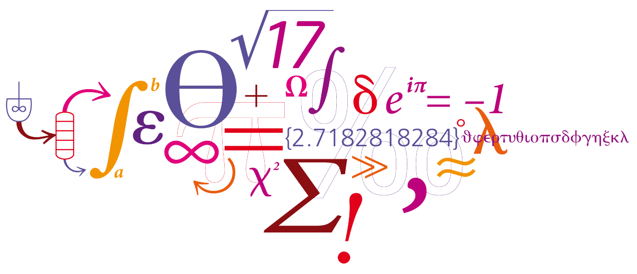
Technical University of Denmark

22110 Python & unix for Bioinformaticians

Python Project

Data Analysis

Aditya Leju Chacko s212923
Eleni Kiahaki s222777



January 22, 2024

Contents

1 Introduction

Data analysis is defined as the practice of working with data to glean useful information, which can then be used to make informed decisions. The aim of the present "Data analysis" project was to extract data and meaning out of them, in order to answer statistical questions.
As source of data, a flat file database was used, including personal information about 500 different individuals (CPR, first and last name, height, weight, eye color, blood type, children).
The exact purpose was to construct a program that reads the aforementioned database once, stores the data in a suitable data structure of our own choice and analyzes specific parts of data to extract meaningful statistical results. It is worth noting that most of the effort was put into the extraction of the data, rather than the statistical analysis of them.

2 Contribution

Equal contribution from both members

3 Theory

3.1 Age calculation:

The calculation of the age of people in the database was made using the 5th and 6th digits of each person's CPR, as they represent the person's birth year. Taking into account that the dates in the given database range between 1900-1999 and assuming that we are doing the analysis primo 2000, we calculated the ages with the following formula:

$$Age = 100 – int(CPR[5:7])$$

which is a simplified version of :

$$Age = 2000 – (1900 + int(CPR[5:7]))$$

3.2 Gender determination:

The gender of every person in the given database was determined by the last digit of his CPR number. As it is stated in the project's description, odd last digit indicates a male while even number, a female.

3.3 Calculation of number of cousins:

The calculation of the number of cousins was done for each and every child separately. Thus, siblings, who have the same number of cousins, constitute separate elements in the lists of cousins (cousinlist, cousinlistclear).

3.4 Height distribution:

In order to estimate the distribution of people in relation to their height, the mean height value was calculated. Then, people whose height deviates more than 2,5% from the mean, above or below of it, were considered as tall or short, respectively. People whose height ranges within 2,5% of the average value, were considered as normal.

Short:

$$height < 0.975(average.height)$$

Normal:

$$0.975(averageheight) <= height <= 1.025(average.height)$$

Tall:

$$height > 1.025(average.height)$$

3.5 Weight distribution:

In weight distribution, the different groups that people were divided into, were determined using BMI categories.[1]

Slim:

$$BMI < 18.5$$

Normal:

$$18.5 <= BMI <= 24.9$$

Fat:

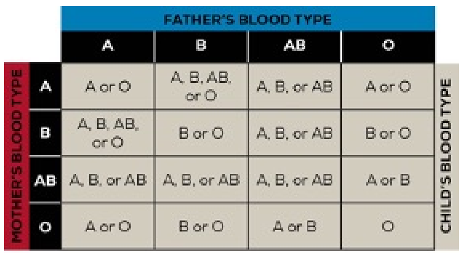
$$BMI > 18.5$$

3.6 Blood type inheritance and compatibility:

Blood type is determined by the ABO type and a negative or positive Rh factor. The ABO blood type is determined by the presence or not of A and B antigens on the surface of red blood cells. More specifically, the A and B blood types are characterized by the exclusive presence of A or B antigens respectively, the AB blood type has both types of antigens while the O blood type has neither of them[2].
Red blood cells sometimes have another antigen, a protein known as the RhD antigen. If this is present, the blood group is RhD positive. If it's absent, the blood group is RhD negative[3].

3.6.1 Blood type inheritance:

The A and B antigen molecules are made by two different enzymes. These two enzymes are encoded by different versions/alleles, of the same gene. The A allele codes for an enzyme that makes the A antigen, and the B allele codes for an enzyme that makes the B antigen. A third version of this gene, the O allele, codes for a protein that is not functional. Everyone inherits two alleles of the gene, one from each parent. The combination of these two alleles determines your blood type[2].
All possible alleles combinations and the resulting blood types are summarized in the table bellow:



		FATHER'S BLOOD TYPE			
		A	B	AB	O
MOTHER'S BLOOD TYPE	A	A or O	A, B, AB, or O	A, B, or AB	A or O
	B	A, B, AB, or O	B or O	A, B, or AB	B or O
	AB	A, B, or AB	A, B, or AB	A, B, or AB	A or B
	O	A or O	B or O	A or B	O

CHILD'S BLOOD TYPE

Figure 1: Blood inheritance [4]

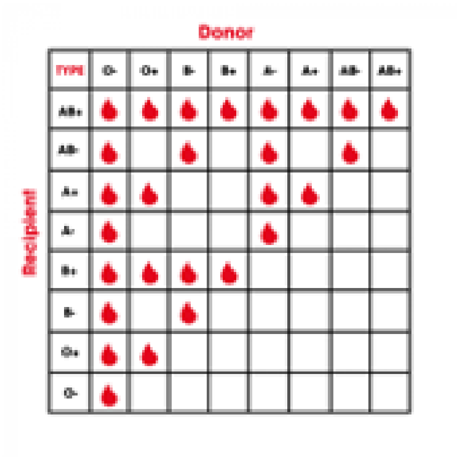
Rh status is also genetically inherited, but separately from the ABO blood type. If the dominant Rhesus D antigen is inherited from one or both of the parents, then the resulting blood type is Rh-positive. Otherwise, the blood type will be Rh-negative[5].

3.6.2 Blood compatibility

Every person has developed, over time, antibodies against the antigens that his red blood cells lack. As a result, a person with A-type blood will have anti-B antibodies and the other way around. O-type people have both anti-A and anti-B antibodies, while AB-type lack both [6]. Consequently, a person with A blood type, cannot receive B blood because

his body's anti-B antibodies will interact with B blood's B antigens and generate an immune response to the recipient. On the other hand, AB-type people, can receive blood from everyone, because of the lack of specific antibodies, while O-type individuals only from people of the same blood type[7].
As far as Rhesus compatibility is concerned, A person with the Rh positive factor will not make anti-Rh antibodies. Those with Rh negative factor will produce the antibodies. Therefore, someone with Rh+ blood can receive both Rh+ and Rh- transfusions, but those with Rh- can only receive Rh- blood[8].

The blood type compatibility is summarized in the table bellow:



Donor
Recipient

TYPE	O-	O+	B-	B+	A-	A+	AB-	AB+
AB+	●	●	●	●	●	●	●	●
AB-	●		●		●		●	
A+	●	●			●	●		
A-	●				●			
B+	●	●	●	●				
B-	●		●					
O+	●	●						
O-	●							

Figure 2: Donor/Receiver chart [9]

In question 15, as not true parents were considered the parents whose kids could not have inherited the blood type they have, from them.

4 Algorithm Design

4.1 Main Dictionary

Algorithm used to create the main dictionary is given below:

```
Lines 18-101

    Open the .db file
    Iterate through each line in the file
        if line has main CPR
            CPR number is stored
            Every time a new CPR is encountered,
            variables are storedin the main Dict

        if line has the first name
            first name is stored
        if line has the last name
            last name is stored
        if line has the Height
            height is stored
        if line has the weight
            Weight is stored
        if line has the Eye Color
            Eye color is stored
        if line has children
            children is stored
        if line has blood type
            blood type is stored

    Note: Variables are checked in every line using regular expressions

    Structure of the main dict:
    CPR- primary key
    height, weight, first name, last name, Children are stored as values
    to their respective secondary keys
```

4.2 Functions

4.2.1 'dividetogroups' function:

The 'dividetogroups' function is used everytime we are asked to estimate the distribution of a feature, for all entries in the database. This happens in questions: 1 (age and gender distribution), 3, 5 (distribution of age at which someone becomes father/mother for first time). This function has one parameter, a list of numbers, and depending on its maximum and minimum element, it divides the elements in a number of groups of 5-year range, and then it counts how many elements belong in each group. The pseudo code for the function is given below:

```
Lines 109-163
        sort the list that is used as parameter

        count how many digits are there in maximum
        and minimum age

        isolate all digits except the last of minimum and maximum age
        (or the 1 digit when len(string) = 0)

        initialize a list where elements will be the counts of ages
        for each category
        initialize an empty string, for the printing format
        initialize two dicts, to display the results
        in numbers and percentages


        if last digit of minimum age < 5:
            round the age: 10* ( minimum age's digits except the last)
        else:
            round it adding 5: 10* (minimum age's digits except the last)+ 5


         if last digit of maximum age < 5:
            round the age: 10* ( maximum age's digits except the last)+ 5
        else:
            round it adding 10: 10* ( maximum age's digits except the last) + 10


        for i in range (minrounded age, max rounded age, 5):
            append the limits of the age group in the list groups

        iterate over groups
            initialize counter
            if it is not the last element of groups:
                iterate through the list/parameter of function:
                if the element is smallr than an age limit and bigger
                form the previous age limit:
                    increment the counter
                append the final value of the counter in counts list

        iterate over  counts
            construct string that shows the age group
            add the age group as a key and the number of people in it as
            its value in dictionary
            add the age group as a key and the percentge of people in it
            as its value in dictionary

            return the dict with the percentages
```

4.2.2 'averagefunc' function:

This function uses a list of numbers as parameter, and calculates the mean value of its elements. The pseudo code that describes the function is provided below:

```
Lines 170 - 178

        initialize local variable that will be used for the summary
        iterate through list
            add each element to the summary variable
        divide the average with length of list
        return the result
```

4.2.3 'underline' function:

Function used only for underlining specific text in print statements.

4.3 Fathers & Mothers dictionary

The code also uses the mothers and fathers dictionary at multiple points to solve the questions. The Pseudo code/ Algorithm used to create it is given below:

```
Lines 189 - 229

    Iterate through every CPR number (primary key) in main dict
        if the last digit of the CPR is even
            the CPR is female
            counter(females) is iterated
            if the CPR has children
                Children is added as values with the mother as key
                in dictionary called 'mothers'

        else if the last digit is odd
            the CPR is male
            counter(males) is iterated
            if the CPR has children
                Children are added as a value list with CPR as Key
                in dicionary called 'fathers'
```

4.4 Children's Dictionary (childict)

The childrens dictionary consists of a child's CPR as the keys and the child's parents as values. Below is the pseudo code used to create it:

```
Lines 259 - 279

      Store all the mothers(keys) from the dictionary 'mothers' as a list
      Store all the fathers(keys) from the dictionary 'fathers' as a list

      Iterate through the list of all fathers

          store the children of a father in a list (kidlist)

          Iterate through the lit of children each father has

              Look for the child's CPR in the values of mothers dict
              If child CPR is found

                  assign the mother(key) that matches the value in
                  'mothers'and father's CPR as values with child's
                  CPR as key in a new dict called 'childict'
```

4.5 Question 7

The following pseudo code is used to calculate the age difference between the parents and is also used to store each unique pair of parents in a list. This unique parents list is used later in questions 12-14.

```
Lines 283 - 307


    The list of children(keys) in 'childict' is sorted based
    on the values(parents). With this, siblings will be
    adjacent to each other in the list of children.

    Iterate through the Children's CPR (sorted)

        If the Child at position i does'nt have the same parents as
        child at postion i+1 (they aren't silings) the parents are
        the parents at position  i are stored as a unique pair of
        parents in list 'uniqueparentpairs'.

            the age difference of these unique parents are
            calculated and stored in a seperate list
            'parent_agediff'
```

4.6 Question 2,3,4,5 & 10

The following logic is used to find out the first time a mother/father became a parent and finding out the gender of the first born children.

Below is the pseudo code of Question 4,5 and 10:

```
Lines 317 - 348

    Iterate through all the mothers in dict 'mothers'

        Store the children of a mother in a list

        Iterate through the list of children

            Find out the age of each child from
            'maindict' and assign the child's CPR
            as key and age as value in a new dict
            called 'kidsagedict'

            The CPR of children in 'kidsagedict'
            is sorted based on the values(ages)

            The mothers age is subtracted by the
            age of the eldest kid is stored in a
            list called 'firstmotherlist'
            (This gives the first time the mother
            became a parent)

            The eldest kid's gender is noted and
            based on that either a counter for
            first born girls or first born boys
            is iterated.
```

The same logic is applied to solve the questions 2 and 3 (lines 370 - 390). The only difference is that the eldest kid's gender is not noted.

4.7 Cousins Dictionary(cousins dict)

The cousins Dictionary was created to find out the number of cousins each person has Inorder to determine if a person has cousins, the person needs to have grandparents in the database.

Below is the pseudo code used to create the cousins dict:

```
Lines 447 - 485

        Iterate through list of children(keys) in the childdict
```

```
            stores the CPR of child’s mother
            stores the CPR of child’s father

            Iterate through every CPR in maindict

                If mothers CPR is found as an entry in a CPRs [’Children’]

                    Store all the Children(Mother’s sibings) in a list

                    Iterate through the list with Children

                        Checking if the children in list have children
                        (Checking if cousins exist)
                        Counting the number of kids each sibling has
                        (counting the number of cousins from the Mother’s side)

                Repeat the same procedure for the father’s CPR
                sum the number of cousins from the mother’s side and father’s
                side and store it in a dictionary(cousinsdict) as value
                and Child’s CPR as key
```

4.8 Grandkids dictionary (grandkids)

The grandkids dictionary was created for finding out the grandchildren that can be blood donors to their grandparents. The grandkid’s CPR is the key in the dictionary and the value is the grandpaents the person has. Here is how the grandkids dictionary was created:

```
Lines 412 - 436

       Iterate through the list of children in childict
            store the value of the mother and father of a child
            from childict

            Iterate through the list of children in childict again

                    Check if the mother and the father are present as
                    keys in childict.If so, Store the value of the
                    mother and father’s parents(grandparents)
                    in 2 seperate lists

             add the child’s CPR as key in grandkids with the grandparents
            (both from the father’s side and the mother’s side as values)
            the Value is a list of lists.
```

4.9 Questions 12, 14 (Statistics related to height and weight)

Questions 12 and 14 were approached in a similar way, for two different features: In question 12, height was examined, while in 14, weight. The pseudo code that explains the logic behind both questions is written bellow:

```
lines 558 - 594

    Initialize counters for couple categories,
    for each possible height / weight combination

    Iterate over the list of uniqueparentspairs
        if the element is within the limits of a height/ weight group
        (* if it is a tall-tall couple, append it to list)
            add it to the counter of relevant couple category

Question 14 : Lines 628 - 656
```

* Every couple consisting of two tall parents, is appended to a list named tallparents, that will be used in question 13.

4.10 Question 13 (Statistics related to tall peoples' kids)

In question 13, it was estimated whether tall people are having tall kids. This was done as the pseudo code bellow describes:

```
Lines 599 - 620

    Initialize counters for tall kids and kids of tall couples
    iterate through tall parents
        calculate the total number of kids of tall parents
        iterate through kids of tall parents
            check if they are tall and add them to counter
```

4.11 Question 15 (Finding out non biological parents)

Pseudo code for finding out non biological parents is given below:

```
Lines 661 - 726

       Iterate through the keys in childict
            Store bloodtype of child, father and mother

            (Checking if the Rhesus of the parents doesn't
            match the child)
            If both parents are (+) and child is (-)
            Else if the parents are both (-) and
            child is (+)
                child is non-biological
                and is added to a list('adopted')

            Else the blood group of parents and child is
            compared based on blood inheritance
                If the blood group of child doesn't comply
                with the blood inheritance chart, child
                is determined to be non-biological and
                is appended to the list('adopted')
```

4.12 Question 16 & Question 17

Pseudo code for finding out the if the father can donate blood to the son (Question 16):

```
Lines 736 - 771

    Iterate through the keys(Father's CPR) in dict 'Fathers'

        Store father's blood type
        Store  Father's children in a list

        Iterate through the list of children
            Check if the child is a son
                Store son's blood type

                Compare father's and son's bloodtype
                based on blood donor/reciever chart
                in fig 2.

                if father can donate blood to his
                son/sons store the father's CPR with
                bloodtype as key and the son/sons' CPR
                with bloodtype he can donate to as value
                in dictionary 'f_cbloodict'
```

The logic used to find out if the grandchild can donate blood to the grandparents (Question 17) is is the same as in the Question 16 , except instead of using the dict 'fathers' the dict 'grandkids' is used. The code is found in lines 793 - 845

5 Program Design

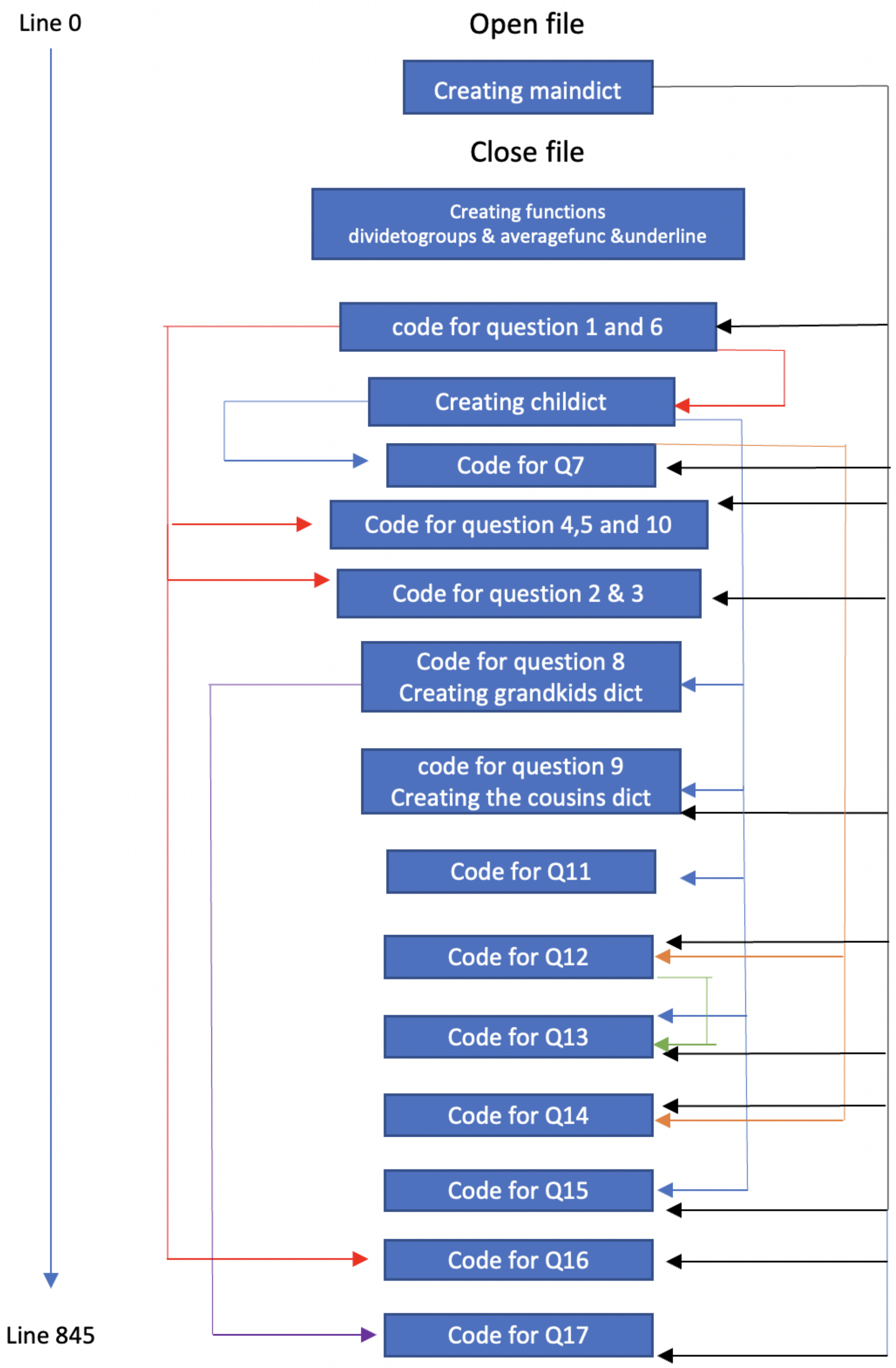

Line 0
Open file
Creating maindict
Close file
Creating functions dividetogroups & averagefunc &underline
code for question 1 and 6
Creating childict
Code for Q7
Code for question 4,5 and 10
Code for question 2 & 3
Code for question 8 Creating grandkids dict
code for question 9 Creating the cousins dict
Code for Q11
Code for Q12
Code for Q13
Code for Q14
Code for Q15
Code for Q16
Code for Q17
Line 845

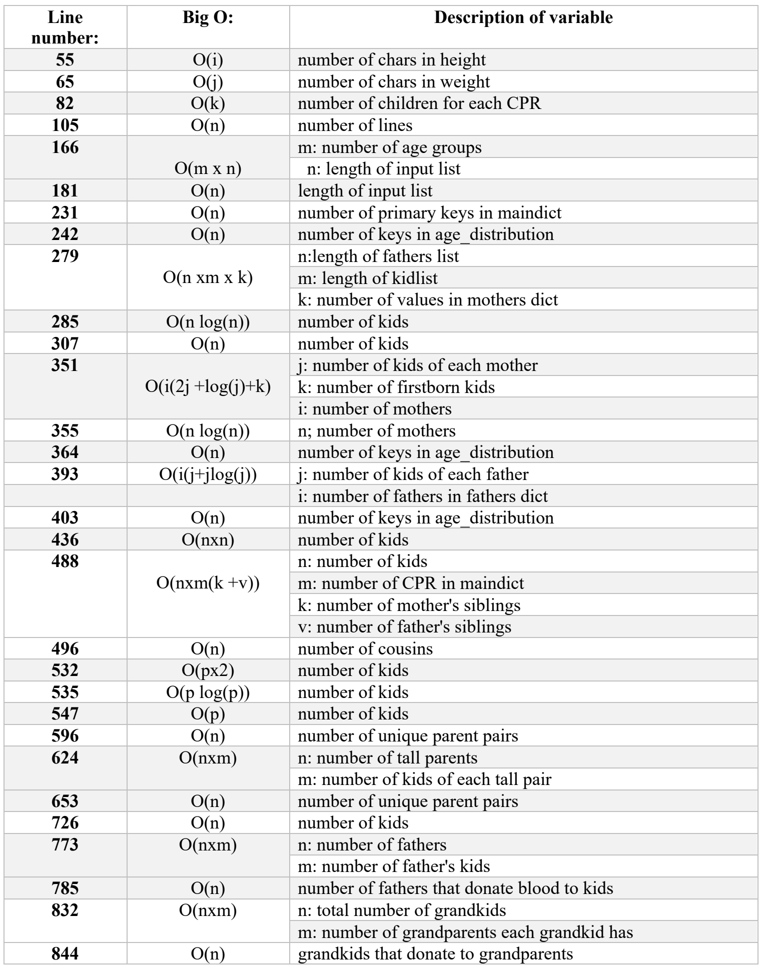
Figure 3: Program Design

The above program design portrays how different elements of the program interact with each other. The blocks show each separate piece of code. This block may represent a data structure that can later be used to solve multiple questions or it can represent a piece of code written to solve a particular question/questions. The arrows indicates if a particular part of a block(piece of code) is used in another questions algorithm and where it is being used. The program design also shows the general direction in which the code runs.

6 Program Manual

The user of the program is required to store the program file and the .db file in the same folder of the computer. Use the commands chmod a+x < *filename* > and ./ < *filename.py* > in terminal. The output will be presented on the terminal itself

7 Run Time Analysis



Line number:	Big O:	Description of variable
55	O(i)	number of chars in height
65	O(j)	number of chars in weight
82	O(k)	number of children for each CPR
105	O(n)	number of lines
166	O(m x n)	m: number of age groups
		n: length of input list
181	O(n)	length of input list
231	O(n)	number of primary keys in maindict
242	O(n)	number of keys in age_distribution
279	O(n xm x k)	n:length of fathers list
		m: length of kidlist
		k: number of values in mothers dict
285	O(n log(n))	number of kids
307	O(n)	number of kids
351	O(i(2j +log(j)+k)	j: number of kids of each mother
		k: number of firstborn kids
		i: number of mothers
355	O(n log(n))	n; number of mothers
364	O(n)	number of keys in age_distribution
393	O(i(j+jlog(j))	j: number of kids of each father
		i: number of fathers in fathers dict
403	O(n)	number of keys in age_distribution
436	O(nxn)	number of kids
488	O(nxm(k +v))	n: number of kids
		m: number of CPR in maindict
		k: number of mother's siblings
		v: number of father's siblings
496	O(n)	number of cousins
532	O(px2)	number of kids
535	O(p log(p))	number of kids
547	O(p)	number of kids
596	O(n)	number of unique parent pairs
624	O(nxm)	n: number of tall parents
		m: number of kids of each tall pair
653	O(n)	number of unique parent pairs
726	O(n)	number of kids
773	O(nxm)	n: number of fathers
		m: number of father's kids
785	O(n)	number of fathers that donate blood to kids
832	O(nxm)	n: total number of grandkids
		m: number of grandparents each grandkid has
844	O(n)	grandkids that donate to grandparents

Figure 4: Big O calculations in the code

8 Conclusion

8.1 Analysis of results:

8.1.1 Gender distribution:

Percentage of men: 48.6 %
Percentage of women: 51.4 %

The gender distribution, written above, appears to be normal, as it doesn't deviate significantly from the expected one (1:1).

8.1.2 Age distribution:

Age(years)	Percentage
10-15	6.2 %
15-20	4.2 %
20-25	5.2%
25-30	9.0%
30-35	8.8%
35-40	6.8%
40-45	3.2%
45-50	5.2%
50-55	8.8%
55-60	5.6%
60-65	4.6%
65-70	1.8%
70-75	2.8%
75-80	3.4%
80-85	3.4%
85-90	4.4%
90-95	1.0%

Table 1: Age distribution of CPR's in the database

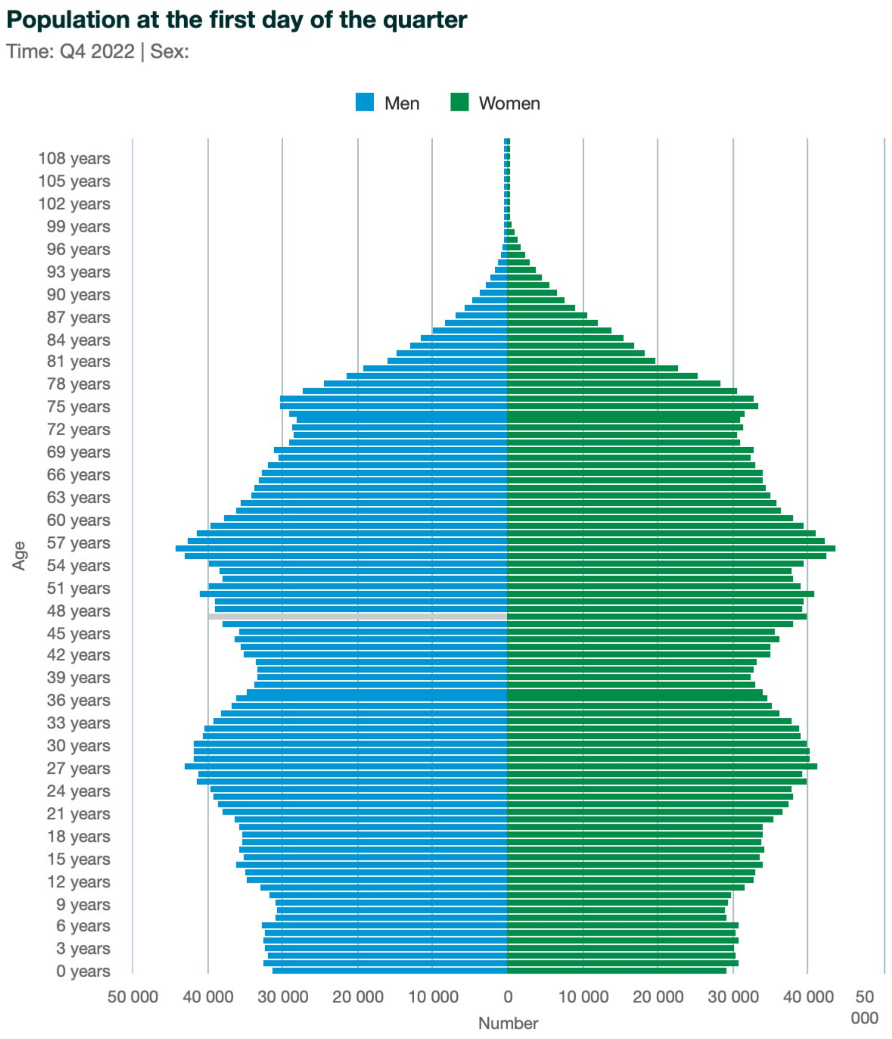


Figure 5: Age distribution of population [10]

Having observed our age-distribution data and having compared them to the ones provided in STATISTICS DENMARK, we came to the conclusion that they are reasonable enough. More specifically, two peaks can be observed in the age groups 25-30 and 50-55, while the ages range, approximately, from 10 to 95 years old. Those facts agree with the official statistics mentioned before. However, it is worth mentioning that we didn't detect people younger than 10 years old, which doesn't agree with the figure above.

8.1.3 First time fatherhood/motherhood:

Age of first time motherhood	Percentage
15-20	5.932203389830509%
20-25	61.86440677966102%
25-30	29.661016949152543%
30-35	2.5423728813559324%

Table 2: Age distribution of people that became mothers for the first time

Age of first time fatherhood	Percentage
15-20	14.40677966101695%
20-25	54.23728813559322%
25-30	28.8135593220339%
30-35	2.542372881355932%

Table 3: Age distribution of people that became fathers for the first time

The first time motherhood and fatherhood distribution, seems to be sensible, taking into account that the ages of fist time fathers/mothers range approximately from 15 to 35 years old. It is worth noting that the majority of people, become parents of their first child at an age between 20-25 years old.

8.1.4 Gender of firstborn:

The likelihood of the fistborn being a male is 45.925925925925924 %
The likelihood of the fistborn being a female is 54.074074074074076 %

As it can be seen above, the likelihood of the first born being a female is bigger compared t the likelihood of it being a male. Nevertheless, the difference is slight, as it was expected.

8.1.5 Marriage between tall people:

The percentage of couples consisting of two tall people is 0.11016949152542373 %, out of all couples.
Out of all couples with at least one tall partner, couples with 2 tall partners are 17.56756756756757 %.

It seems that couples consisting of two tall people constitute a small minority of the existing couples. From the second percentage, it is understood that tall people rarely marry other tall people.

8.1.6 Marriage between fat people:

The percentage of couples consisting of two fat people is 0.1694915254237288 %, out of all couples.
Out of all couples with at least one fat partner, couples with 2 fat partners are 24.096385542168676 %.

It is obvious that couples consisting of two fat people account for the minority of the existing couples. From the second percentage, it is understood that fat people do not often marry other fat people.

8.1.7 Children of tall people:

The percentage of kids of tall parents that are tall is 36.666666666666664 %.

The aforementioned percentage, shows that more than 1 out of 3 kids of tall parents' kids, will be tall.

8.1.8 Fathers donating blood to sons:

Fathers that can donate blood to at least one of their sons are 50.
Sons that can receive blood from their father are 67.

8.1.9 Grandkids donating blood to at least one grandparent:

The number of grandkids that can donate blood to at least one grandparent is 132.
The number of grandparents that can receive blood from their grandkids is 176.

8.2 Weaknesses and future improvements

The program developed, creates multiple data structures (dictionaries) to answer the questions. Using this method of answering the questions, the advantage is that the program doesn't need to iterate many times through the same data structure, thus making the code for certain questions a lot easier to execute. The disadvantage of this is that if something is overlooked while creating one of the data structures, the program will not run accurately. Another point to consider is that in the database given to develop the program, there weren't any observable people that had children with multiple partners. Because of this, the questions that required us to answer based on this phenomenon, may need more rigorous testing with a database that does contain people that have had children with multiple partners. This conclusion can however be made for the code in general as it was made based on 1 example of a .db file. If many different .db files were used to help develop the program, the program's robustness would definitely be a lot better. Hence, using multiple databases to test the program could also be a good next step for future improvements. The program is made to run through a database only specific to Denmark(CPR numbers) and cannot be used for databases in other countries. The variable names at certain places could also be a bit confusing for going through the code.

References

[1] Bmi. `https://www.psychologytoday.com/us/blog/eating-disorders-news/201503/the-new-improved-bmi`.

[2] Abo inheritance. `https://learn.genetics.utah.edu/content/basics/blood`.

[3] Rhesus determination. `https://www.nhs.uk/conditions/blood-groups/`.

[4] OneBlood. Blood type inheritance. `https://www.oneblood.org/media/blog/target-your-type/how-do-blood-types-work.stml`.

[5] Rhesus inheritance. `https://stanfordbloodcenter.org/can-two-rh-positive-parents-have-an-rh-negative-child/`.

[6] incompatibility. `https://www.bswhealth.com/locations/temple-blood-center/blood-type-genetics-and-compatibility`.

[7] Blood types. `https://www.oneblood.org/media/blog/target-your-type/how-do-blood-types-work.stml`.

[8] Rhesus compatibility. `https://www.redcrossblood.org/local-homepage/news/article/what-is-the-rh-factor--why-is-it-important-.html`.

[9] NZBlood. Blood donor/receiver chart. `https://www.nzblood.co.nz/about-blood/`.

[10] Statistics denmarks. `https://www.dst.dk/en/Statistik/emner/borgere/befolkning/befolkningstal`,.