1. You would like to create a new dataset (using R code) that contains the unique patients who appear in both datasets. Note that both datasets have two columns (A and B) in common i.e. columns representing the first name and the last name. However, the spellings/text of these column names don’t match.

A diagram of a patient's record

Description automatically generated

Here the left is bill.data while on the right is vital.data. The question wants us to make a new record with the list of first name and last name, but the condition is that person has to be on both bill and vital data set. So, if we combine those two dataset, this is the inner join we get, and the inner join will consist of the common attributes that are available in both the datasets with first name and last name. So, we people like Bill Walsh, Joe Smith that are in both dataset will fall in the common inner join. We will be using the merge function to join it.

The output would be:

A screen shot of a number

Description automatically generated

1. You would like to create a new dataset (using R code) that contains all the patients appearing in either dataset.

In this dataset, we've included all information from both the "bill" and "vital" datasets, including their common elements. In simpler terms, it's the combination of the "bill" dataset and the "vital" dataset, encompassing all data from both sources.We will be using the merge function to join it but all will be true condition.

A diagram of a common data

Description automatically generated

Here is the output of the data:

A close-up of a computer screen

Description automatically generated

1. Write R code to find patients records where insurance starts with characters “Se" in the merged dataset generated in part a.

We utilize the subset function in conjunction with the grepl function to identify patient records within the merged dataset from question a, where the insurance information starts with the characters "Se." This approach is chosen to filter and extract specific subsets of data based on a defined criterion. The grepl function, with the regular expression ‘Se’, performs a pattern match to identify insurance entries that commence with "Se," disregarding case sensitivity by (ignore.case=TRUE). The resulting ***subsetSE*** dataset contains records meeting this criterion, providing a focused subset of the merged data that specifically relates to patients with insurance starting with "Se." This targeted extraction allows for efficient analysis or reporting tailored to this particular subset of interest.

Here is the output:



1. Write R code to compute overall (combined) mean of diastolic and systolic blood pressure using the merged dataset generated in part a.

Here we want the overall mean of diastolic and systolic blood pressure using the merged dataset generated earlier. The mean function is employed to calculate the mean values, and the na.rm = TRUE argument ensures that any missing values are excluded from the calculation. By separately calculating the mean values for systolic and diastolic blood pressure, we gain insights into the average levels of these vital health indicators across the entire dataset. This information is valuable for obtaining a comprehensive overview of the combined blood pressure measures in the merged dataset, aiding in the assessment of the general cardiovascular health represented in the data. The resulting variables, meanSystolic and meanDiastolic, encapsulate these computed mean values, facilitating further analysis or reporting on the overall blood pressure trends within the merged dataset.

Here is the output:

A close-up of a white background

Description automatically generated

1. Write R code to delete the last row in the merged dataset generated in part a. (Don’t use any hard-coded index in the R code)

Here we want to remove the last row from the merged dataset generated earlier. By using the approach of indexing, specifically innerjoin[1:(nrow(innerjoin)-1), ], we ensure the deletion of the last row without relying on hardcoded indices. Consequently, the modified innerjoin dataset reflects the removal of the last row. Here I have used dim(innerjoin) before where the dimensions are 8 12 to check if the code successfully deletes the last row or not.

Here is the output  
A white background with black text

Description automatically generated

1. Write R code to create another column “YearVisited”, extract the year information (for example 14 from 4/18/14) from the “date” column, and store the year information in the newly created column “YearVisited”. Use the data generated in part e.

Here we are creating  a new column named "YearVisited" in the dataset generated in part e. This column is designed to store the extracted year information from the existing "Date" column. Leveraging the format function and converting the "Date" column to a date format using as.Date, the code accurately captures the year portion of each date and stores it in the newly created "YearVisited" column. The % symbols and the m/%d format are used to specify the structure of the date. The % symbols act as placeholders that indicate different components of the date (like month, day, or year). In format="%m/%d/%y", %m represents the month, %d represents the day, and %y represents the two-digit year. So, if we have a date like "4/18/14," the code interprets the format as month/day/year and extracts the corresponding values.

Here is the output:

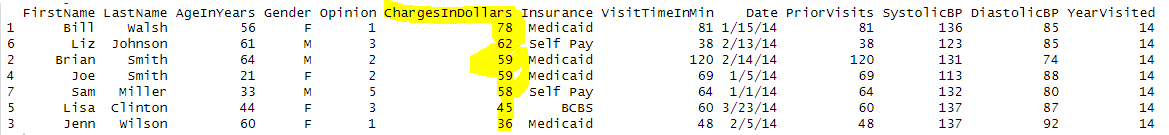
A screenshot of a computer

Description automatically generated

1. Sort the data by ChargesInDollars (in descending order) using the data generated in part f.

Here we are supposed to sort the dataset generated in part f based on the "ChargesInDollars" column in descending order. By using the order function with -innerjoin$ChargesInDollars, we instruct R to arrange the rows of the dataset in a way that the "ChargesInDollars" values are in decreasing order. This operation is valuable for organizing the data, allowing us to quickly identify and analyze the highest charges.

Here is the output:



1. Find unique values of the insurance type using the data generated in part g.

Here we are identifying the distinct values present in the "Insurance" column within the dataset generated in part g. As  the name used in g is also data manipulated in innerjoin, here also we will work data in innerjoin. By employing the unique function on innerjoin$Insurance, the code efficiently extracts and lists unique insurance types present in the dataset. This operation is crucial for gaining an understanding of the diversity of insurance providers represented in the data. Identifying unique values in the "Insurance" column allows for a comprehensive overview, aiding in subsequent analyses or categorizations based on the different insurance types.

Here is the output:

A close up of words

Description automatically generated

1. Write R code to delete the columns – PriorVisits and Opinion in the dataset generated in part g. (Don’t use any hard-coded index in the R code)

Here we are removing the "PriorVisits" and "Opinion," from the dataset generated in part g. By utilizing the logical condition !names(innerjoin) %in% c("PriorVisits", "Opinion"), the code ensures that the removal is not based on hard-coded indices, enhancing adaptability to changes in the dataset's structure. The %in% operator is used to test whether elements on the left-hand side are members of the set on the right-hand side. In this case, it checks if the column names in names(innerjoin) are in the set containing "PriorVisits" and "Opinion.". The ! (logical NOT) negates the result of the %in% operation. So, it checks for column names that are NOT in the specified set mentioned inside the c.  We could use index as well but due to question we will be using the name of the column.

Here is the output:

A close-up of a computer screen

Description automatically generated