ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Data Visualization

Process book

Group 75

Hani Boudabous
228133

Hedi Driss
240582

Hugo Bonnome
227370

December 22, 2017

Contents

1 Introduction

From the 20th century, a new revolutionary way of transportation appeared, airplanes. As with every human invention, it is not perfectly made and there can be accidents, leading to the crash of the plane and unfortunately in this case, a probable death for the humans on board.
To see what are the main causes of airplane crashes, we drew a first draft shown below of what our visualization would look.

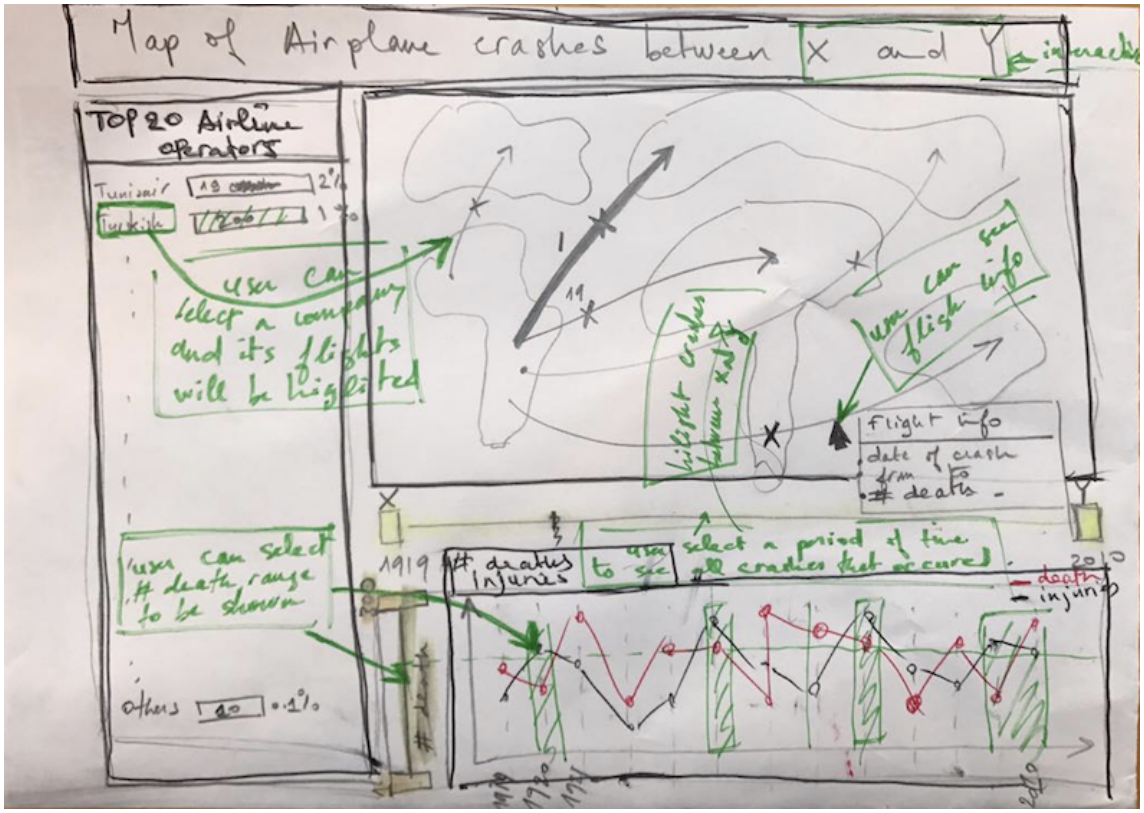

Figure 1: Draft of the visualization

2 Dataset and Preprocessing

2.1 Data description

The dataset used is taken from *Socrata*[1] website. It contains all the crashes (more than 5000) that happened since 1908. Every data sample (airplane Crash) is described by multiple features. We only kept the features that are going to be used for the map and charts visualization as the *crash location*, *crash year*, the *number of deaths*, the *airline company* ...

2.2 Data analysis and preprocessing

To analyze and preprocess the data, we use the scripting language Python. The data is read from a Csv file into a Data Frame, a data structure from a python library named Pandas.

[1]Source : https://opendata.socrata.com/Government/Airplane-Crashes-and-Fatalities-Since-1908/q2te-8cvqh

2.2.1 Geolocalisation

The first thing we did was to find the crashes coordinates. This part is crucial as it allow us to precisely draw the crash points in our map. To do so we used Google API, that has proven to be very effective for this task. From a 'city, country' input format, it returns many informations on the entry in a JSON format from which we only take latitude and longitude coordinates. if the API has not been found, it returns Nan value that we drop after using the dropna() function for the 'location' (name of the location) and 'LatLongCrash' (coordinates of the crash) features. We were able to get the geolocalisation of more than 95% (almost 4900) of the crashes.

2.2.2 Crash Route

After that, as shown in the draft, we had to find the route (the take off and supposed landing location of the plane). This task is easy when there is no transits in the the plane route but becomes less obvious when the plane has to stop somewhere. The route of the plane is specified in the Data Frame as follows :

	Location	Route
0	Pokhara, Nepal	Jomsom - Pokhara
1	Near Ayan, Russia	Khabarovsk - Ayan
2	Lexington, Kentucky	Marco, Fl - Lexington, KY
3	Rio Branco, Brazil	Cruzeiro do Sul - Tarauaca - Rio Branco

Figure 2: Location and Route features from the crashed Data Frame

To solve this problem we had to design an algorithm which would return the route where the crash happened. The steps are below : Loop on every route and see if there is a transit location. if not just return the route; else find the two closest locations in the route to the crash location. We used the geopy library to compute the distances. every step of the algorithm will be commented and detailed in the python code.

This is how we proceeded for the map visualization. Let us now explain briefly what are the preprocessing steps for both the interactive line chart and table.

2.2.3 line chart and table preprocessing

For the table in figure 3, it shows the number of crashes for the selected year range by company. Hence we need to group the data by company and do the sum of all fatalities for every company. About the line chart of figure 4, we just grouped by year and did the sum of the number of crashed airplanes for every year.

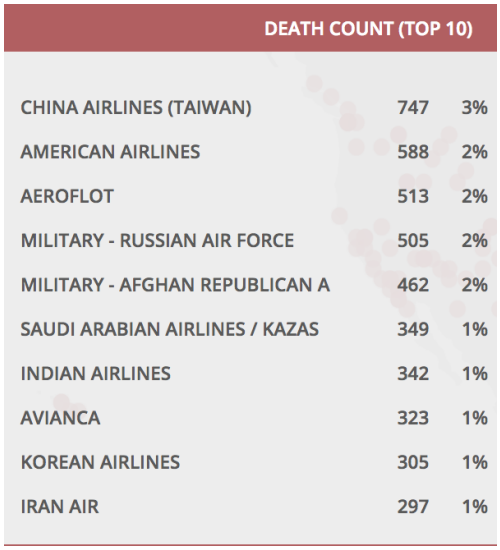

Figure 3: Death per airline Company table

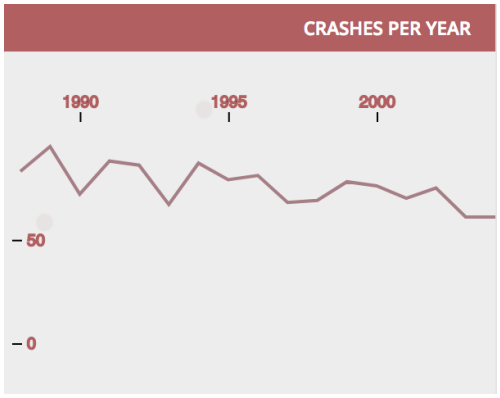

Figure 4: Number of Crashes per year Line chart

3 Visualizations

3.1 Goal of the visualization

The goal of the visualization is to find hidden patterns in the airplane crashes by looking to the different crash information for different year ranges.

3.2 Map visualization

3.2.1 Description

It is an interactive representation of all crashes that occurred in a certain year range. Once the user select a year range, the map is populated with markers on every crash location. The map visualization is the principal visualization. It occupies the totality of the window. The menu containing the Line chart (see next section) can be opened whenever the user drags the mouse to the left border of the map. It is opened from the left side of the map with transparent layout to not entirely hide the part of the map it occupies.

At first, we designed the map in a way to show all details about every crash, but we soon noticed that the big size of the dataset (more than 5000 crash) makes the map overloaded and the user is no longer able to distinguish different crashes' information. Hence, we just kept the crash location as the only primarily point to describe a crash. This way, the view is lighter and the

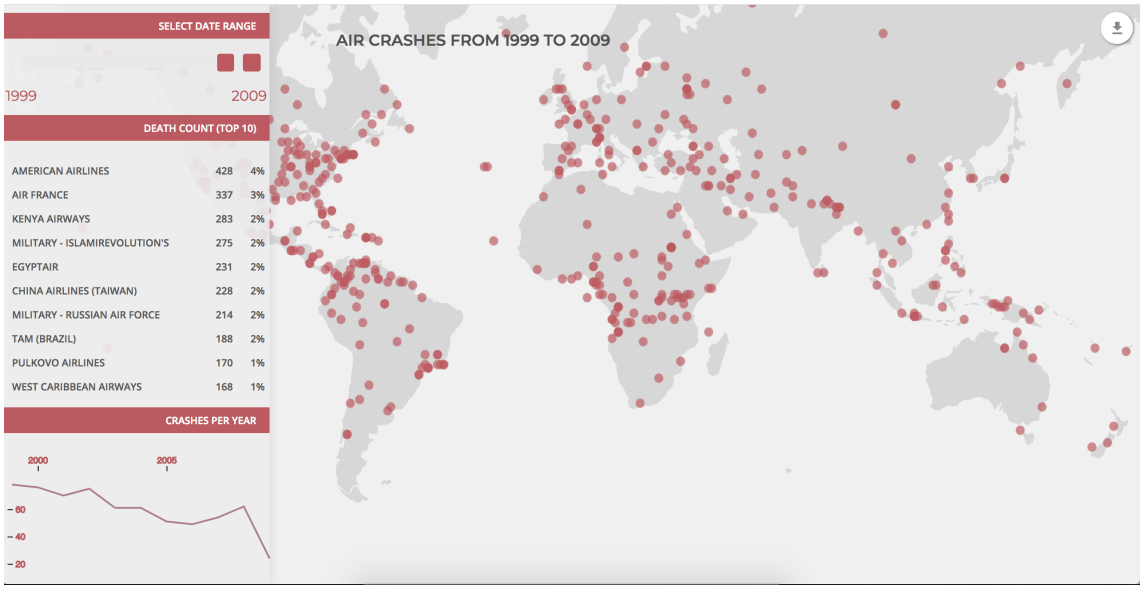

Figure 5: The map shows all the crashes that occurred between 1956 and 2009

different patterns can be spotted from the distribution of the crash points world wide. Therefore, the user can see all crashes locations at the same time and for more information about a crash, s/he needs just to roll over a crash marker to see a detailed description of the corresponding crash according to the available data: departure, destination, location, reason of the crash...

We did several design choices for the map for a better visualization. First, we eliminated all unnecessary details (e.g. different country/city names and labels...) to reduce the cognitive load and enable the user focus only on the important information plotted on the map. Second, we chose a light and reduced opacity color for the map and a brighter color for different markers put on top of the map for a better visibility (red for crash region and white balloon description).

3.2.2 Implementation

We tested 3 APIs for the map visualization:
1. Google maps
2. Leaflet
3. AmCharts

We chose AmCharts essentially because it scales better with a high number of markers and lines on the map. Also, it presents an advanced image animations with just a few configuration options. In fact, the map take care of different details: speed, size, trajectory and the rotation of the image to face the direction it is moving to. For example, to draw a dashed line in Google maps, you need to animate an SVG image along a line which overloads the thread when multiple dashed lines needs to be plotted. Also, animations in AmCharts are much smoother compared to the other APIs.

In AmCharts the map is an instance of the class AmMap. It is constructed with an initial setting where every attribute is configurable with high flexibility.

Lines are instance of class MapLine. A MapLine can be configured by its different properties e.g. position , color, arc, arrow properties...

Images are instance of class MapImage. A MapImage can be saved locally, uploaded via an URL or via svgPath. It can also be configured by its several properties e.g. scale, roll-over color/scale... Ta animate an image along a line, it has to be linked to a line by its id. Also, different animation properties can be configured.

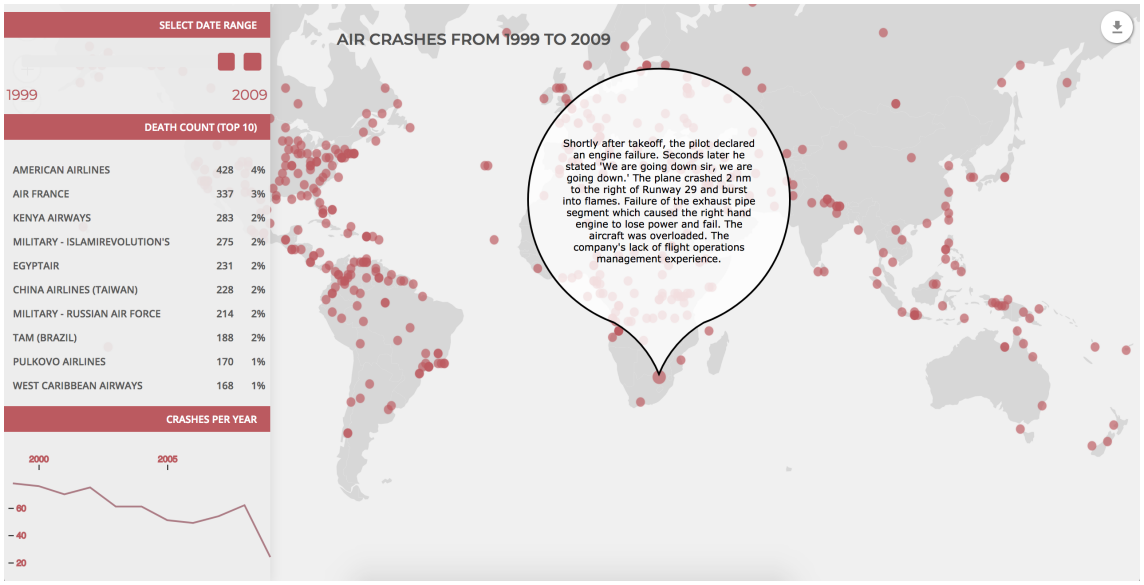

Figure 6: Once the mouse rolls over a crash, a detailed description of the crash is displayed

3.2.3 Evaluation

The main challenge of the map visualization is to reduce the delay of loading and plotting the crashes information for a selected range. In fact, the number of crashes may be big enough to delay the map response for few seconds which is very undesirable. Therefore, we found a good trade-off between the delay and the different features to plot for every crash.

We also optimized our code to reduce calls to high complexity functions as much as possible. For instance, every time a line or an image is added to the map a call to the function map.validateData() is needed in order to update the map view. But since this function is slow, we only call it after a reasonable number of images/lines are added.

This results in a smooth and highly interactive visualization that plots all crash points and load their description in less than 2 seconds and instantaneously for smaller ranges (<30 years).

3.3 Left window

3.3.1 Description

The left window contains:
Range selector: Allows the user select a range to animate all visualization.
Airline death count table: Display the airline companies that made the most deaths in the selected range.
Number of crashes per year line chart: plots the number of crashes for every year in the selected range
The left window was designed in an ergonomic way with the same color palette as the map. A very simple design which is important to display many variables in the same view without overloading the view.

3.3.2 Implementation

Range Slider : The visualisation aim to return many insight about airlines crash on a range of years selected by the end user.
One of the first task was to allow the user to easily indicate a time period. We used a jquery range slider to handle this task.

Total crashes per year :The visualisation required to return the number of death each year on

the selected range in order to see the global trends. Therefore, We choose to implement the display using a line chart. D3 was used to build the visualisation. The data was grouped by year and aggregated on the total death column reducing the value using the sum function.

Total Death per operator : We was motivated to exposed the operators with the biggest number of death in the selected range.
This is why we wanted to create a reactive table to display the body count and the overall death percentage of the operator.
The data was grouped by operator, aggregated on the total death column reducing the value using the sum function and finally sorted in a descending way. We create a javascript function to easily allow to update the view.

Controler : One of the requirement made during the meet up was to display all the variables at once. We opted for a left aside menu containing the range slider controller, the total crashes per year line chart and the total death per operator table. We put the map in full screen beside the menu and unified the design using burgundy as our main color.

Loading the data : Data is loaded in javascript using the unblocking javascript promise. When the data are downloaded and ready to be used, the promise executes the given callback that just looks to parse the csv rows into a list of javascript objects and to update the view. Make the view react to user event : The model react when the user releases the mouse. If the range has been modified it try to update all the displayed chart to remain consistent with range selected by the end user.

4 Contribution and Peer assessment

4.1 Hani Boudabous

4.1.1 Individual contribution

Map related tasks:
- Test several Map APIs
- Populate map with crash markers, descriptions and animations

4.1.2 Peer assessment

Highly motivated and innovative group.
On time on meetings.
They take decisions after considering all opinions.
Good experience.

4.2 Hedi Driss

4.2.1 Individual contribution

Contributed to building the map and the relevant informations to show:
- Preprocessing tasks
- Line chart

4.2.2 Peer assessment

Everyone did its part and overall satisfied of the team work.

4.3 Hugo Bonomme

4.3.1 Individual contribution

- Made the select range year bar
- Created the skeleton of the website and made everything work together
- Table of crashes per year

4.3.2 Peer assessment

Work distributed equally between the group members.

5 Insights and conclusion

The interactive visualization try to simplify the understanding of the evolution of airplanes crashes through the years from the line chart.
The table gives insights about the evolution of the different operators that are the cause of biggest number of fatalities.
The role of the map is to give geographical intuition of the areas where most crashes happens.
Overall, we tried to build a visualization that simplifies the search of patterns in the crashes for different factors.
We implemented many other visualizations that we could not include (show the taken route for every crash, Simulate the airplane crash) because the visualization was drastically slower and was not user friendly anymore and that is why we did a tradeoff between multiple map functionalities and reactive website.