Video: https://youtu.be/r7TtgnWqm9Y

At our first group meeting on the 15th March, we split our project into specific tasks, and then divided those projects among the team. The code would be stored in a SQL database, with the code to create and read it written by Timofejs. The user would input queries using a control panel consisting of buttons, toggles, and dropdowns, created by Beatrice. The code would be displayed in two ways: graphs, developed by Liam, or a spreadsheet, made by Shengxin. We established a schedule of meetings every Monday to discuss progress and plan for the next week. To handle communication outside of meetings, we created a Discord server, which is a chatroom with different channels for discussing features, project specs, and asking for assistance. This model enabled us to easily agree on functionality and quickly solve problems that arose during development.

Liam

To make functional bar charts, I evaluated a variety of third-party libraries and settled on giCentre Utils. Their classes allowed for creating customisable charts, handling the complexities of axes and labelling. To handle bar charts, I created my own class, `Bar`, which takes two parameters: a query, e.g. `"ORIGIN"`, and an integer that filters the data according to various properties, such as cancelled flights. In essence, the class works by querying the data, creating a new instance of the `BarChart` class from giCentre, creating page turn buttons, and then providing methods for team members to draw the chart, change the displayed page, and destroy the page buttons. Pages and page buttons were necessary as the number of labels is usually far too large to display on one chart.

The pie chart class, `Pie`, works similarly - it takes the same parameters for the same reasons, and features methods with the same names that carry out similar tasks. The main difference is that giCentre does not provide a pie chart class, so I based my code off of this example from the Processing website. The code creates a series of arcs, proportioned and positioned to form the chart, and coloured according to a list of 64 hex codes chosen to be as dissimilar as possible.

The scatter plot class, `Scatter`, uses the `ScatterPlot` class from giCentre. In addition to a query and an integer that filters the data, this class also takes a parameter that allows the user to bin the data into groups, each represented by a point on the scatter plot. Passing a value of `1` results in no binning.

I made a heatmap class called `Heat`. This is much more limited in scope compared to the other classes - its only purpose is to graph the frequency of flights out of specific states. This takes a parameter to filter the data, and contains methods to query and draw the data. To make this, I used an SVG file with different IDs for each state, allowing me to use `PShape.getChild()` to fill each state individually.

I also created the class `Descriptive`. The class currently holds simplistic methods for finding the largest value, smallest value, mean, median and mode. It also contains a couple methods used by my chart classes to correctly query the data.

Beatrice

I was in charge of the control panel. Our idea was to divide the screen in two parts, the one on the right (¼ of the total screen size) was the control panel and the one on the left would show the graphs or the spreadsheet based on the user selection. The user, through buttons and dropdown lists, can sort the data by distance or lateness, view only late, cancelled or on time flights. Moreover, they can select the departure and arrival airport and the date range. I've also implemented a map, therefore if the user selects the "`Map`" option, they can view the departure and/or arrival flights from that specific airport. To make the graphs/spreadsheet appear on the screen, a "`Send query`" button is pressed. If

the user wants to make a different selection, they will click on the "`Make another selection`" button, the left part of the screen will go back to being blank and the program will be ready to take another input.

To design the buttons, I took inspiration from the widget class I did in the first half of the module. However, since multiple selection is possible, I used an array of booleans to make the program understand which levers were selected. I created a `Button` class that has a method draw that draws the buttons; a setColor method that determines if the knob has been clicked twice, changes the background colour if the button is clicked once and deselects it if it was clicked twice or if another widget in the same line is chosen; a pressed method which, based on the event returned when the mouse is clicked on the button, assigns true to the corresponding case. In the event "`Send`", the graphs are drawn and the dropdown lists get closed. In the event "`Go back`", all the buttons are set to false and the dropdown lists are initialised.

When the map is called, the program will draw small clickable squares on each state. The knobs are designed in the `State` class, which has the same structure and functionalities of the `Button` class, with some differences. The main differences are that the user can select only one state at a time and that the spreadsheet is not called directly (like the graphs are called in the Button class); when the knob is clicked, the program will assign the code of the airports to a String `airports`, which will be used in the Data Piping to call the spreadsheet.

The `ControlPanel` class is my " main ", the `Button` and `States` classes are called and all the widgets are instantiated, the dropdown lists are initialised, using the controlP5 library, and I wrote a method to customise the aspect of the list. I added the texts in the `draw` method, while the buttons and the states are drawn in the main.

Initially, I was supposed to add a feature that allowed the user to select a specific flight, however, eventually, Shengxin took care of that, while I proposed the idea of having the map and I implemented my suggestion. I also added a plane image (after the in class-presentation, while I was fixing some bugs) that appears when the graph page is loading.
I didn't encounter any specific problem while doing my part. At the beginning it was a little challenging to understand how to start doing everything, but after planning how I wanted to structure the code, everything went smoothly.

Shengxin

I am responsible for spreadsheets, the approximate interface design of a spreadsheet is to have an overview main spreadsheet and a subSpreadsheet which is a single flight. And implement functions such as page buttons by using controlP5 library with the guide of my team, a search bar for the specification of a single flight and data summary illustration. With the help of Tim's dataBase and dataPiping classes, the spreadsheet can load all flights in multi-queries, users can explore the rest of the filghts by turning the "page+" button and turn to previous page by "page-", resetting pages to page1 by the "reset page", inputting row number to get details about a single flight. The display between two spreadsheets includes the number ofthe page left, total pages, current page, total flights and current row number.

To implement line drawing of a spreadsheet, with our demonstrator and team members' help, my idea cames out that draw rectangles as a 2D array-cells in the draw function of the class. Limit the quantity per page by setting row and column. For loading data, I create a n by 18 2D-array(n is the total flight) to store data in the data array. For selecting specific columns of original data, I create another new-array to store the selected column from the data array. The single flight information is also a n by 18 2D-fullDataArray. Then draw them when 2D index of rectangles drawing is larger than

[0][0] because row 0 and column 0 are used to draw headers and row numbers. To turn the current page to the next page or back to the last page, I used a variable called baseRowNum in the draw so that it will update when the mouse function change page number so that the draw function will draw the right range of rows on the current page. This algorithm would be related to drawing the row index in the main spreadsheet. To locate the specific flight, my key input is based on key in the processing website. The key input will directly change the row number in sub spreadsheet to implement the search function.

```
int pages = totalRows / rowsPerPage;
if (totalRows % rowsPerPage != 0)
  pages++;
if (pageNum < pages) {
  pageNum++;
}
```

Figure 1.0

```
for (int j = 0; j<=fullHeaders.length; j++)
{
  if ((j > 0)&&(fullData.length>0)) {
    fill(0);
    textSize(12);
    text(fullData[rowNumber-1][j-1], 40 + pos
  }
}
```

Figure 1.1

```
else if (key <= '9'
&& key >='0')
{
  inputST += key;
  input = int(inputST);
```

Figure 1.2

Some problems I encountered in the spreadsheet. Due to a mistake in the algorithm, the page number cannot illustrate correctly, it would be less 1 than the actual page number when pages are not totally full filled. Adding an if statement in the draw to update the illustration when the page changes to deal with it. Another issue in sub spreadsheet that the data[][] nullException would be thrown out if the single flight does not exist, I restored data[][] by using a fullData [][] and adding if statements in the fullData[][] draw so that if data[][] is null, fullData[][] will not throw nullException and the spreadsheet data text will be empty. The case about the search bar was it cannot limit input type and get numbers as integers. According to the reference in the processing website, key input is a char type based on ASCII code, the example in the website uses to limit the characters so I think I could use the same way to limit number input as well, so I used int() to convert char to int, and added if statements to limit input type and enables input of numbers greater than one digit by an algorithm. Hence I solved that problem(as figures illustrate above and reference in the links).

Timofejs

My job was creating the database and providing a class and methods for my team members to access it. I decided to use a SQL database, which is preprocessed, i.e. it is created on first launch and then does not need to be regenerated every time the program is started. This results in significantly improved performance relative to other methods.

I created a class, `DataBaseInteraction`, and my teammates used this to access the database. One method, `getFilteredFlightData`, was used to create ResultSets by various graphs that were then manipulated to get the data into a workable format. I was also on hand to help out my teammates in using this code through our Discord.