Project 2 write up

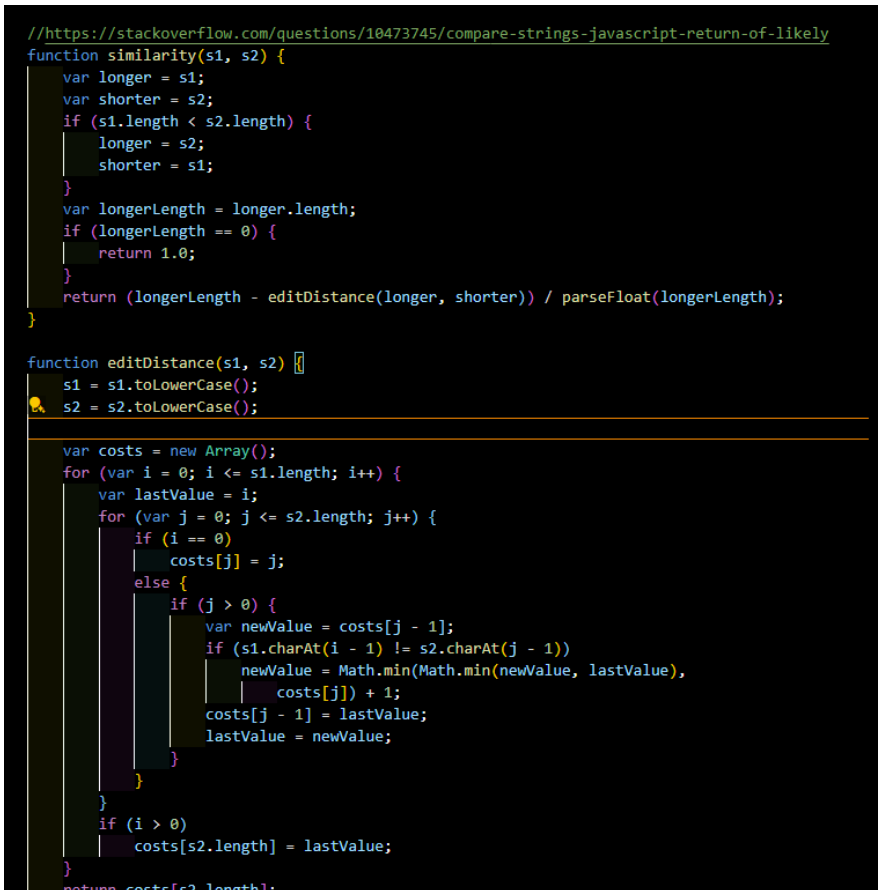
https://timmasso.github.io/IG-Project2-Full/PlaneMap.html

An interesting story I wanted to dive into while watching the data from project 1 is the difference of incidents between operators. I thought to myself, how can I show which operator has more incidents and where they typically happen using data wrapper and maps.

The first step I took was getting more complete data. I saw that the NTSB website gives different information depending on which search query you use. I had used the old search query for my project one as I had not noticed a difference in the data itr gave me but while looking for the phase of flight value in the csv I saw that the old query result did not include it in, but the new CAROL query did. Another problem I ran into was that if I requested too much data in a downloadable csv the site would go into a request loop and never load the csv to download so I had to request the data from 1980 to 2024 in two 20 year requests. Then I just copy and pasted the second into the first csv since they had the same column headers.

Now with the correct csv data I used the same code from project 1 to just find the commercial operators and most used planes. Since I didn't have a list of all the operators I had to alter the code just a little to look for any text in the string, then do the lookling. Then I used the same code to convert it to geojson and then to add the airport lat and lng.

The main obstacle I had to overcome was the issue of counting and separating the operators and what phase of flight each incident happened in. As a datawrapper needs to see things in a specific way. So the goal layout would be | Operator | Approach | landing | take off |... but what we had was | Operator | PhaseOfFlight|. Meaning I had to find a way to create new columns for each phase of flight and also count the number of operator pairs of that phase. Along with the same for injury types per airline. But the format would be similar.

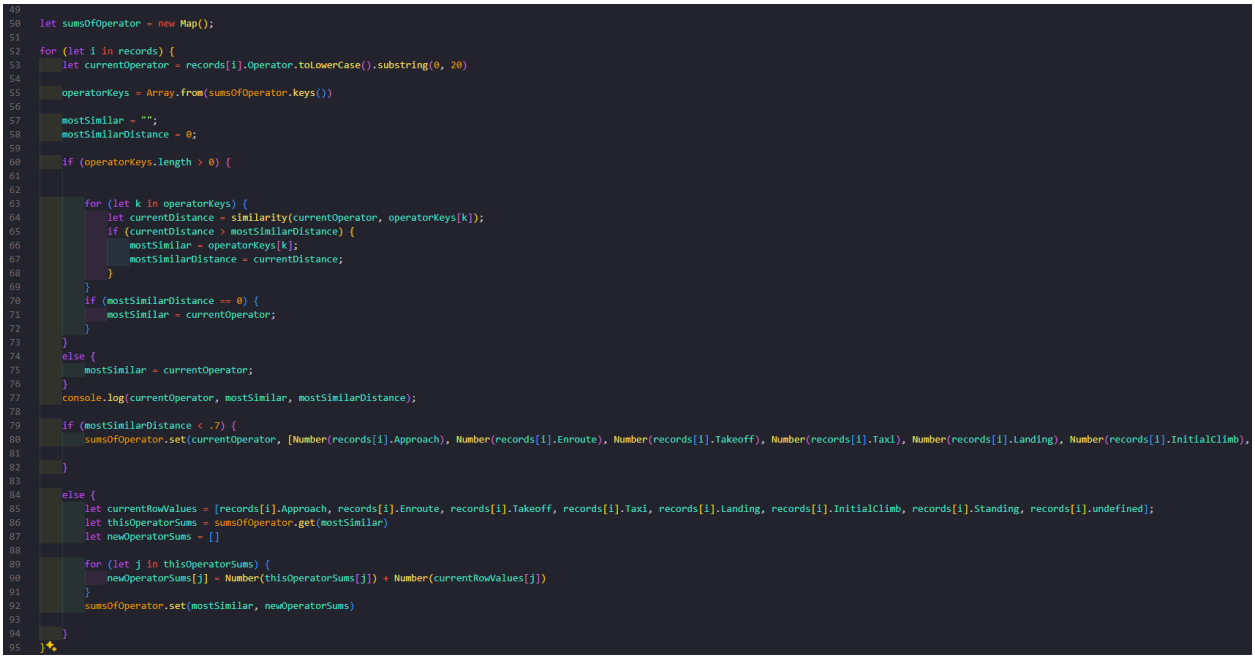


```
//https://stackoverflow.com/questions/10473745/compare-strings-javascript-return-of-likely
function similarity(s1, s2) {
    var longer = s1;
    var shorter = s2;
    if (s1.length < s2.length) {
        longer = s2;
        shorter = s1;
    }
    var longerLength = longer.length;
    if (longerLength == 0) {
        return 1.0;
    }
    return (longerLength - editDistance(longer, shorter)) / parseFloat(longerLength);
}

function editDistance(s1, s2) {
    s1 = s1.toLowerCase();
    s2 = s2.toLowerCase();

    var costs = new Array();
    for (var i = 0; i <= s1.length; i++) {
        var lastValue = i;
        for (var j = 0; j <= s2.length; j++) {
            if (i == 0)
                costs[j] = j;
            else {
                if (j > 0) {
                    var newValue = costs[j - 1];
                    if (s1.charAt(i - 1) != s2.charAt(j - 1))
                        newValue = Math.min(Math.min(newValue, lastValue),
                            costs[j]) + 1;
                    costs[j - 1] = lastValue;
                    lastValue = newValue;
                }
            }
        }
        if (i > 0)
            costs[s2.length] = lastValue;
    }
```

My first conceptual thought on how I would need to approach this problem is that there are some operators that have different names on the reports, so I would need to consolidate those into one name. After looking at how to do that I discovered a formula called Levenshtein distance, which defines how similar two strings are with a number based on how many changes have to be made

to get the target string. With that it would make more sense to create a row for the first such as row 1 would be “United” so then i thought it would be faster to create then compare. So with the first row being “United” if the next operator is “Delta” the number given by the comparison formula would be very close to 0 as the amount of changes would be high so the similarity is very low. Since the string is not the same as the first row it should create a new row with that new operator.

Along with the earlier formula it was advised that I used a storage tool called hashmaps. Where I could store an operator's phases of flight as keys that would make an array of the number of phases of flight in an array to add to it. used for comparing new rows would be saved and referenced. For instance if the first row was | united | landing |. It would add to the array in the hashmap and make it. Where the key would be United and would have an array of the phases of flight, United => [1, 0, 0, 0,] and so on for each operator. Syntax was helped by my brother as I was still new to learning about hashmaps and how to store information in them.



```
let sumsOfOperator = new Map();

for (let i in records) {
    let currentOperator = records[i].Operator.toLowerCase().substring(0, 20)

    operatorKeys = Array.from(sumsOfOperator.keys())

    mostSimilar = "";
    mostSimilarDistance = 0;

    if (operatorKeys.length > 0) {


        for (let k in operatorKeys) {
            let currentDistance = similarity(currentOperator, operatorKeys[k]);
            if (currentDistance > mostSimilarDistance) {
                mostSimilar = operatorKeys[k];
                mostSimilarDistance = currentDistance;
            }
        }
        if (mostSimilarDistance == 0) {
            mostSimilar = currentOperator;
        }
    }
    else {
        mostSimilar = currentOperator;
    }
    console.log(currentOperator, mostSimilar, mostSimilarDistance);

    if (mostSimilarDistance < .7) {
        sumsOfOperator.set(currentOperator, [Number(records[i].Approach), Number(records[i].Enroute), Number(records[i].Takeoff), Number(records[i].Taxi), Number(records[i].Landing), Number(records[i].InitialClimb),

    }

    else {
        let currentRowValues = [records[i].Approach, records[i].Enroute, records[i].Takeoff, records[i].Taxi, records[i].Landing, records[i].InitialClimb, records[i].Standing, records[i].undefined];
        let thisOperatorSums = sumsOfOperator.get(mostSimilar)
        let newOperatorSums = []

        for (let j in thisOperatorSums) {
            newOperatorSums[j] = Number(thisOperatorSums[j]) + Number(currentRowValues[j])
        }
        sumsOfOperator.set(mostSimilar, newOperatorSums)

    }
}
```

One thing that I noticed as I was running is that a change in capitalization is counted in the similarity formula as a change so it would make words look more different to the code. For instance United and UNITED are spelled the same but it would take 5 changes of the capital letter to make the same word and sometimes that added with taking away INC and CO, the code would see it as a new operator. To remedy this before the formula would look at the list it would all be set to lowercase so no capitalizations would need to be done. With that change it worked perfectly. But at the end I would have to manually put in the new CSV headers. To the new list. With the new lists datawrapper can properly take the data and display it how i wanted it.

Now I was thinking about how to display my information on a webpage. I wanted to display the graphs and 4 maps. 3 of which had maps of a single airliner. What about html is that I should do boxes in boxes to make layout easier. I decided to do 4 columns to display the maps. To do this I learned that divs inside divs are relative to the largest div. So I made 4 large column divs and smaller squares inside and inserted the maps into the smaller boxes.

To get the data for just one operator I did the full cycle of what I did to all the data but instead of only doing the code when it sees a value in the the operator column it would only look for listed operators. I reused a lot of my old code but slightly modified it to work with operators.

My findings would be stronger with accurate data sources for total number of passengers per airline and total flights with each and global data. But I can't seem to find a reliable source for that data. A general stat is that there are over a billion passengers and millions of flights every year and with almost each airline having under one hundred incidents in 40 years airline safety is extremely reliable and ultimately comes down to human error.