across different regions (with the largest being roughly 6 times the size of the smallest), as can be seen in this plot:

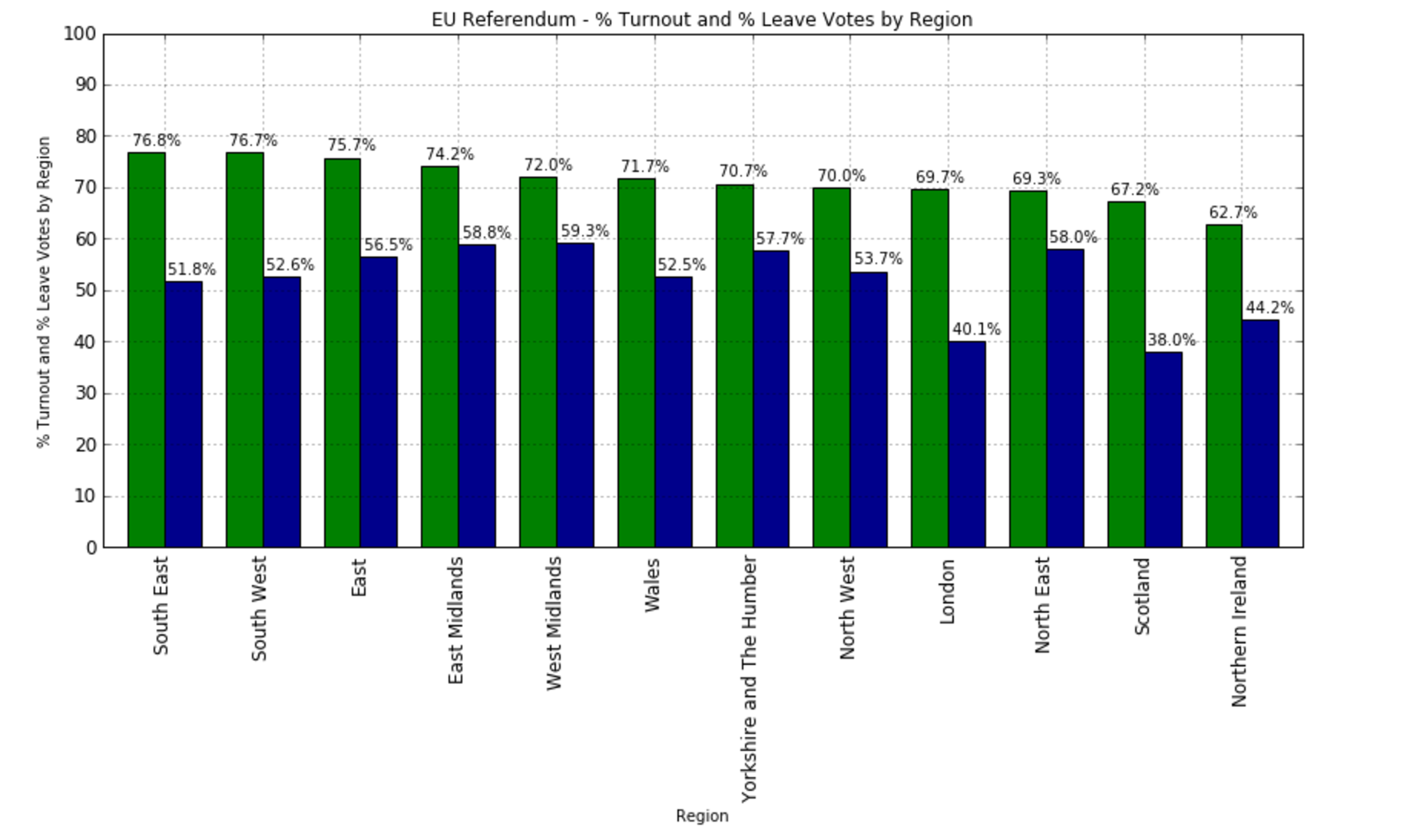


1. Number of votes cast also varied greatly between regions, though it was not the case that a larger electorate resulted in a larger number of votes being cast, due to differences in % turnout between regions. The following plot demonstrates the regional differences in % turnout.



1. The answer to the question ‘is turnout % related to leave vote %?’ was inconclusive. While the plot shows that prominent remain regions (London, Scotland, Northern Island) do have some of the lowest

turnout rates (69.7%, 67.2%, 62.7%), this turned out not to be true for the North-East which also had a low turnout rate of 69.3% but a high leave vote % (58%):



**Conclusions**

As this was a brief, preliminary investigation, no solid conclusions could be drawn about voting patterns and what gave rise to them. However, this analysis did highlight the large differences both in terms of votes cast (turnout), and leave vote % between regions. Further analysis could investigate a possible connection between turnout % and leave vote % by voting area, as the small number of data points at the regional level makes any kind of conclusion hard to reach. Further investigations could also seek to correlate other factors, such as ethnic diversity, employment rates or average incomes against turnout and leave vote % to see if there is any correlation.

**References**

EU referendum report by electoral commission:

<http://www.electoralcommission.org.uk/__data/assets/pdf_file/0008/215279/2016-EU-referendum-report.pdf>

PDF from electoral commission explaining different categories of ‘spoiled votes’:

<http://www.electoralcommission.org.uk/__data/assets/pdf_file/0012/87699/UKPE-doubtfuls-booklet.pdf>

BBC News – EU referendum results and interactive map:

<http://www.bbc.co.uk/news/politics/eu_referendum/results>

EU referendum data for each voting area, available from electoral commission:

<http://www.electoralcommission.org.uk/find-information-by-subject/elections-and-referendums/past-elections-and-referendums/eu-referendum/electorate-and-count-information>

Wikipedia article on EU referendum:

<https://en.wikipedia.org/wiki/Results_of_the_United_Kingdom_European_Union_membership_referendum,_2016>