Google FORMulating Community-Based Empowerment

Adding a Google Form to our mapplication!

Goals

- Create a Google Form with meaningful questions
- Embed a Google Form into a website using an iFrame
- Implement a trigger for geocoding “location” data in Google Sheets

Starting template code for lab #4

Use the following template code or your lab assignment #3a or #3b:

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
```

```
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2023)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

styles/style.css

```
body{
    display: grid;
    grid-auto-rows: auto 1fr;
    grid-template-areas: "header" "main_content" "footer";
    background-color: aqua;
}

header{
    grid-area: header;
}

#footer{
    grid-area: footer;
}

.main{
    grid-area: main_content;
    grid-template-areas: "content" "main_map";
    display: grid;
}

#contents{
    grid-area: content;
}

#the_map{
    height:80vh;
    grid-area: main_map;
}
```

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch(`map.geojson`)
    .then(response => {
        console.log(response)
        return response
    })
    .then(data =>{
        // do something with the data
    })
```

map.geojson

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "place": "home",
        "color": "red"
      },
      "geometry": {
        "coordinates": [
          -118.29687953814576,
          34.061455838557535
        ],
        "type": "Point"
      }
    },
```

```
      {
        "type": "Feature",
        "properties": {
          "place": "work",
          "color": "blue"
        },
        "geometry": {
          "coordinates": [
            -118.43969437158387,
            34.07271277905194
          ],
          "type": "Point"
        }
      },
      {
        "type": "Feature",
        "properties": {
          "place": "old home",
          "color": "red"
        },
        "geometry": {
          "coordinates": [
            -118.43848986633458,
            34.05513005654072
          ],
          "type": "Point"
        }
      },
      {
        "type": "Feature",
        "properties": {
          "place": "metro work",
          "color": "blue"
        },
        "geometry": {
          "coordinates": [
            -118.23503623440803,
            34.055738694402294
          ],
          "type": "Point"
        }
      }
    ]
  }
```

Last update: 2023-04-27

Adding more to our `L.GeoJSON`

Remember that putting a variable into a `type` gives you access to different methods?

Rather than just stopping at `L.geoJSON(data).addTo(map)` we are going to expand that part of the code to style the GeoJSON when we add it!

Clickable GeoJSON recipe

This is the basic Leaflet recipe for a clickable geojson:

```
// the leaflet method for adding a geojson
L.geoJSON(data)
    .bindPopup(layer => {
        return "you clicked a geojson!";
    }).addTo(map);
```

Adding GeoJSON functionality

Now that we have that recipe, we need to put it somewhere… Where is the best place for it?

Answer

```
fetch("map.geojson")
    .then(response => {
        return response.json();
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data) 1
        .bindPopup(layer => {
            return "you clicked a geojson!"; 2
        }).addTo(map); 3
    });
```

1. This is where we added the clickable geoJSON recipe!!
2. Notice we are going to a generic `you clicked a geojson` message here!
3. This is where we add the `GeoJSON` to the map.

Rather than just simply returning the `popup` as a generic `you clicked a geojson`, let's use our GeoJSON's `place` property that we created in the first part of the lab!

Checking our logs!

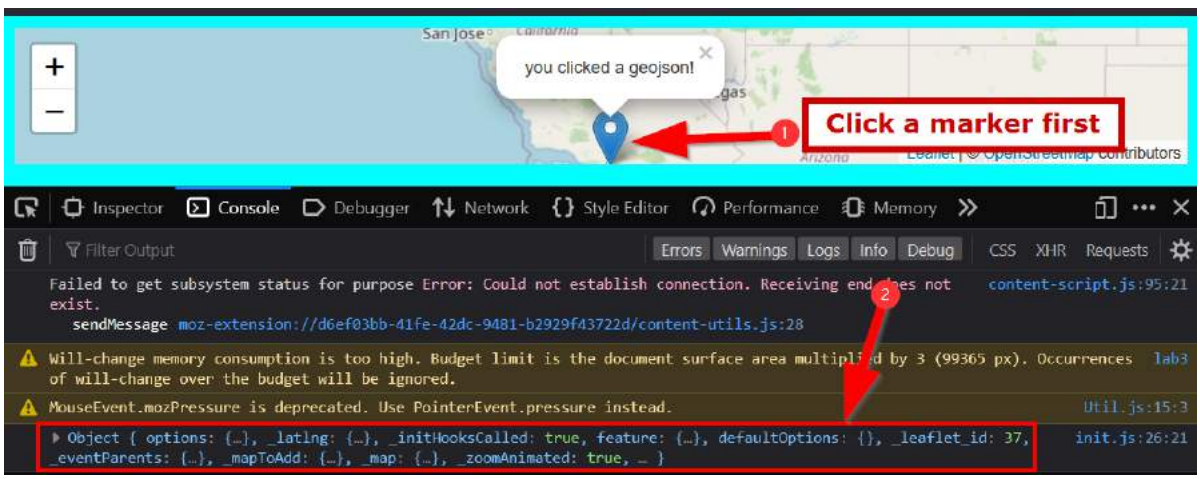
Let's `console.log()` our layer to see how it looks:

Where should the `console.log()` go?

Correct, line 3!

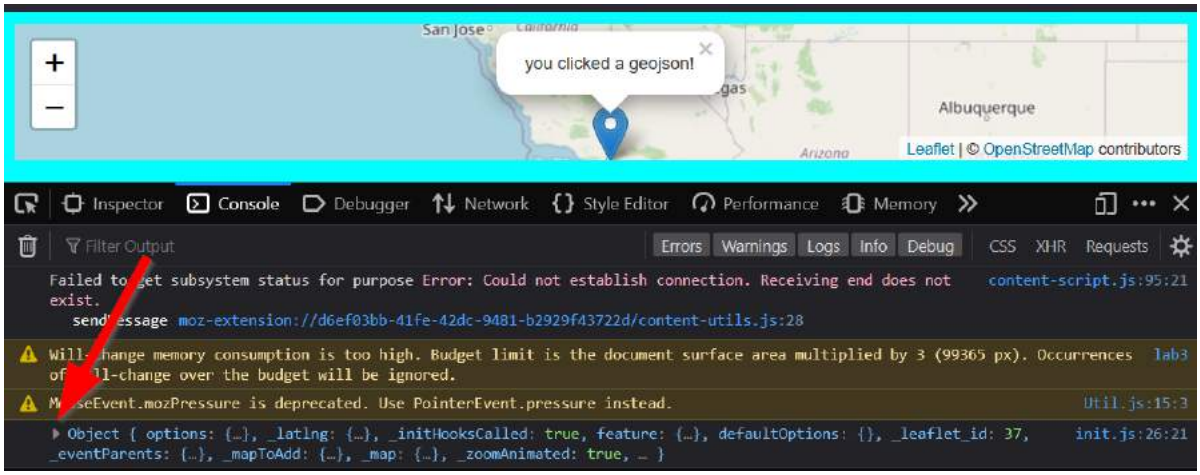
```
L.geoJSON(data,
    }).bindPopup(layer => {
            console.log(layer)
            return "you clicked a geojson!"
    }).addTo(map);
```

Now when you click the marker, this should pop-up in the console:

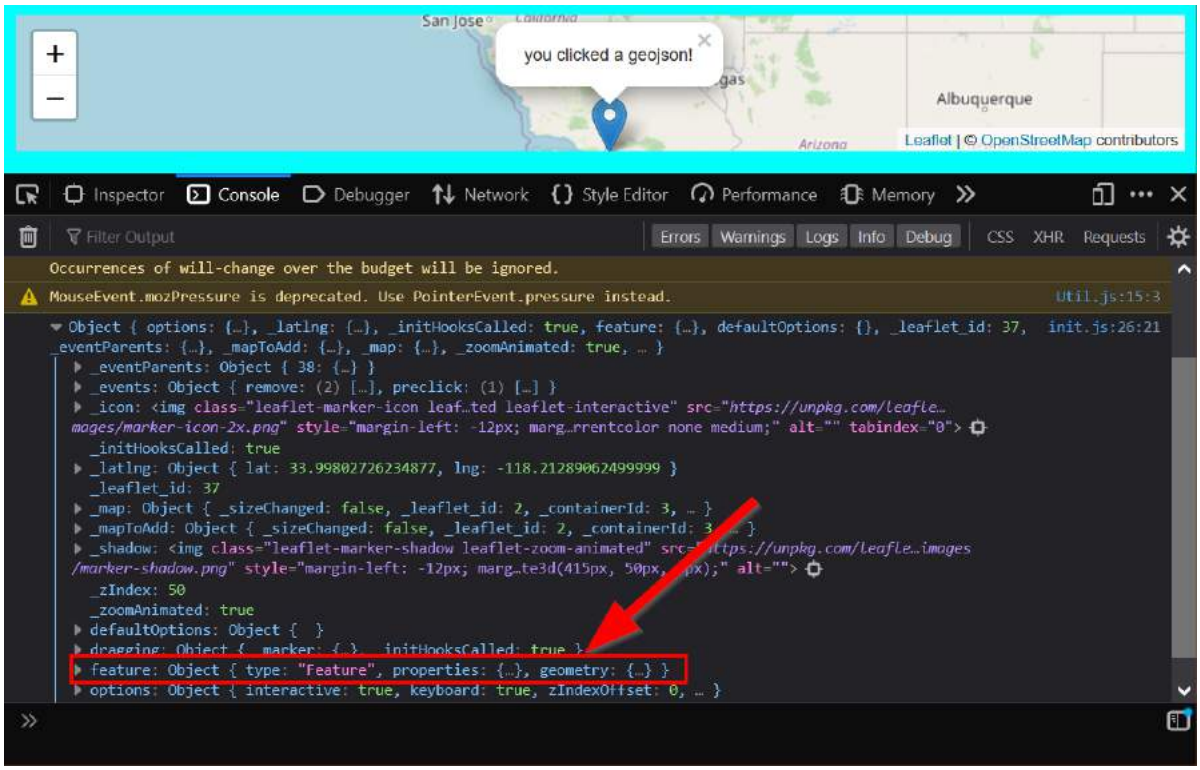



```
Object { options: {…}, _latlng: {…}, _initHooksCalled: true, feature: {…},
defaultOptions: {}, _leaflet_id: 37, _eventParents: {…}, _mapToAdd: {…}, _map:
{…}, _zoomAnimated: true, … }
```

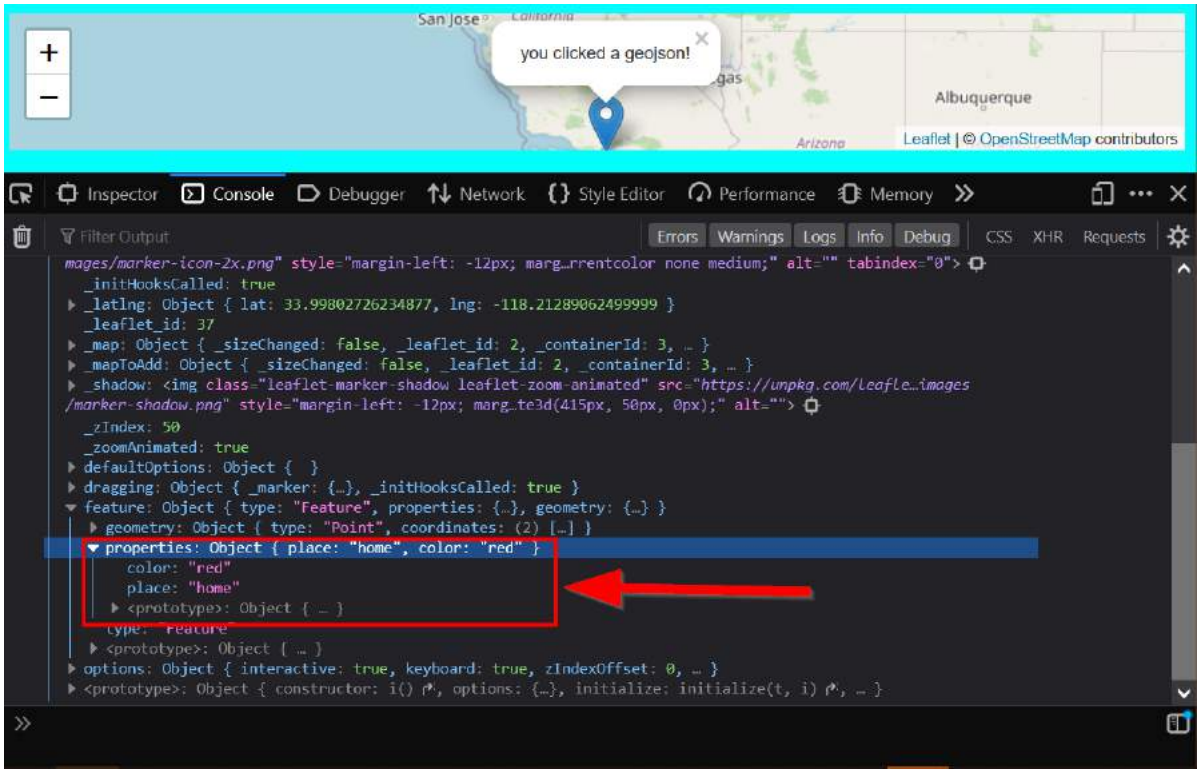
We can drill down into our GeoJSON by clicking on the arrows:



Find the `feature` property and click the arrow to expand it:



Look at the `properties` and notice what is in there!



Right! Those are the columns and values we created from the first part of the lab 3b!

This is called `traversing` the `object` path, and it works the same way when we linked our `photos` or `.css`. The key difference is that it is within one file!

Recapping how we got here, we: 1. Went into the object (`layer`) 2. Clicked on `feature` 3. Clicked on `properties`.

To access the place name, we will need to specify that with `place`.

As a result, our path should look like this:

```
layer.feature.properties.place;
```

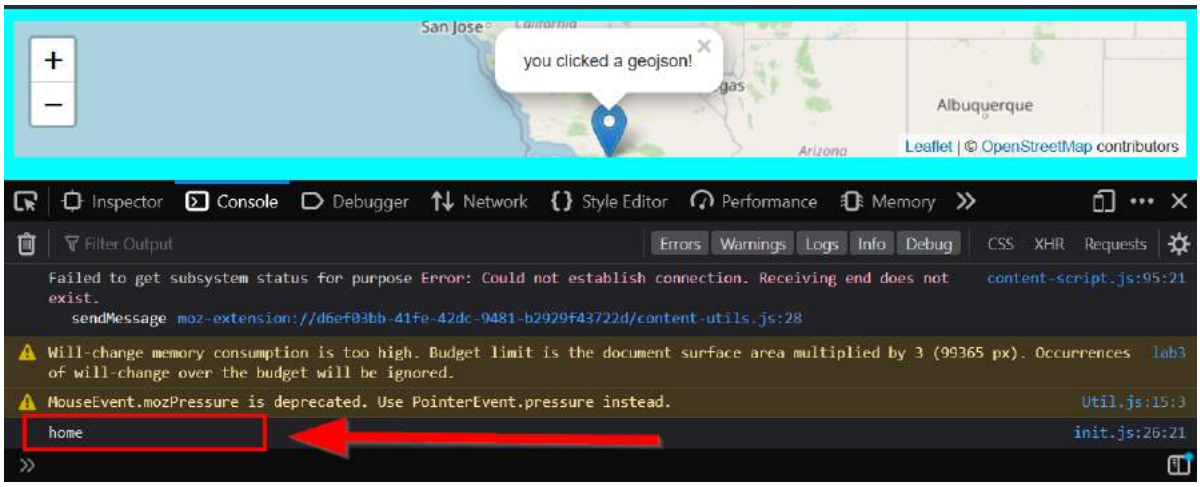
OMG!! The `.` returns?!

Aha, very observant! Similar to `chaining` methods, we use the `.` to chain going down an `object` path. Why is that?! Well.. It has something to do with `classes`, but thats out of the `scope` of this class. (multiple coding puns intended.) If you really want to learn more, click here to read about Object-Oriented Programming and JavaScript: click if you dare!.

Let's `console.log()` the result to make sure we have the right path down:

```
L.geoJSON(data
    ).bindPopup(layer => {
            console.log(layer.feature.properties.place)
            return "you clicked a geojson!"
    }).addTo(map);
```

When you click a point, the correct `value` should show up:



Woo!! Now let's `return` this value instead of the generic message:

```
L.geoJSON(data
    ).bindPopup(layer => {
            console.log(layer.feature.properties.place)
            return layer.feature.properties.place
    }).addTo(map);
```

Now when you click on the map, the `place` values shows up!

Utilizing our GeoJSON's `color` property

Before we finish this module, let's take what we learned one step further and use our `color` property too.

While `bindPopUp()` was nice and an outside method, changing the color needs to be inside of the `L.geoJSON()` call. So we have to attach it to an object inside as follows:

```
L.geoJSON(data, { 1
    style: layer => { 2
        return {color: layer.feature.properties.color}; 3
    }
```

```
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
```

1. Here we add a `,` to add a new value, and then a `{` to start our new object
2. `style` is what Leaflet's `L.GeoJSON()` needs, so we have to use that
3. We are assigning our `layer.feature.properties.color` here!

> **YOU LIE!!! THIS DOES NOT WORK**
>
> Correct! This code will not work because… A GeoJSON's color property can only be set for `L.CircleMarkers`, `lines`, or `polygons` but **not** regular `L.markers`.

Converting our GeoJSON to `CircleMarkers`

Since `{style: "red"}` or any color won't work for our marker, we need to convert it into a circle marker using the `pointToLayer()` method! Again, this has to be inside the `L.geoJSON()` because that is where **Leaflet** must know what color to make the features.

We will use the `arrow-function` so we can type fewer characters:

```
        L.geoJSON(data, {
            pointToLayer: (feature, latlng) => { 1
                return L.circleMarker(latlng, {color: feature.properties.color});
2
            }
            // ... other code here
```

1. Here we pass in our `feature` and `latlng` into the simplified `=> function`
2. Now we convert it to a `L.circleMarker()`, with `latlng` being the first parameter, then setting `color` to the `feature.properties.color`.

The `fetch`'s final `.then` should now look like the following:

```
fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
```

```
                    return L.circleMarker(latlng, {color:
feature.properties.color})
                }
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
    })
```

Checkpoint

Our current `init.js` JavaScript file should look like this:

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
                    return L.circleMarker(latlng, {color:
feature.properties.color})
                }
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
    })
```

Your final `init.js` should look like this:

js/init.js

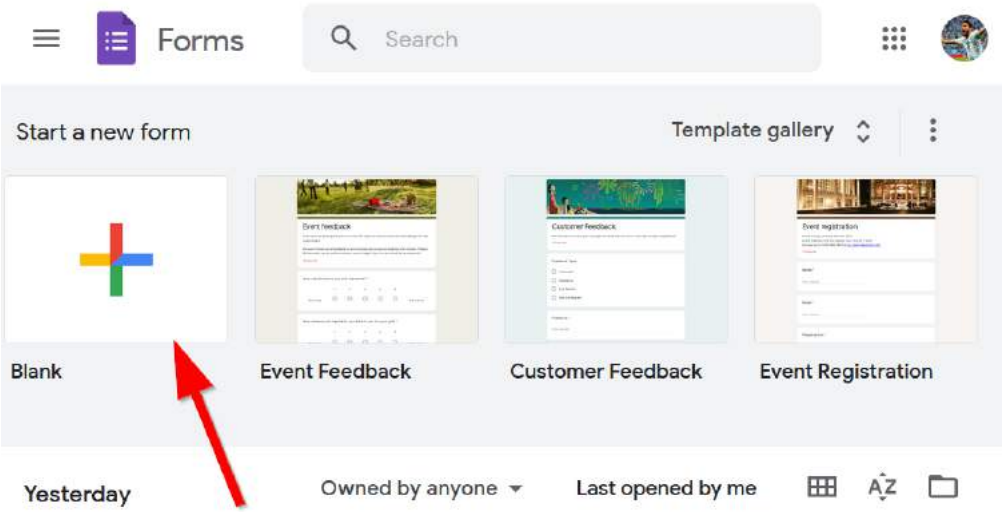
```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
                    return L.circleMarker(latlng, {color:
feature.properties.color})
                }
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
    })
```

Last update: 2023-04-27

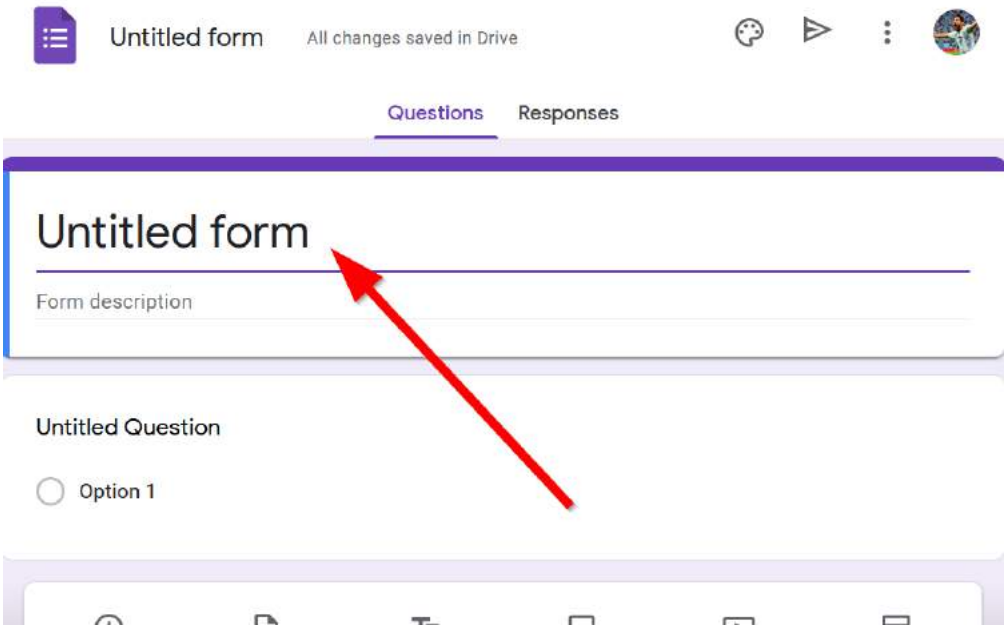
Our community-based survey

Creating a new Google Form

Navigate to Google Forms and click on “Blank”:



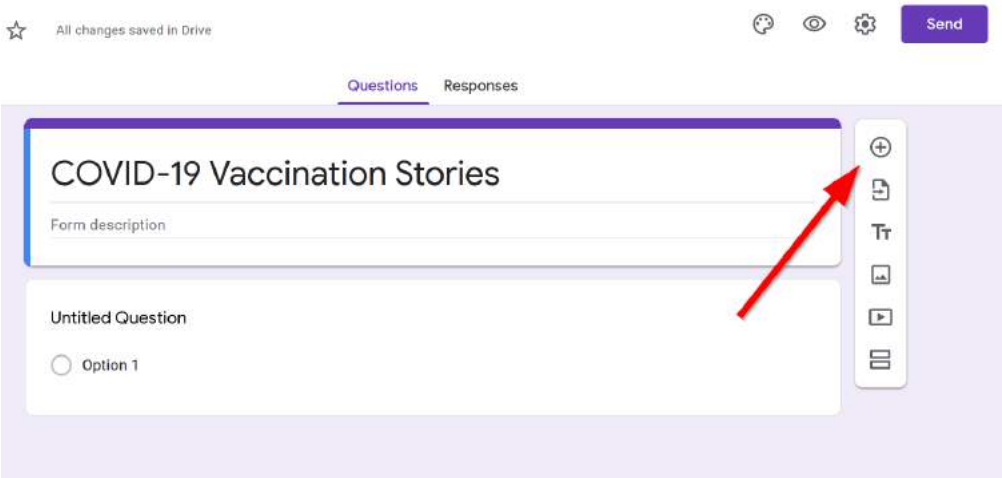
Give a title to your Google Form by clicking on “Untitled Form”:



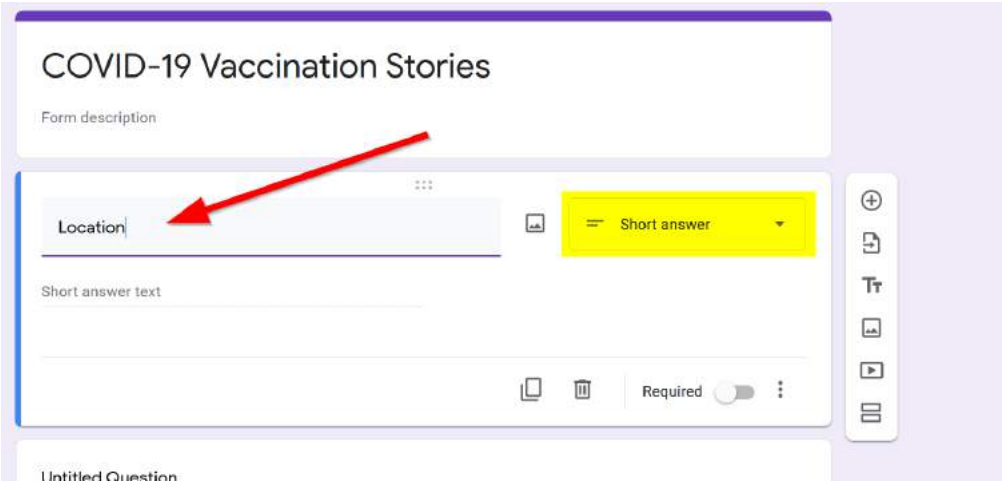
Add a little description about the survey form here. For our example we will be collecting stories about vaccinations during COVID-19.

Adding a Google Form Question

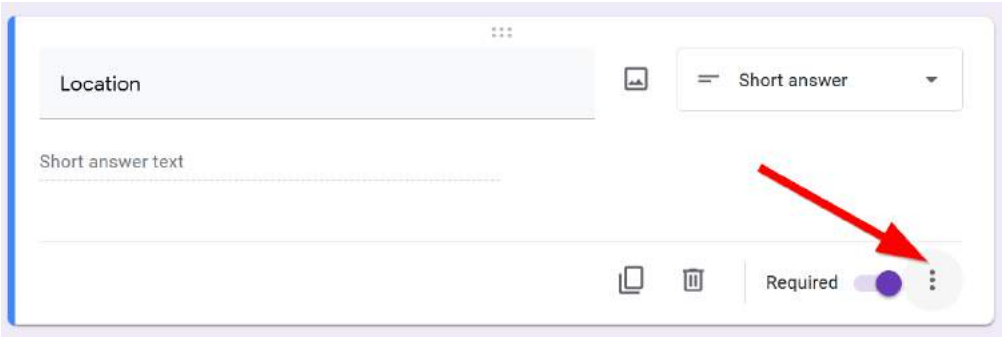
Click on the “Add Question” button to add a question:



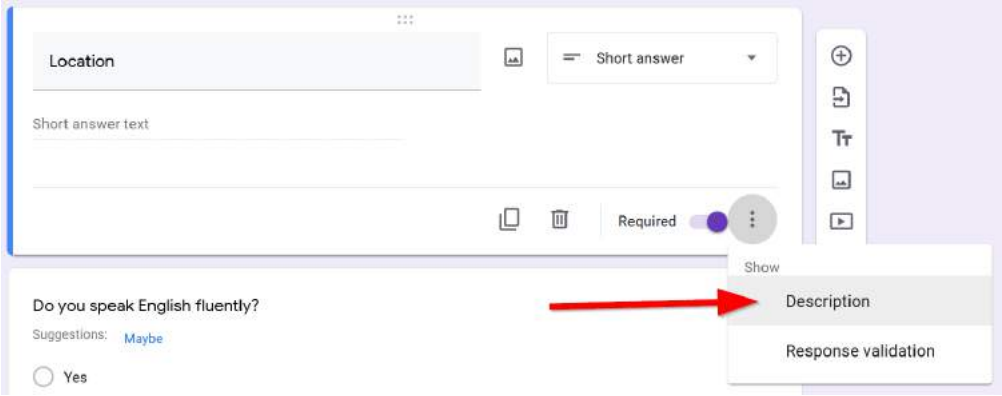
Start typing “Location” and Google will automatically guess the question type for you.



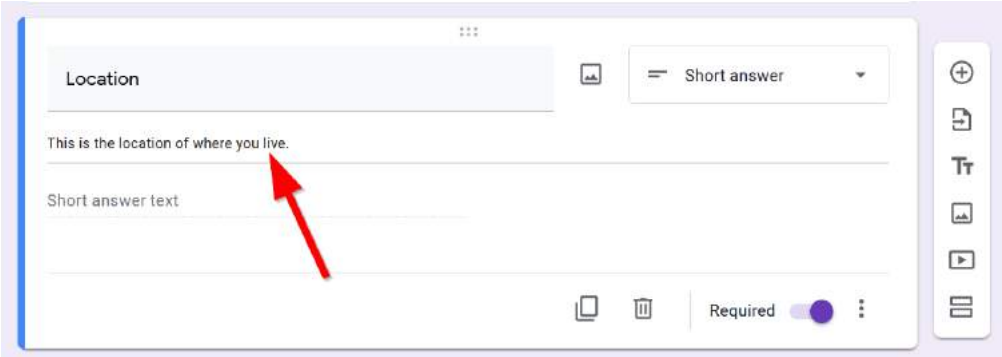
You can fill in descriptions below the question to help with answering by clicking on the “more” triple dots:



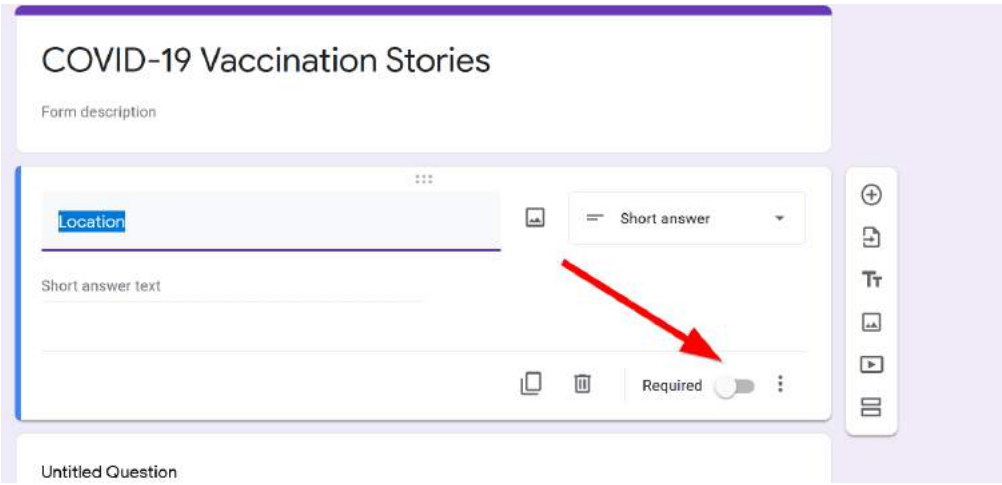
Then going to description:



And typing out a help description:

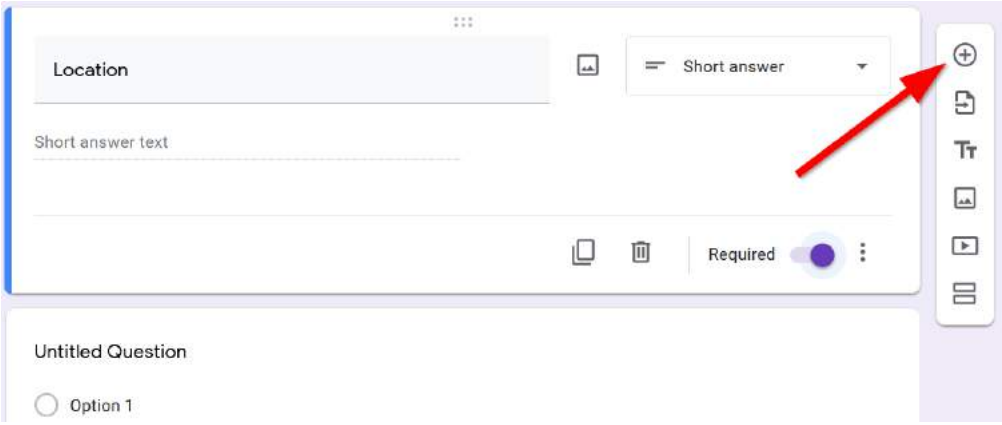


You can check the “required” mark to make this question necessary to move on.

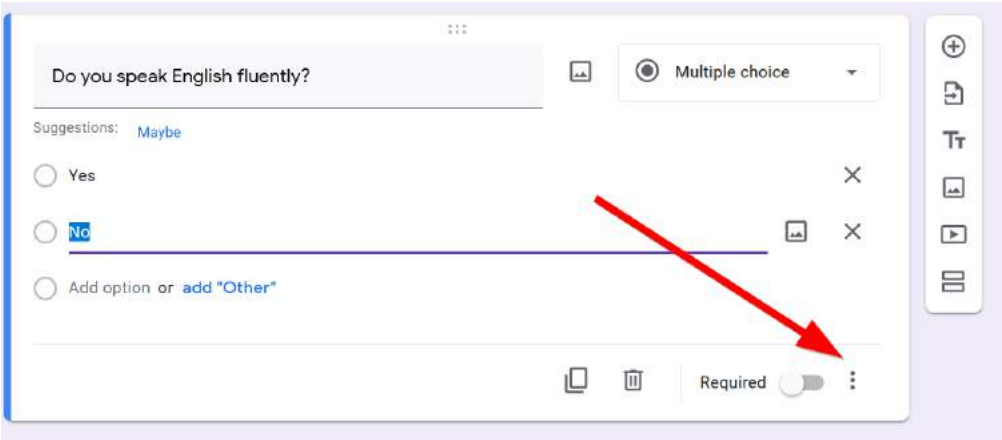


Conditional Questions

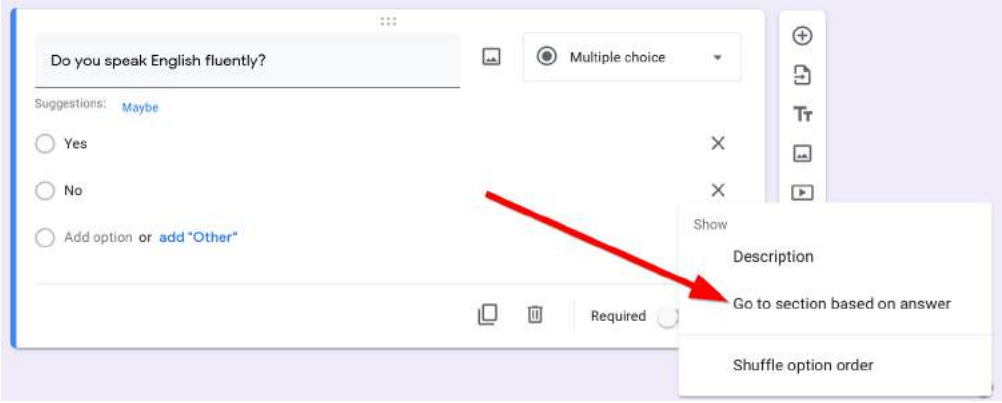
Add a relevant question to help guide the user about the survey, `Do you speak English fluently`?



Click on the `triple dots ...`:

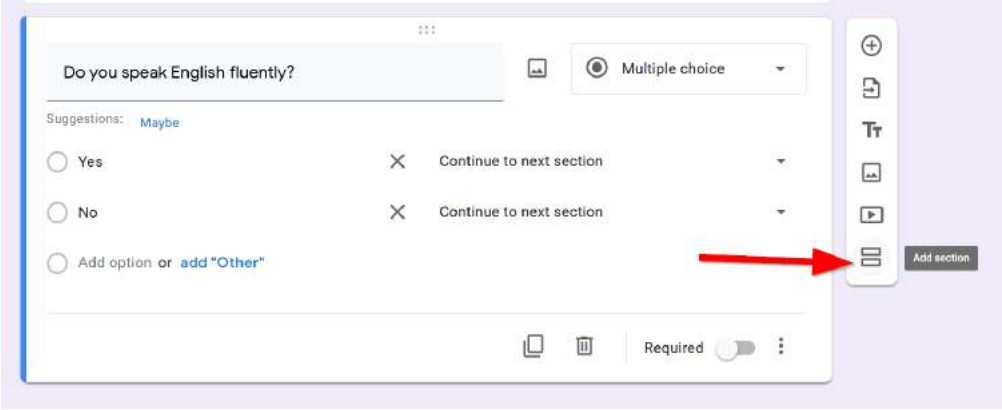


Select `Go to section based on answer`:



Adding new sections

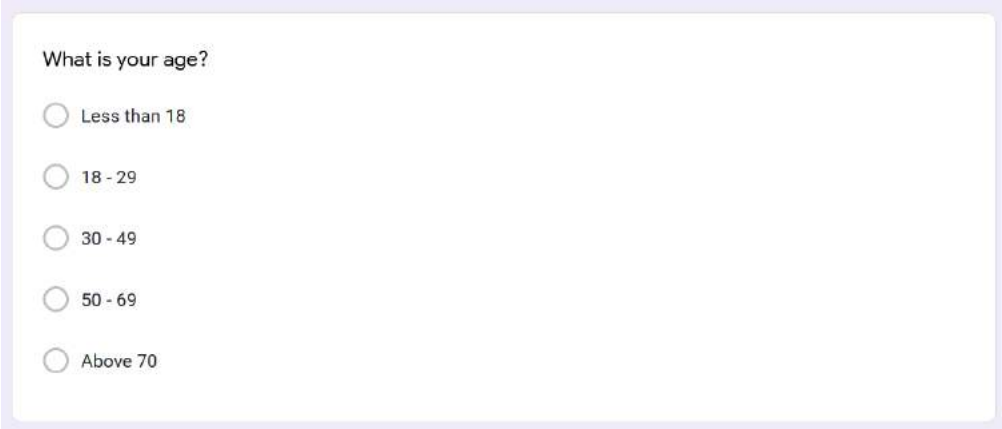
Add a new section:



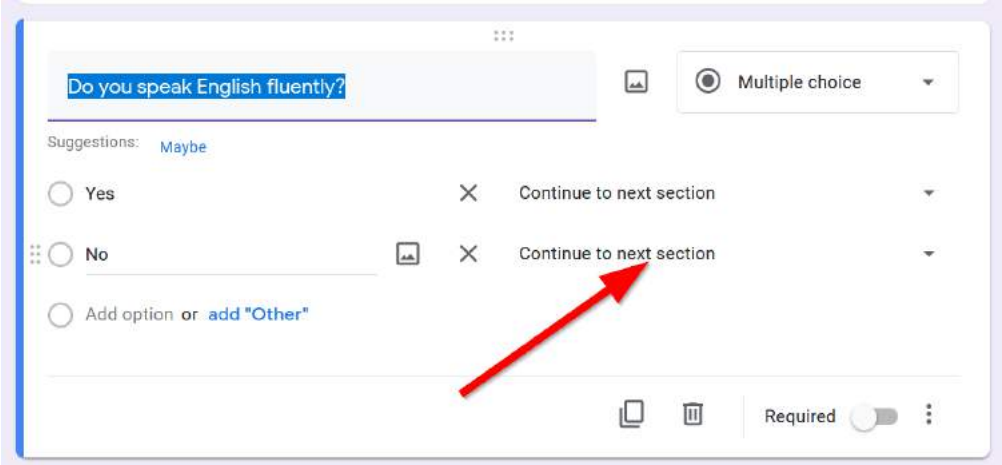
Title it **Language Details**

Add one question on **What language do you primary speak at home?**

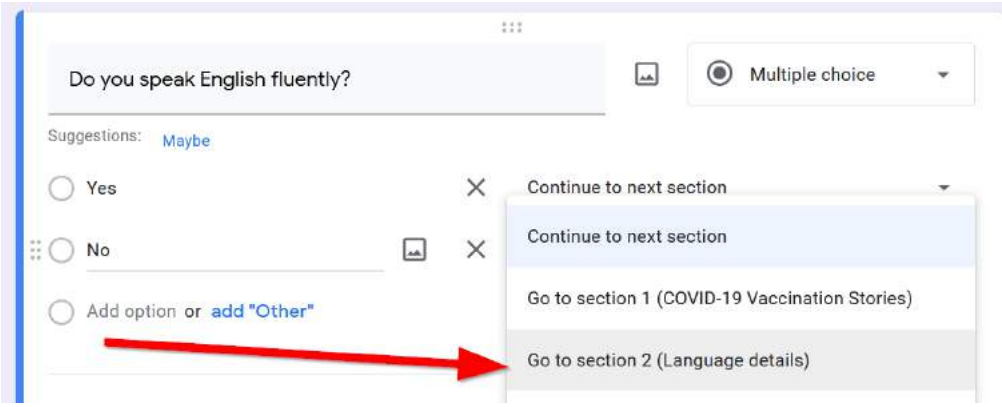
And another on **What is your age?**



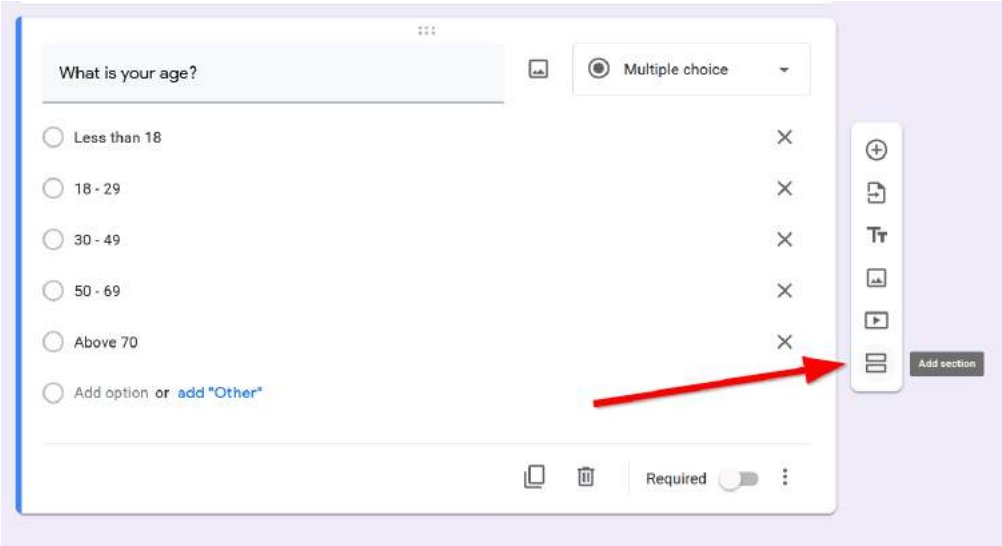
Go back to the question **Do you speak English fluently?** and for No:



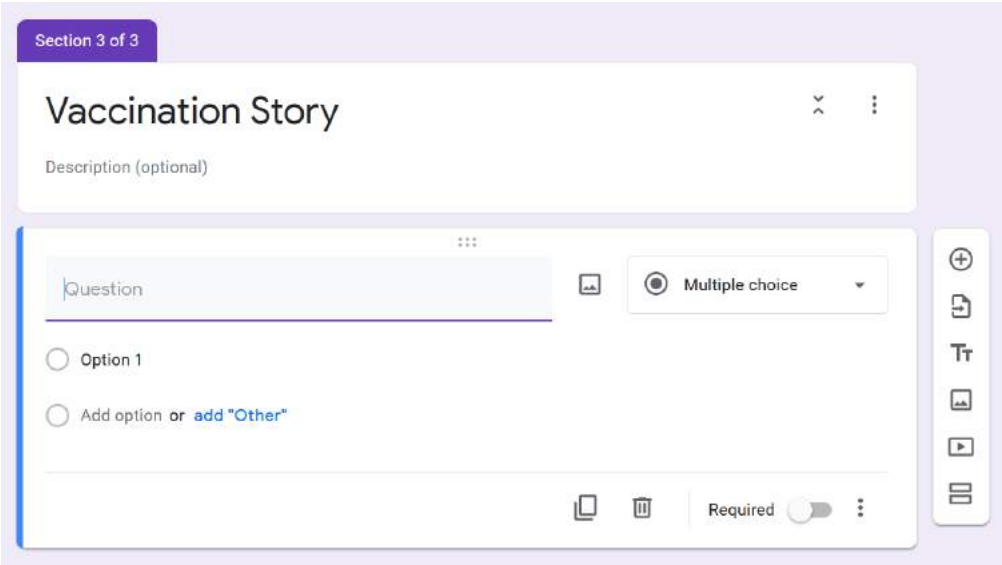
Choose **Go to Section 2**.



Scroll to the bottom and add a new section:



Call it **Vaccination Story** and add a new question:



Add the question:

> **Would you be comfortable with sharing your story?**
>
> - Yes
> - No

Make it a required question.

⚽ In-class Exercise #1 - Open Ended Questions

Task

1. Add two open-ended questions to your form

Bonus

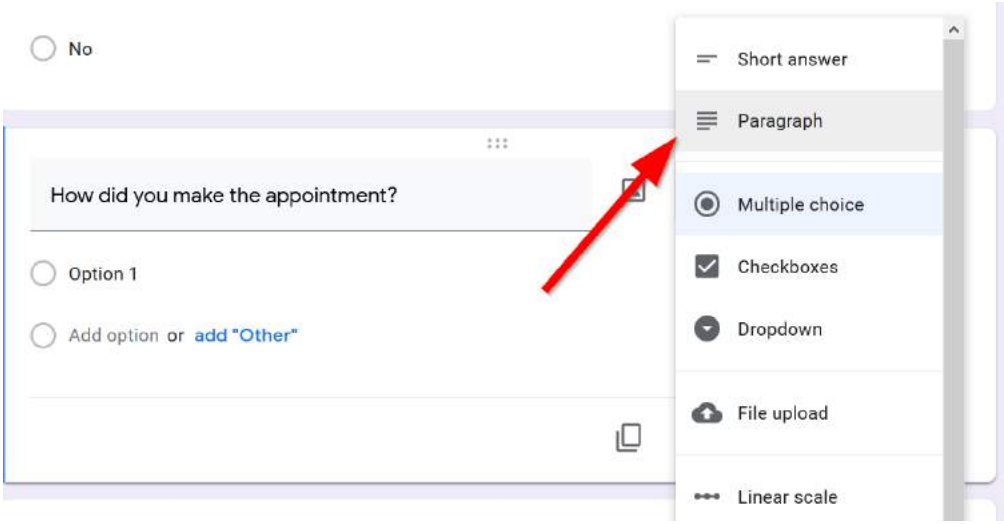
Feel free to make a branching question to the `open-ended questions`, so if some one chooses “No” they submit the form and finish.

Answer

Some open-ended questions can be the following:

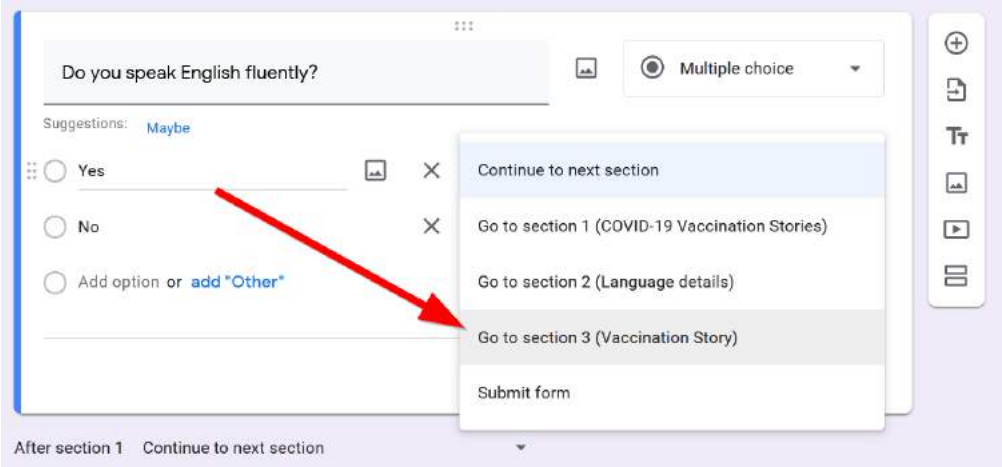
> How did you make the appointment?
>
> How did you get to the location?

Make sure they are both `Paragraph` answer types:



Wrapping the form up

Go back to `Do you speak English fluently` and make **Yes** go to `Section 3`:

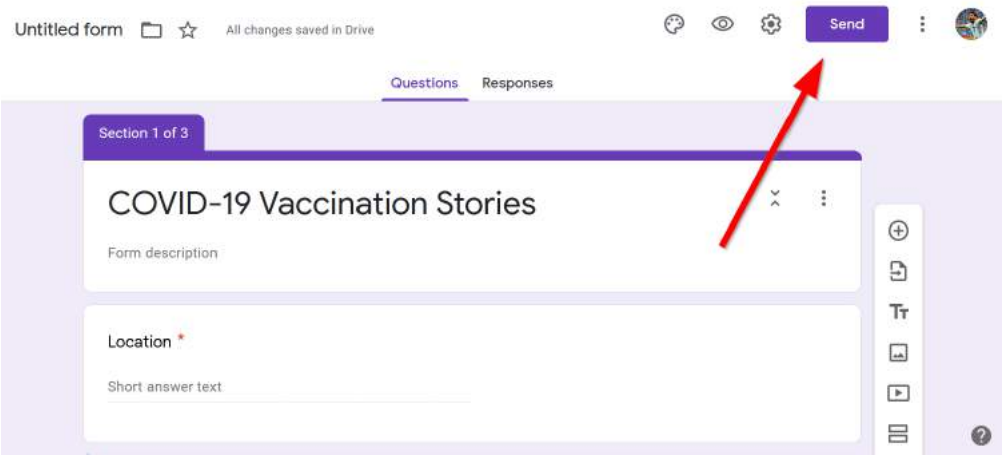


Your final form should look something like this:

https://forms.gle/E8xBqKLJNJyvxGcK8

Sharing your form

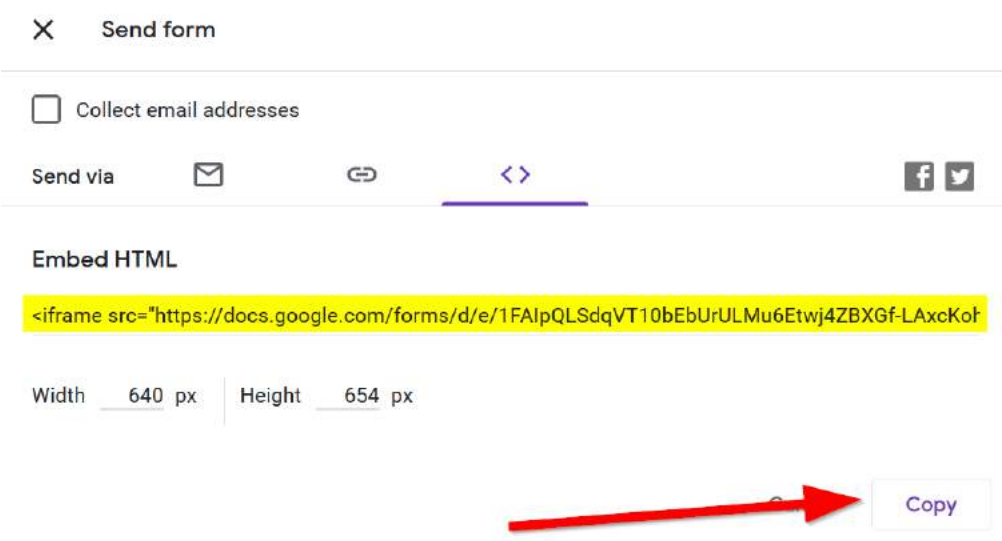
Click the **Send** button to share your form



Embeding your form

Under the `Send` button menu, click on the Embed tab

Copy the Embeded HTML by clicking the **Copy** button:



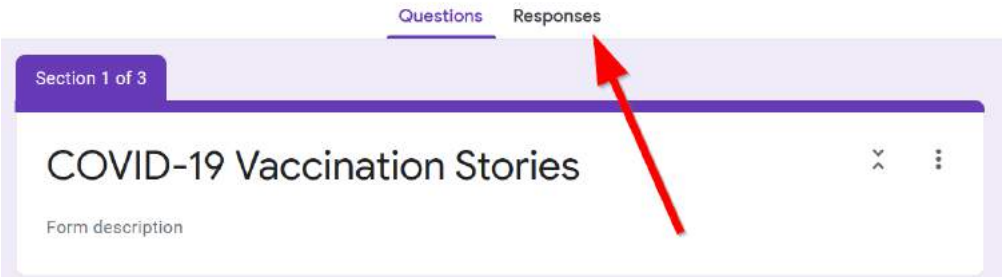
Go back to `index.html` and paste the embedded HTML into the `contents` div:

index.html

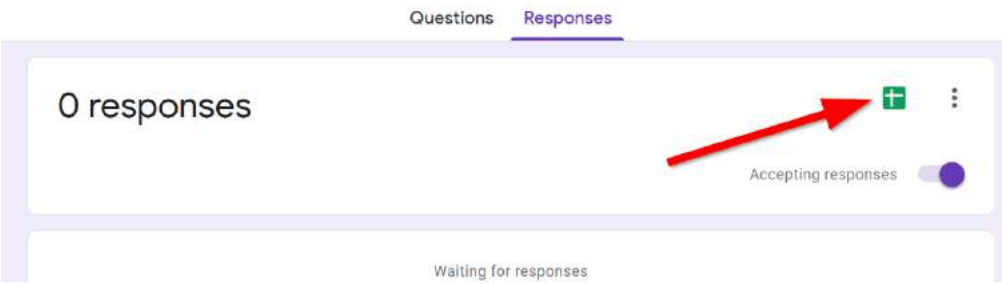
```
<div id="contents">
    <!-- page contents can go here -->
    <iframe
src="https://docs.google.com/forms/d/e/1FAIpQLSdqVT10bEbUrULMu6Etwj4ZBXGf-
LAxcKohAINFbIdZmHS6OA/viewform?embedded=true" width="640" height="654"
frameborder="0" marginheight="0" marginwidth="0">Loading…</iframe>
</div>
```

Accessing the Responses

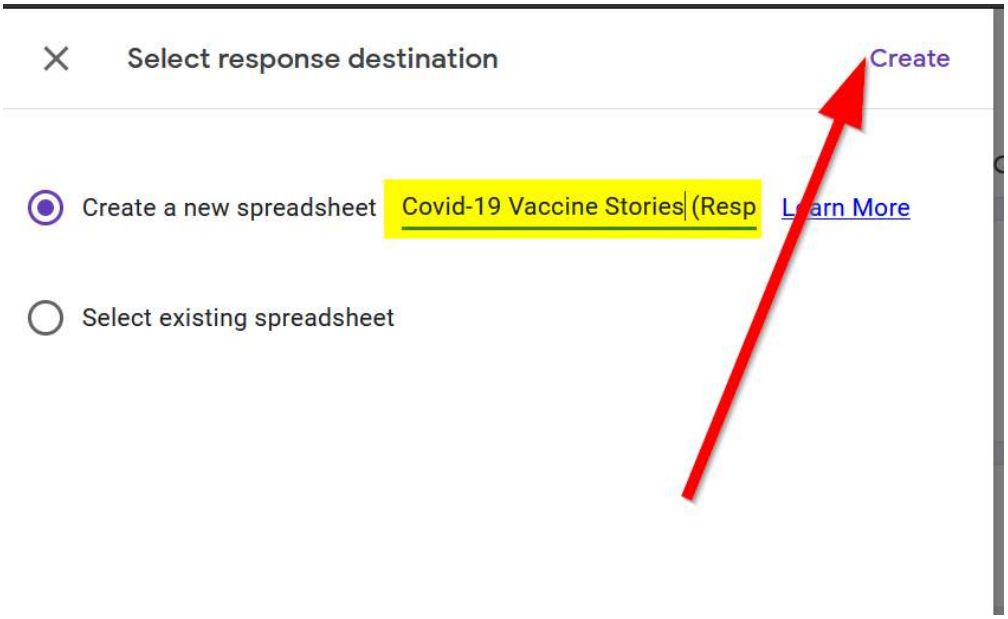
Click on `Responses`:



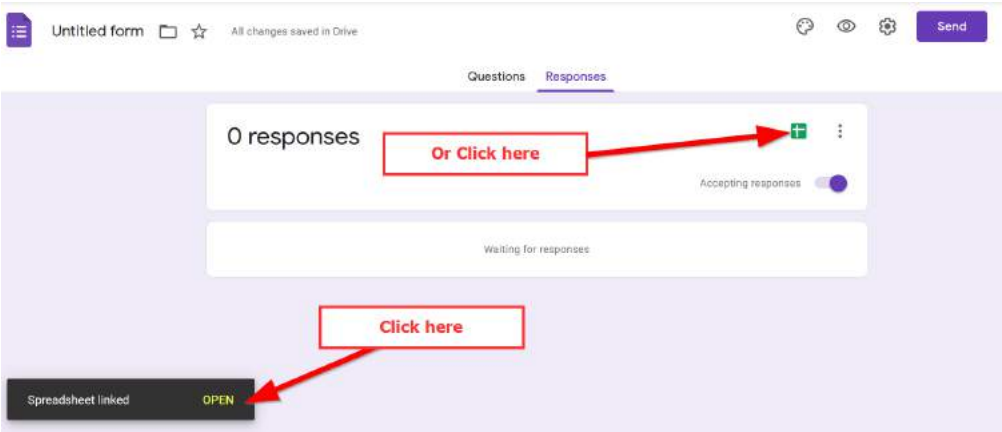
Click on the Google Spreadsheet button:



Change the title and click on `Create` button:



Open the response Google Sheet by clicking open or the Google Sheets icon:



Last update: 2023-04-27

Code Refactoring before geoCODING!

`Refactoring code` means re-writing code **without** changing its function to be more understandable and/or reusable. We are going to refactor so that when we get the Google form data it is simpler to know where to change the code!

Putting the `Fetch` in a function

Our `fetch` call sits out in the middle of nowhere, which is the `Global` space! That is not good because if the `fetch` doesn't work then our page won't load!

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch(`map.geojson`)
    .then(response => {
        console.log(response)
        return response
    })
    .then(data =>{
        // do something with the data
    })
```

Leaving it there will break our code if we leave it there without a file to get, so let's move that `fetch` into a function we will call later:

`fetch` in function

```
function loadData(){
    fetch(`map.geojson`)
        .then(response => {
            console.log(response)
            return response
        })
        .then(data =>{
            // do something with the data
        })
}
// we will put this comment to remember to call our function later!
// loadData()
```

`fetch` out of function

```
fetch(`map.geojson`)
    .then(response => {
        console.log(response)
        return response
    })
    .then(data =>{
        // do something with the data
    })
```

Changing the `map.geojson` into a variable

Let's also change the `map.geojson` to a parameter called `url` that way we can use this function to get different urls!

To make our lives easier, we'll define a new variable called `dataUrl` and use that as a placeholder for our function too.

js/init.js

```
    const dataUrl = "https://some.data.com/"  1
    function loadData(url){  2
        fetch(url)
            .then(response => {
                return response
            })
            .then(data =>{
                // do something with the data
```

```
            })
    }
    // we will put this comment to remember to call our function later!
    // loadData(dataUrl)
    ❸
```

1. The URL variable we can change later when we get the URL we need.
2. The new URL parameter to get data from!
3. Function call that uses the `loadData()` function and `dataUrl` parameter!

Some CSS Touch-up!

Lastly, let's make our survey look a little nicer, by adding two columns to our secondary grid in the `.main` CSS selector.

Change #1 Adding columns lengths

Because we are using `css-grid` the way to add columns is by using the `grid-template-columns` class. We can assign a fixed value, like `100px` for 100 pixels, but let's make our site scalable to any screen size by using a `fr` value for each of `1fr 1fr`. So our current change should look like:

/styles/style.css

```
.main{
    grid-area: main_content;
    grid-template-columns: 1fr 1fr;
    grid-template-areas: "main_map" "content";
    display: grid;
}
```

`fr`eal, an side about CSS unit lengths

CSS has many units for length, such as `pixels` or `%` percentage that can account for how much of a page to cover. However, is a new unit `fr` stands for **fraction** and it represents a fraction of the available space in the grid container. What this means it can automatically account for the `fraction` of a page!!! You can also mix and match units. Learn more here.

CCS Change #2: Putting content on the same row

Now that we created the columns, now we need to assign the columns to the rows! With `css-grid` the `grid-template-areas` property is already how we assign `rows` and `columns`:

```
grid-template-areas: "main_map" "content";
```

Means have one row for `main_map` and one row for `content`.

To put the areas on the same row we modify both of them to be **in** the same **”** **“** pair, and separated by a space ( ), as follows:

```
grid-template-areas: "main_map content";
```

If you change the order, like `"content main_map"` then `content` will show up on the left:

```
grid-template-areas: "content main_map";
```

For now, let’s keep the map on the left.

The resulting CSS should look like the `CSS` after tab:

`CSS` after

```
.main{
    grid-area: main_content;
    grid-template-columns: 1fr 1fr; 1
    grid-template-areas: "main_map content"; 2
    display: grid;
}
```

1. `1fr 1fr` gives us two equal columns, setting it to `2fr 1fr` makes the first column fill up twice the space of the second.
2. `"main_map content"` are in the same quotations `"` now!

`CSS` before

```
.main{
    grid-area: main_content;
    grid-template-areas: "main_map" "content" ;
    display: grid;
}
```

🏁 Checkpoint

Before moving on, make sure your code looks like the following:

index.html

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
                <iframe
src="https://docs.google.com/forms/d/e/1FAIpQLSdqVT10bEbUrULMu6Etwj4ZBXGf-
LAxcKohAINFbIdZmHS6OA/viewform?embedded=true" width="640" height="654"
frameborder="0" marginheight="0" marginwidth="0">Loading…</iframe>
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2023)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

js/init.js

```javascript
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

```

```
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

const dataUrl = "https://some.data.com/"

function loadData(url){
    fetch(url)
        .then(response => {
            console.log(response)
            return response
        })
        .then(data =>{
            // do something with the data
        })
}
// we will put this comment to remember to call our function later!
// loadData(dataUrl)
```

✅ Final Template Code

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

```

```
    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
                <iframe
src="https://docs.google.com/forms/d/e/1FAIpQLSdqVT10bEbUrULMu6Etwj4ZBXGf-
LAxcKohAINFbIdZmHS6OA/viewform?embedded=true" width="640" height="654"
frameborder="0" marginheight="0" marginwidth="0">Loading…</iframe>
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2023)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

const dataUrl = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSpOaH94y-
oguAqbtcvZRyKdrEYiT1JOzW0jmmreznYS8THdQTYQ6cUB7J_68SZLgjpXbB_FY_nDf2A/pub?
output=csv"

function loadData(url){
    fetch(url)
        .then(response => {
            console.log(response)
```

```
            return response
        })
        .then(data =>{
            // do something with the data
        })
}
// we will put this comment to remember to call our function later!
loadData(dataUrl)
```

styles/style.css

```
body{
    display: grid;
    grid-auto-rows: auto 1fr;
    grid-template-areas: "header" "main_content" "footer";
    background-color: aqua;
}

header{
    grid-area: header;
}

#footer{
    grid-area: footer;
}

.main{
    grid-area: main_content;
    grid-template-columns: 1fr 1fr;
    grid-template-areas: "main_map content";
    display: grid;
}

#contents{
    grid-area: content;
}

#the_map{
    height:80vh;
    grid-area: main_map;
}
```

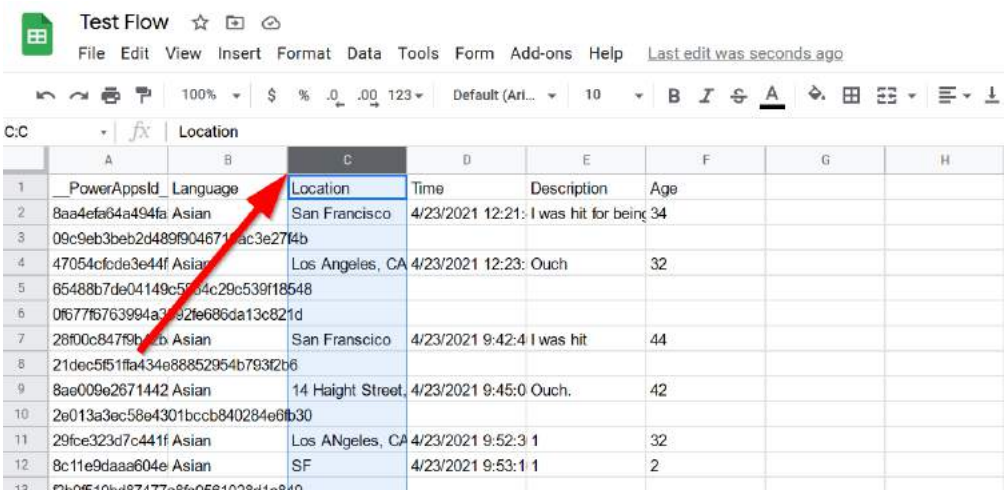
Last update: 2023-04-27

Geocoding with Google

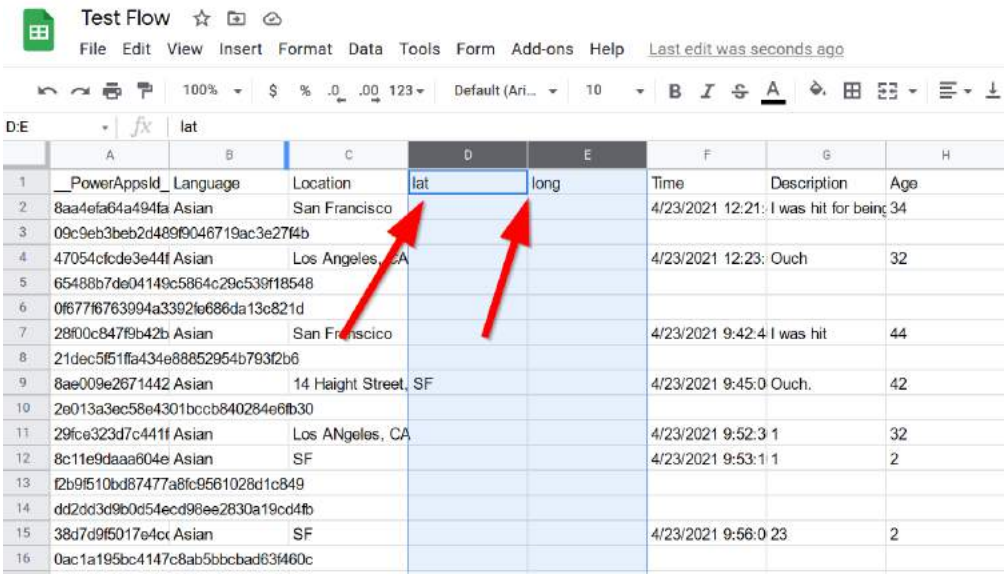
Revisiting Functions

Open up your Google Sheet from part 1.

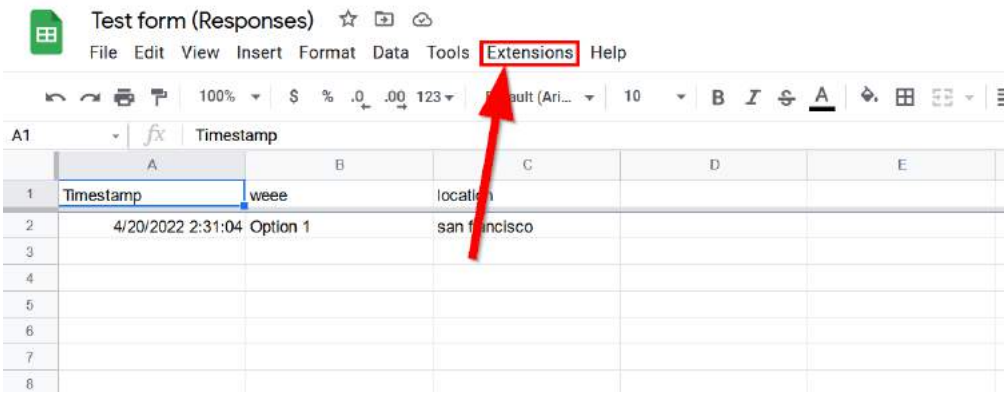
Go to column for `Location` and remember what column it is, for me it is `C`:



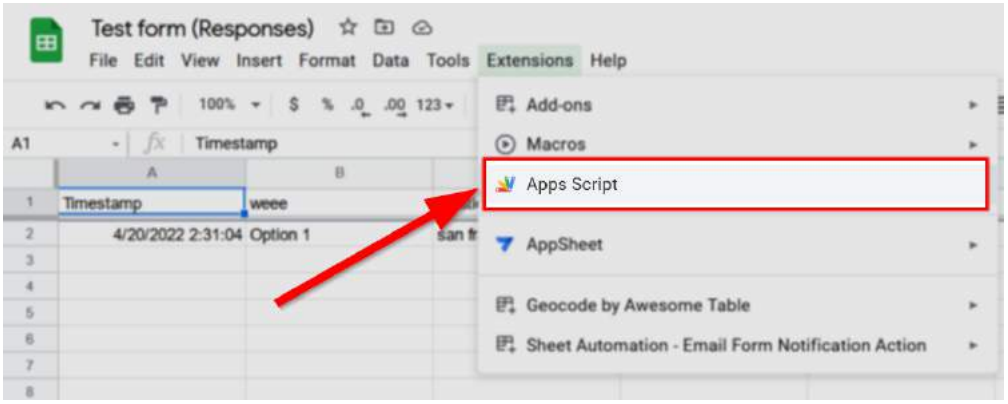
Next, add two columns, one for `lat` and another for `long`:



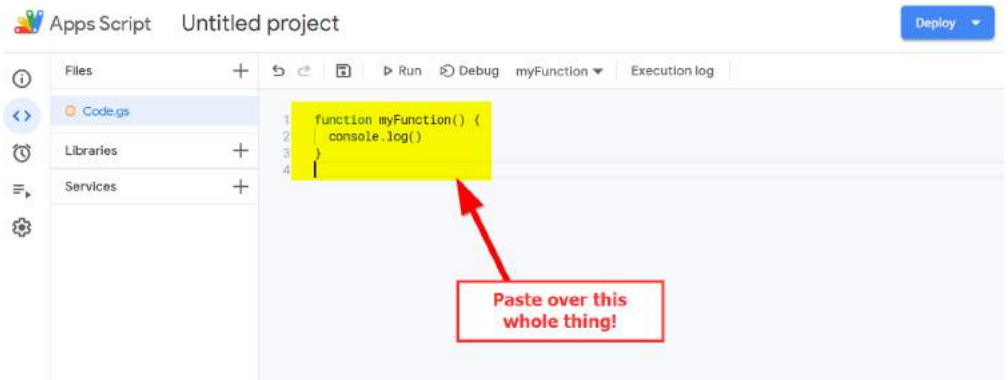
Now click on **Extensions**



Now click on **Apps Script**



When you first launch, you will see a blank myFunction() select it and get ready to paste over it:



Copy and paste the following code into the entire script:

Google's Script Editor

```
function myFunction() {
  let sheet = SpreadsheetApp.getActiveSheet();

  let range = sheet.getDataRange();
```

```
  let cells = range.getValues();

  let latitudes = [['lat']]; 1
  let longitudes = [['lng']]; 2

  for (let i = 0; i < cells.length; i++) {
    // change cells[i][2] if your address is not in column 'C', for example
cells[i][1] for column 'B' or cells[i][3] for column D
     addressColumn = cells[i][2] 3
     let lat = lng = 0;
    if (i > 0) {
      if (addressColumn){
    let address = addressColumn;
    console.log(address)

    if(address){
      let geocoder = Maps.newGeocoder().geocode(address);
      let res = geocoder.results[0];
        if (res) {
          lat = res.geometry.location.lat;
          lng = res.geometry.location.lng;
        }
      }
    }
    latitudes.push([lat]);
    longitudes.push([lng]);
    }
  }
  sheet.getRange('D1') 4
  .offset(0, 0, latitudes.length)
  .setValues(latitudes);
  sheet.getRange('E1') 5
  .offset(0, 0, longitudes.length)
  .setValues(longitudes);
  Utilities.sleep(5000)
}
```

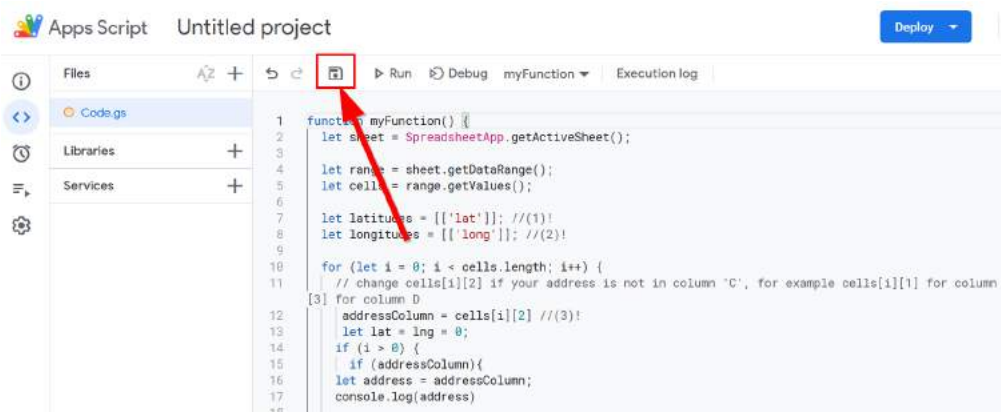
1. This defines the column as latitude to be `lat`
2. This defines the column as longitude to be `lng`
3. `cells[i][2]` the `2` is `number` that your **address** column is in minus 1!! You have to subtract 1 because JavaScript starts counting at `0`!!! For example, column A is `0`!
4. Make sure this column letter matches your `**latitude**` column!!
5. Make sure this column letter matches your `**longitude**` column!!

JavaScript arrays start at 0

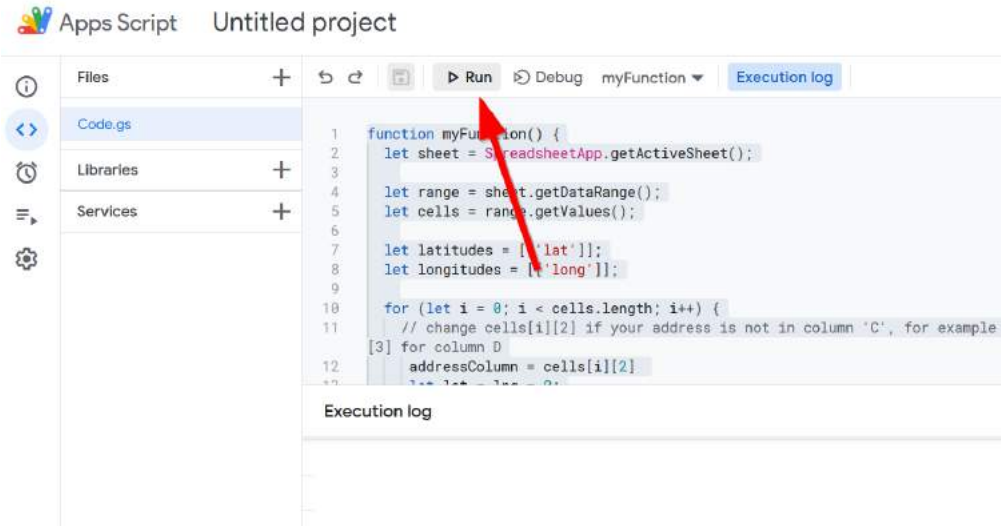
Most programming have two types of indexing, `0` indexed or `1` indexed, which means the number that they start counting lists from. **JavaScript** is `0` indexed, meaning a list with **4** things starts from `0` and ends at `3`, like this: `[0,1,2,3]`.

This is important when we call items from a list and want to get the right item from it!! For example, we have to get the fourth item in the example array like this, `let theFourthItem = myArray[3]`.

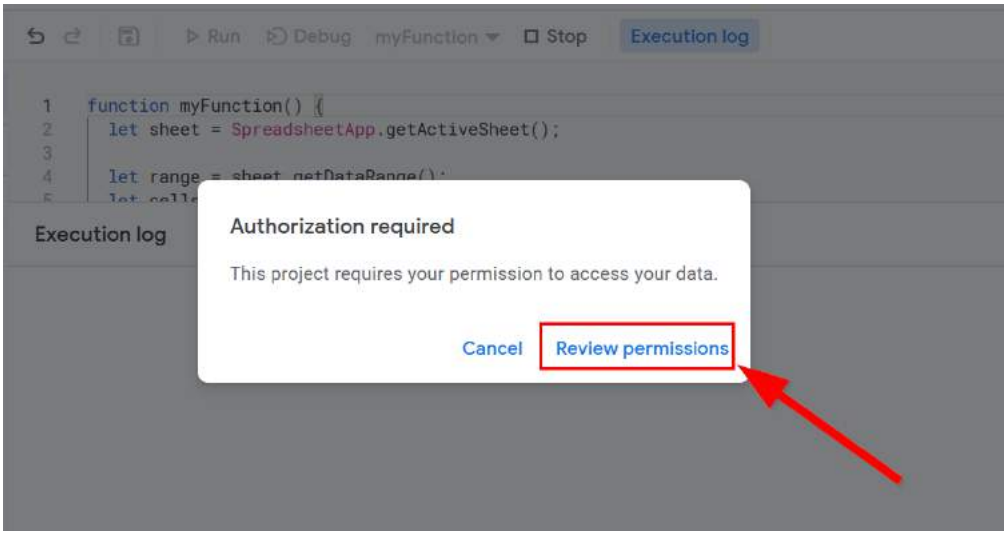
Click on the Save Icon to save your script:



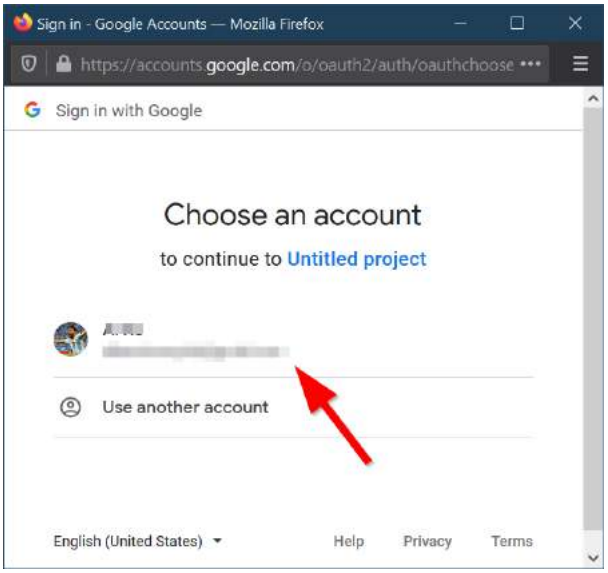
Click on the **Run** button to test the script:



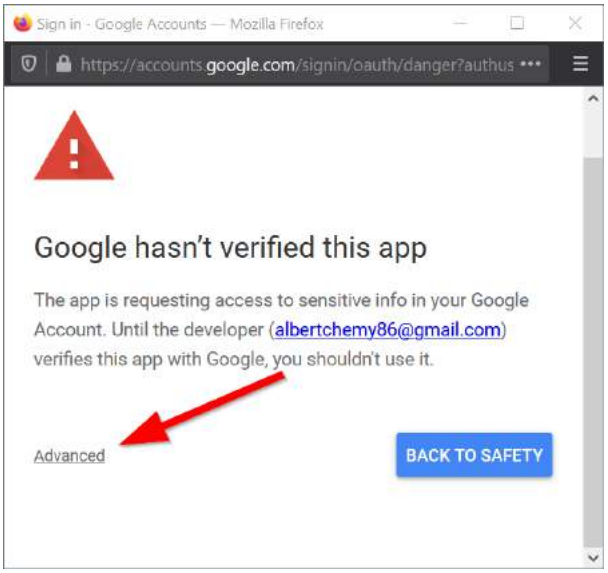
You should get a prompt asking for `Authorization`, click **Review permissions**:



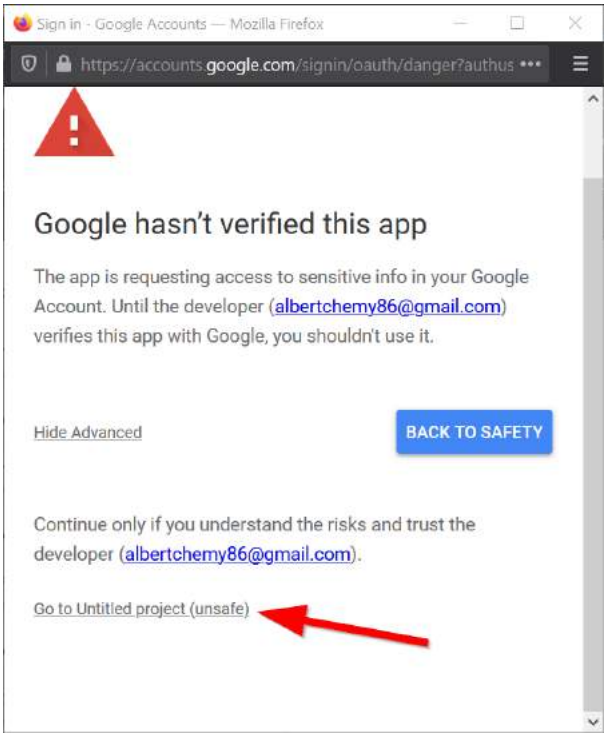
Select your **Google Account** to continue:



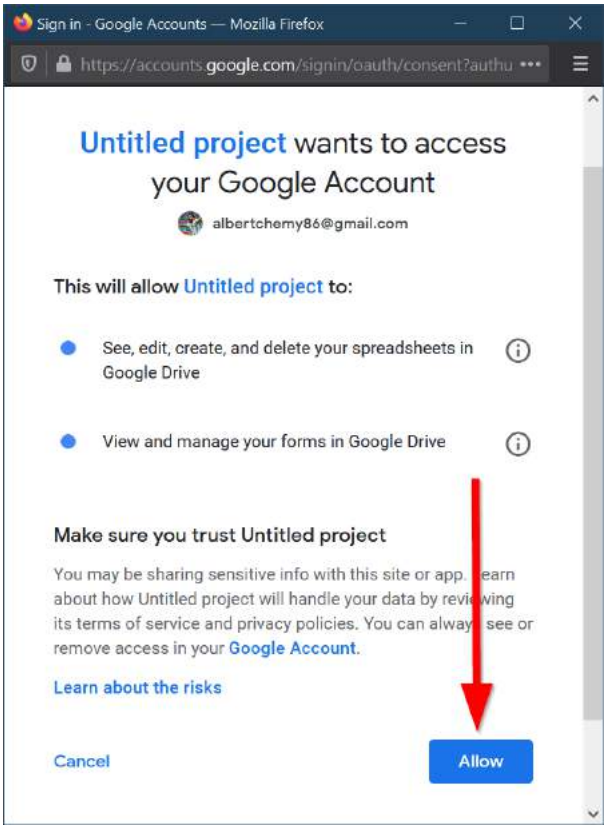
Click on Advanced:



Click on **Go to Untitled Project (unsafe)**



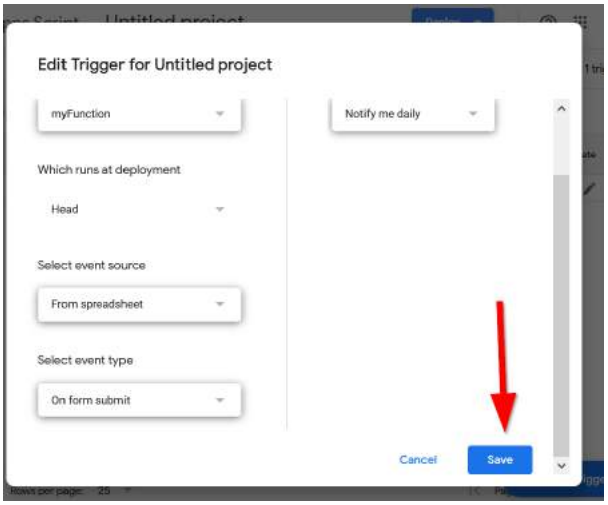
Click on **Allow**



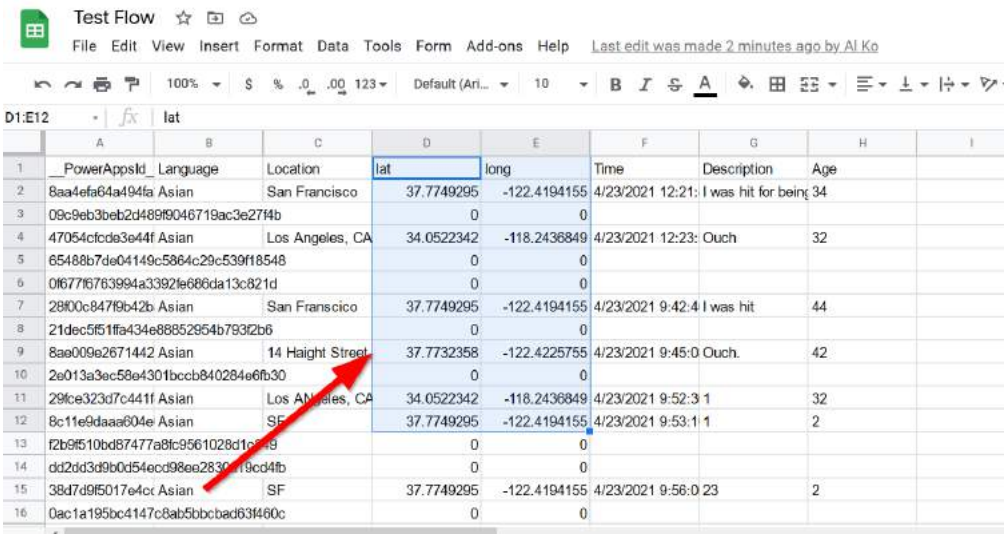
Do you trust yourself? 🤔

Essentially, anyone can write Google Scripts, so Google is making sure that the script is associated with your account before giving it access to this particular spreadsheet.

Click on **Save**:



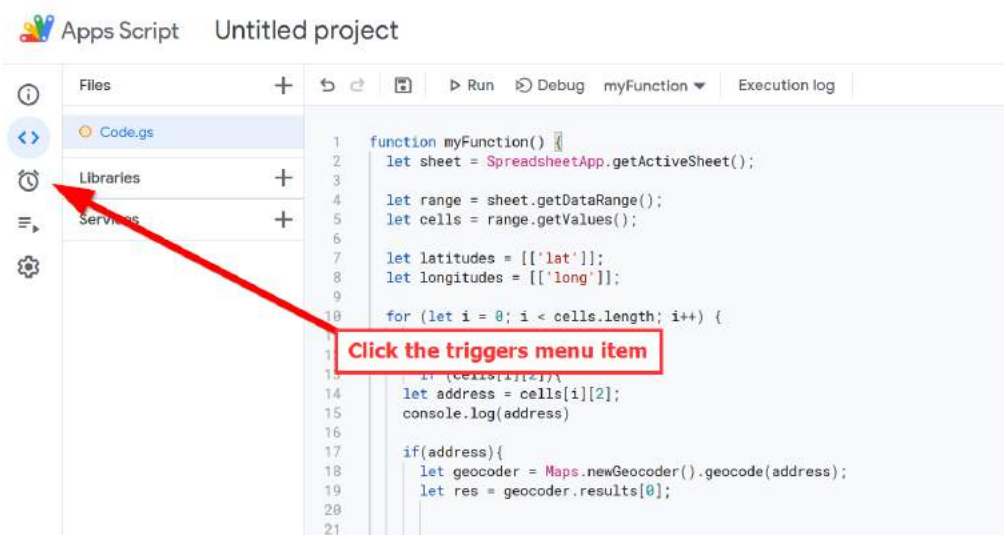
If it ran successfully then you should now see `latitude` and `longitude` filled in the Google Sheet!



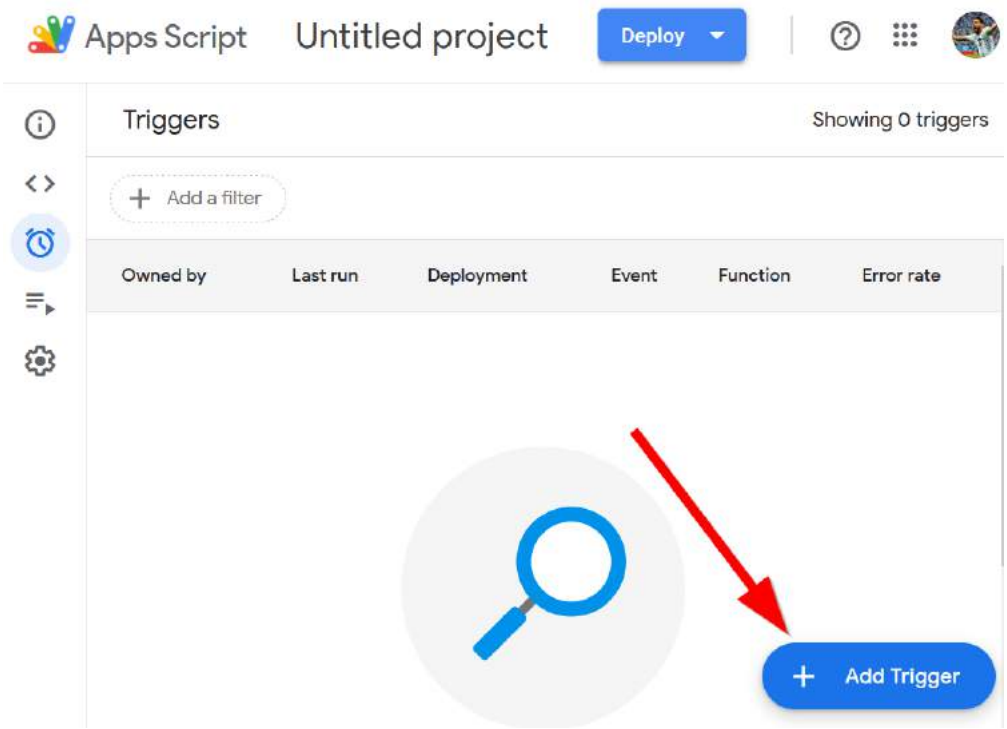
	A	B	C	D	E	F	G	H
1	__PowerAppsId_	Language	Location	lat	long	Time	Description	Age
2	8aa4efa64a494fa	Asian	San Francisco	37.7749295	-122.4194155	4/23/2021 12:21:	I was hit for being	34
3	09c9eb3beb2d489f9046719ac3e27f4b			0	0			
4	47054cfcde3e44f	Asian	Los Angeles, CA	34.0522342	-118.2436849	4/23/2021 12:23:	Ouch	32
5	65488b7de04149c5864c29c539f18548			0	0			
6	0f677f6763994a3392fe686da13c821d			0	0			
7	28f00c847f9b42b	Asian	San Franscico	37.7749295	-122.4194155	4/23/2021 9:42:4	I was hit	44
8	21dec5f51ffa434e88852954b793f2b6			0	0			
9	8aa009a2671442	Asian	14 Haight Street	37.7732358	-122.4225755	4/23/2021 9:45:0	Ouch.	42
10	2a013a3ec58a4301bccb840284e6fb30			0	0			
11	29fce323d7c441f	Asian	Los Angeles, CA	34.0522342	-118.2436849	4/23/2021 9:52:3	1	32
12	8c11e9daaa604e	Asian	SF	37.7749295	-122.4194155	4/23/2021 9:53:1	1	2
13	f2b9f510bd87477a8fc9561028d1c[illegible]9			0	0			
14	dd2dd3d9b0d54ecd98ea2830[illegible]19cd4fb			0	0			
15	38d7d9f5017e4cc	Asian	SF	37.7749295	-122.4194155	4/23/2021 9:56:0	23	2
16	0ac1a195bc4147c8ab5bbcbad63f460c			0	0			

Geocoding on each submission with a trigger

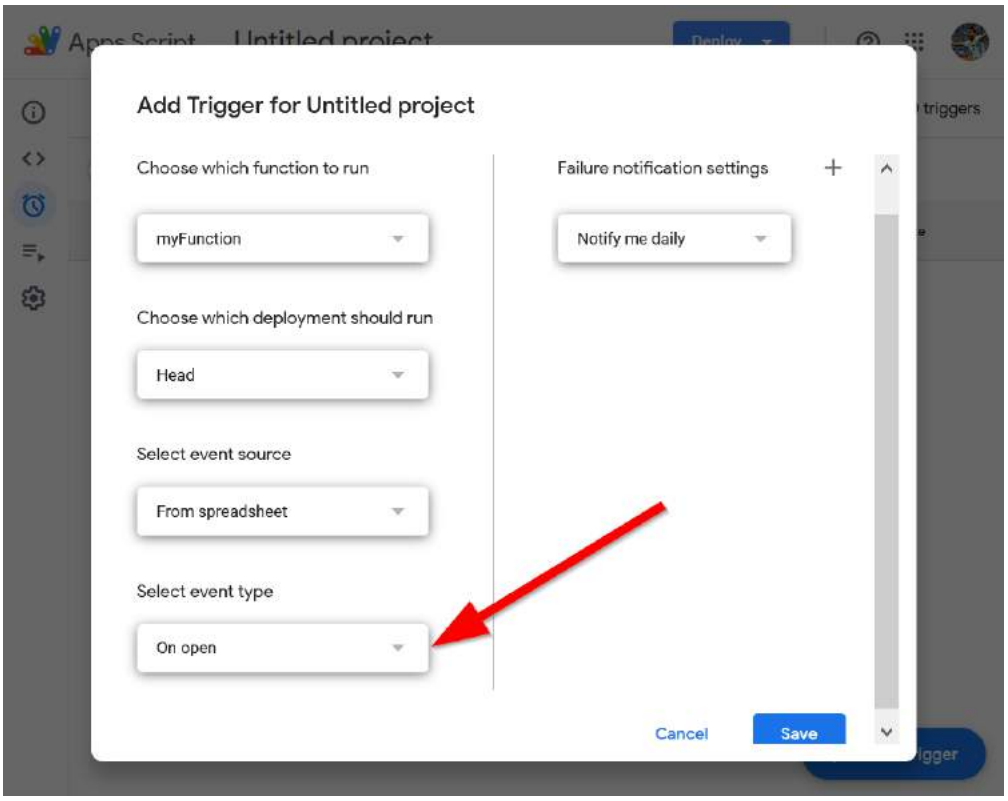
Go back to the Google Scripts and click on the triggers icon ⏱



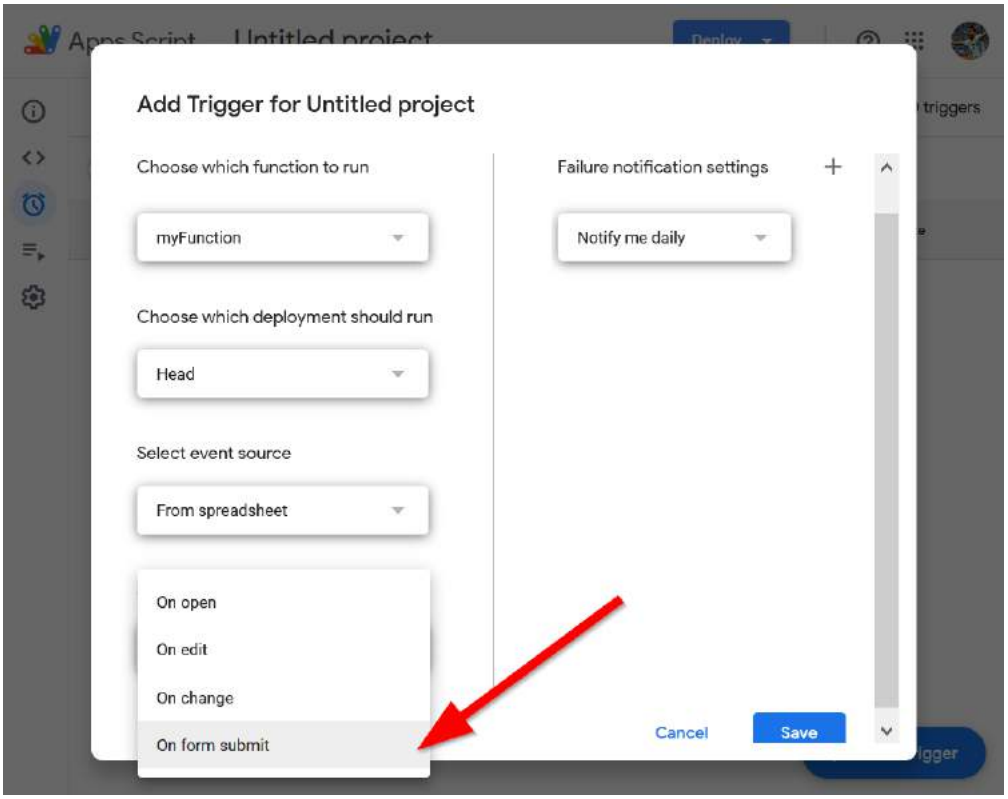
Click on **+ Add Trigger**:



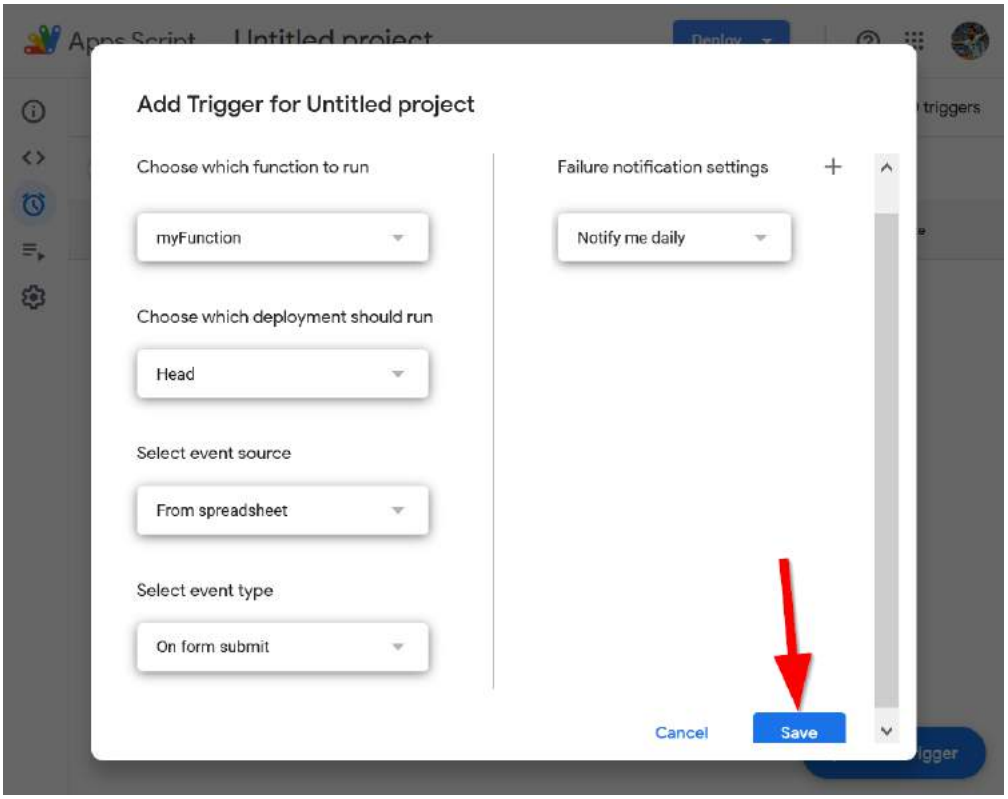
Click on **Select event type**:



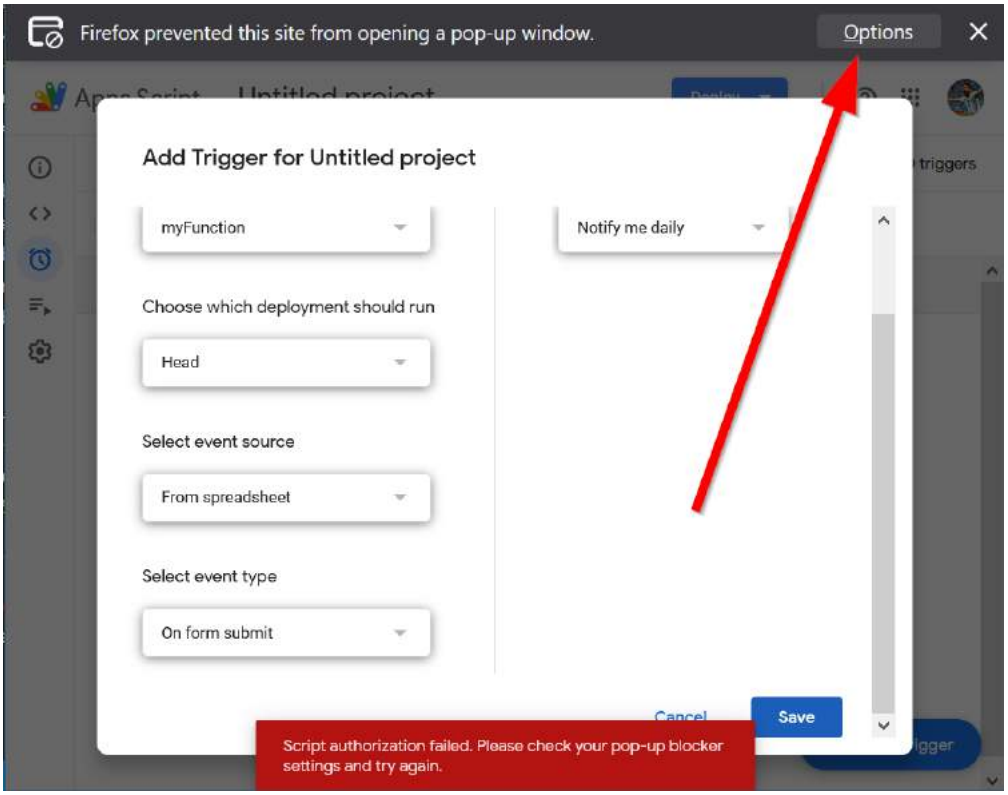
Change to **On Form Submit** so that everytime the the form gets submitted a new record gets latitude/longitude added too!



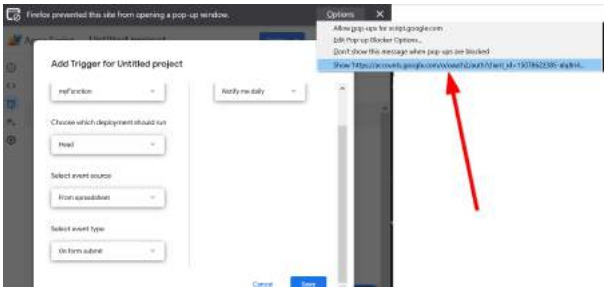
Click **Save**:



A pop-up should appear, but if you have a pop-up blocker like on FireFox, then you may have to click on `Options`:



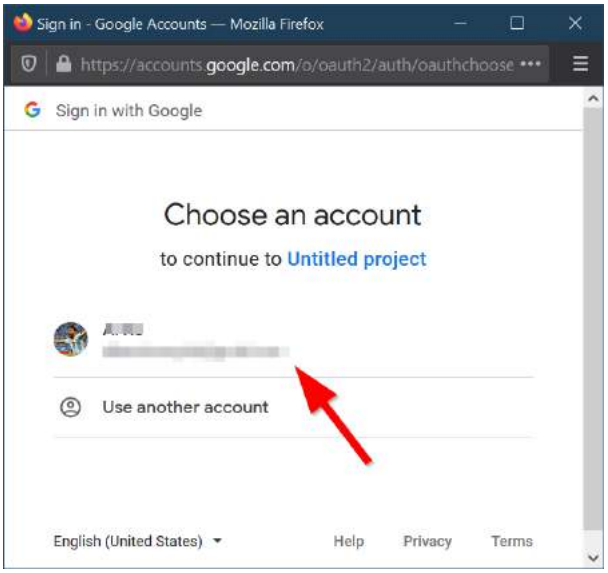
Then Allow this particular popup to appear.



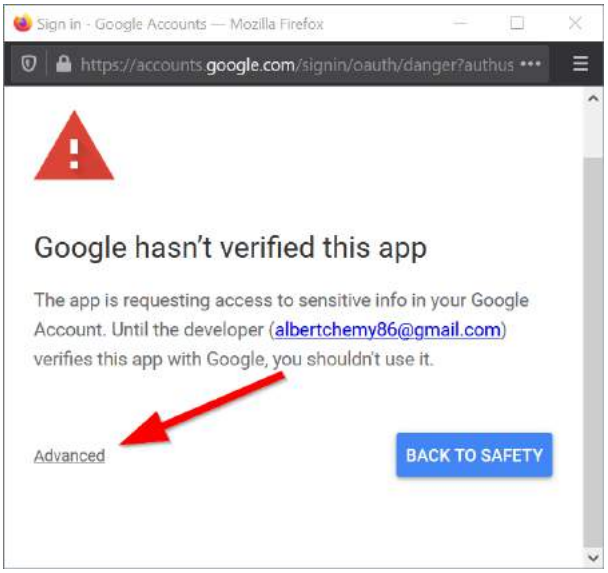
Authorization again?

This is the same authorization as before, but it is for the `Trigger` not the application!

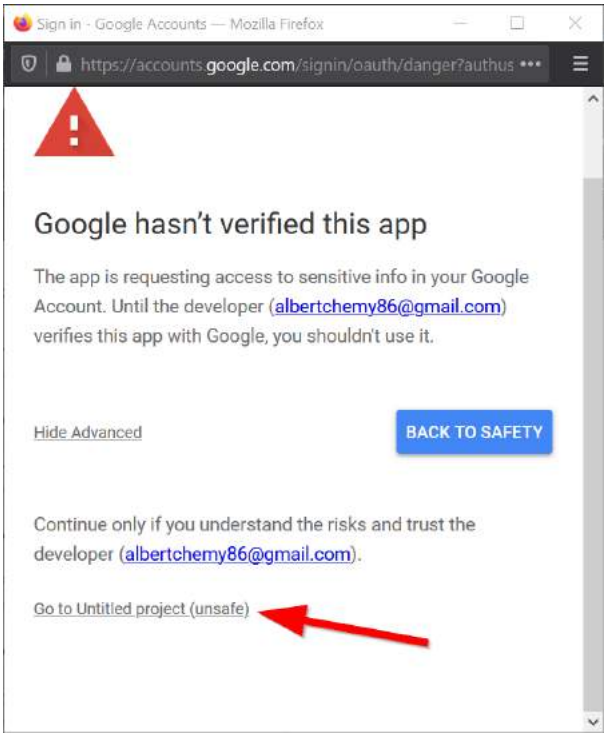
Select your **Google Account** to continue:



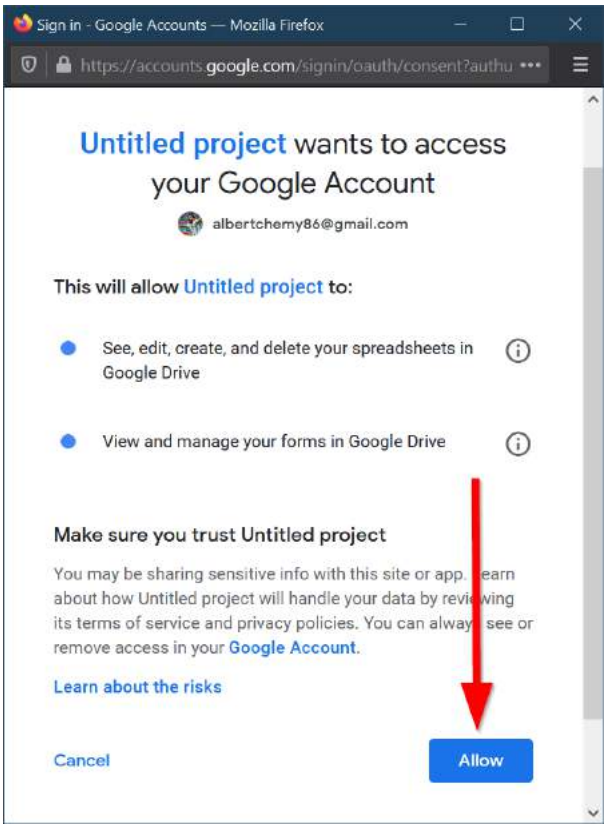
Click on Advanced:



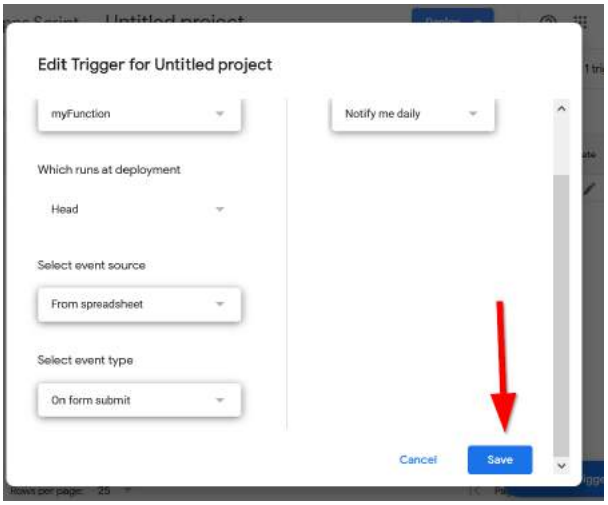
Click on **Go to Untitled Project (unsafe)**



Click on **Allow**



Click on **Save**:



Congratulations, now each time a form gets submitted you will be able to map the locations:

B	C	D	E	F
a8fc9561028d1c849		0	0	
cd98ee2830a19cd4fb		0	0	
Asian	SF	37.7749295	-122.4194155	4/23/2021 9:56:0
c8ab5bbcbad63f460c		0	0	
Asian	S	37.0366406	-95.6714121	4/23/2021 9:56:4
Asian	SF	37.7749295	-122.4194155	4/23/2021 9:58:4
abb1b919573eebefe2		0	0	
ebbe15ab609dc4ac0		0	0	
Non-Asian	SF	37.7749295	-122.4194155	4/23/2021 10:04:
Asian	405 Hilgard	34.0691706	-118.4431977	4/23/2021 10:52:
bfb354fc24d69d4ce7		0	0	
Asian	millbrae, ca	37.5985468	-122.3871942	4/23/2021 10:54:
Asian	90024	34.0631451	-118.4367551	4/23/2021 10:57:
Asian	アメリカ	37.09024	-95.712891	4/29/2021 10:59:

⚽ In-class Exercise #2 - Test your form!

Tasks

1. Add 2-3 locations to your Google Form and see if the new locations work.
2. Check to see if the locations are accurate or not!
3. What type of locations do not show up?

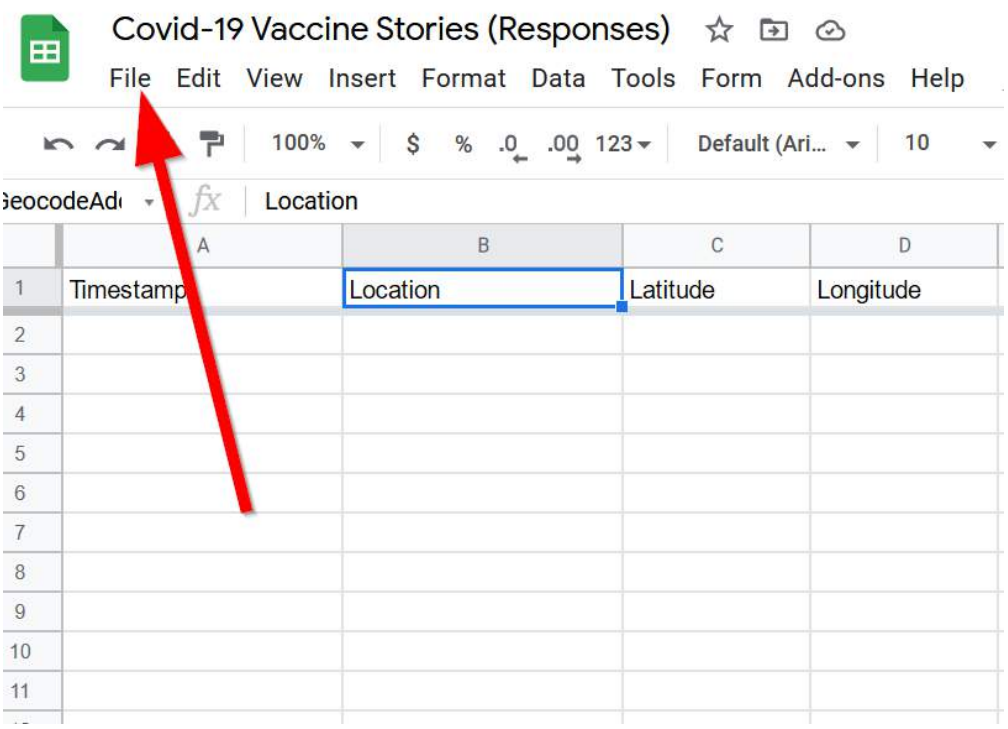
Answer

Locations that do not show up are those that are blank or are not able to be located by the Google Geocoder.

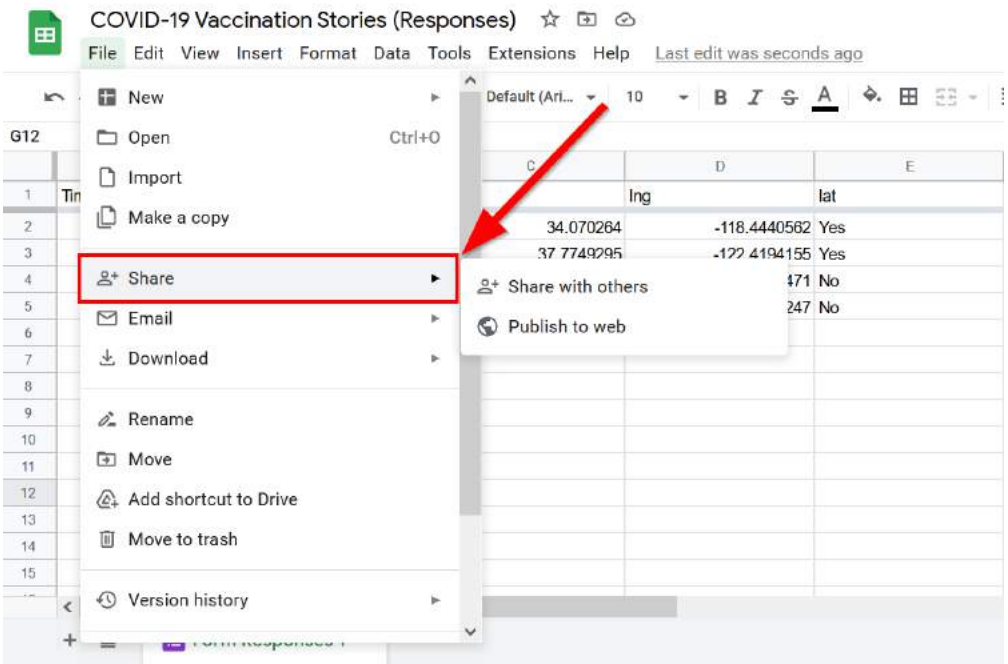
Publishing your survey

Now that our data is able to be geocoded, the final step is to publish the spreadsheet so we can can bring it into our HTML file through JavaScript next week.

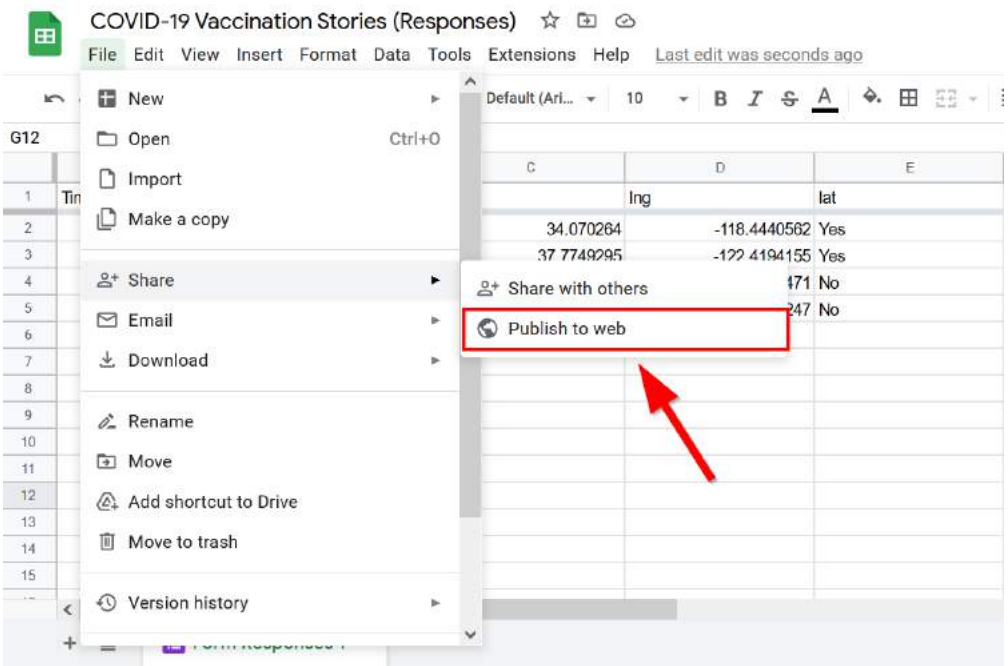
Go to **File**:



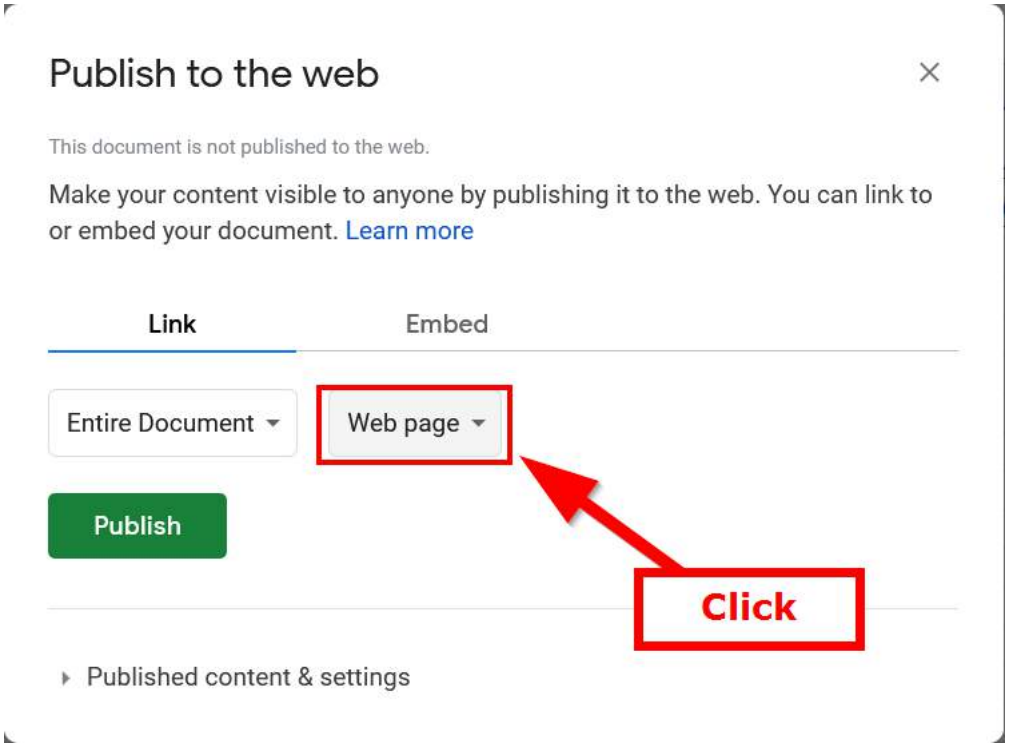
Click on **Sharing**:



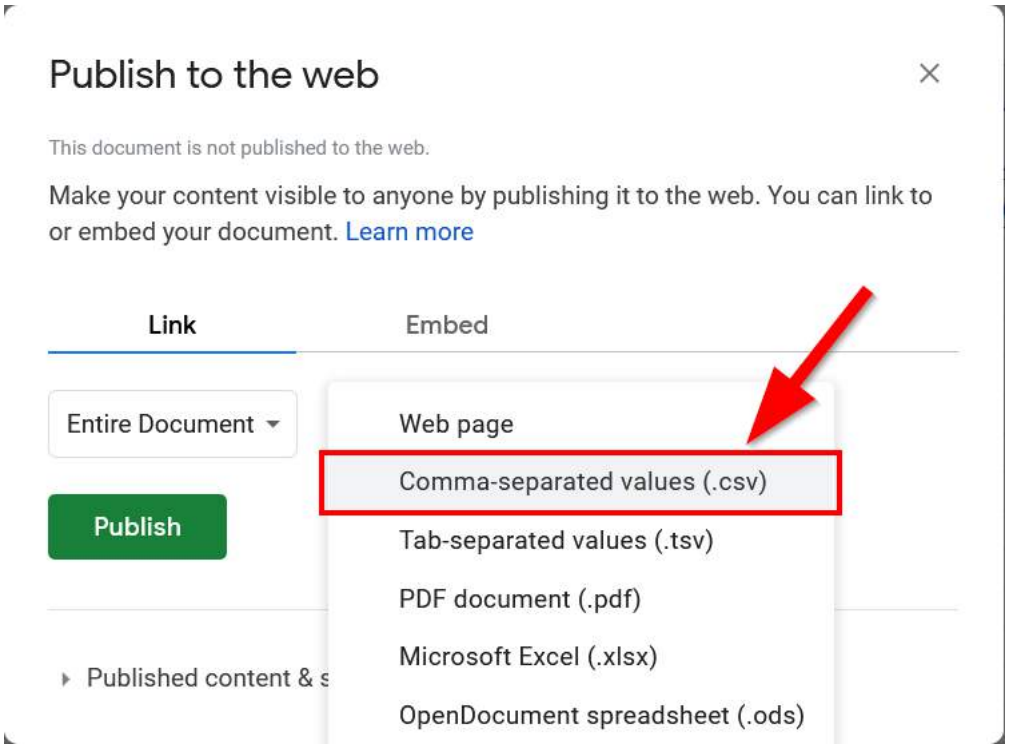
Click on **Publish to web**:



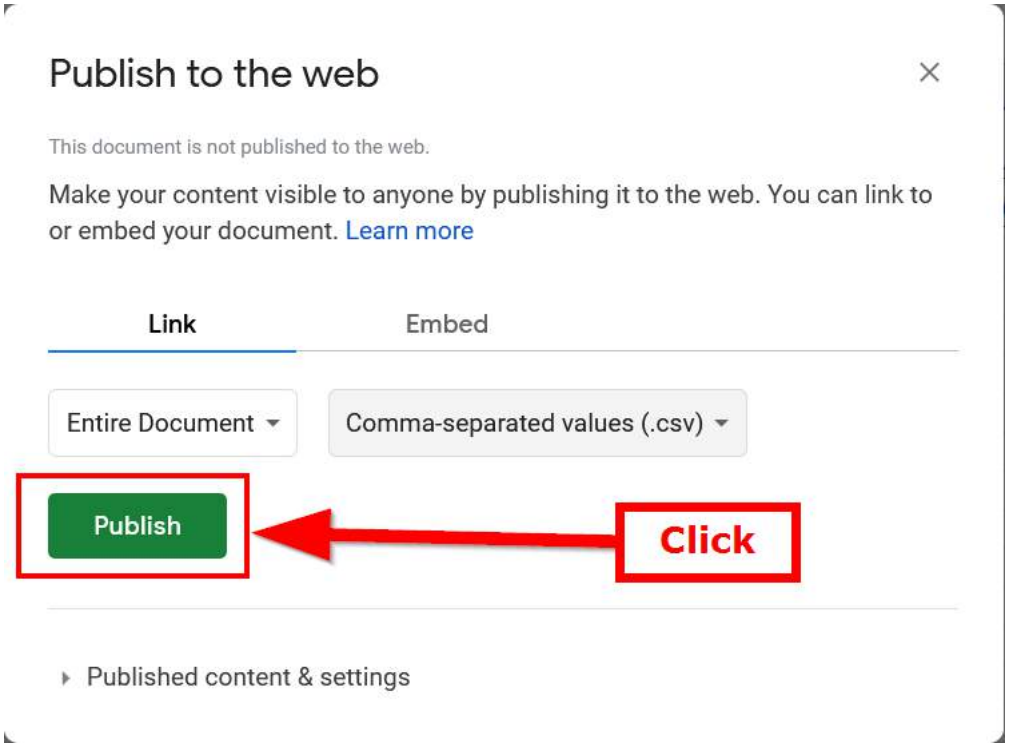
Click on **Webpage**:



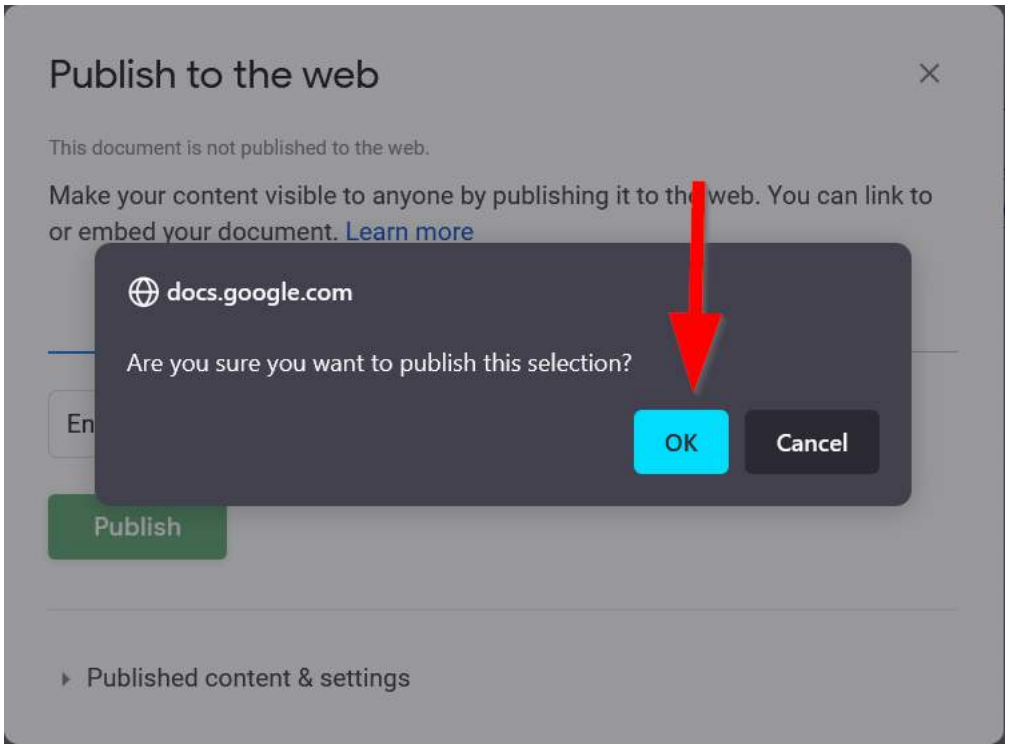
Choose **CSV**:



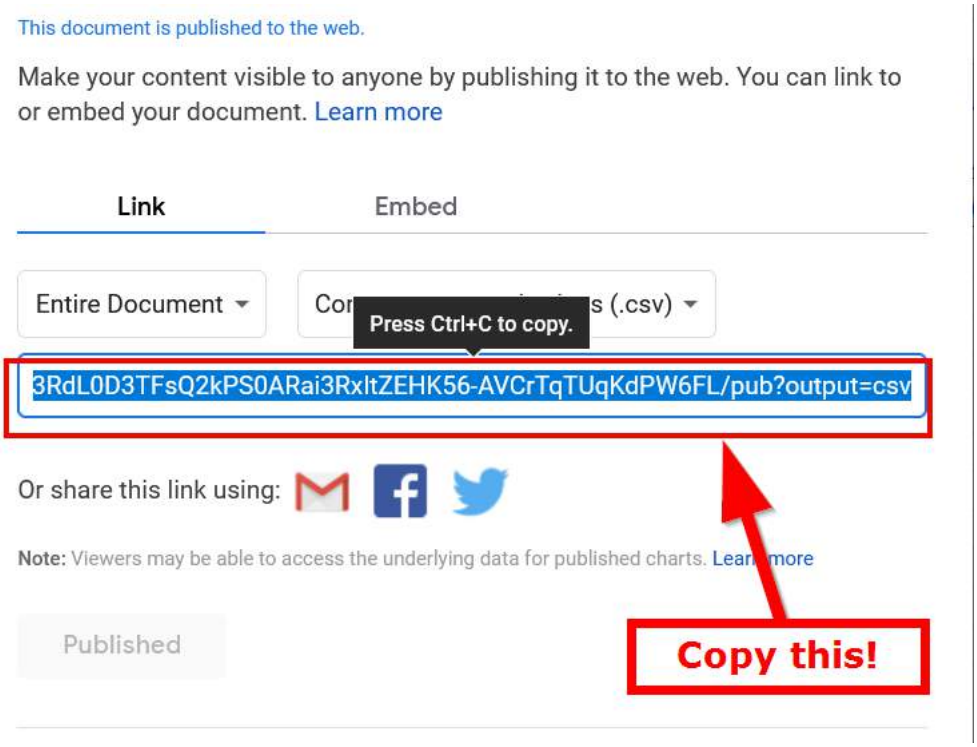
Click **Publish**:



If this warning pops-up click on **OK**:



Copy the URL in the address bar:



Paste it into your dataUrl variable like so:

js/init.js

```
const dataUrl = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSpOaH94y-oguAqbtcvZRyKdrEYiT1JOzW0jmmreznYS8THdQTYQ6cUB7J_68SZLgjpXbB_FY_nDf2A/pub?output=csv"
```

Uncomment the `loadData(url)` function to test if it's working:

```
// we will put this comment to remember to call our function later!
loadData(dataUrl)
```

Since there is only a `console.log()` in the `loadData()` function, you have to open the browser's console to check.

🏁 Last Checkpoint

Your `init.js` should look like the following:

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

const dataUrl = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSpOaH94y-
oguAqbtcvZRyKdrEYiT1JOzW0jmmreznYS8THdQTYQ6cUB7J_68SZLgjpXbB_FY_nDf2A/pub?
output=csv"

function loadData(url){
    fetch(url)
        .then(response => {
            console.log(response)
            return response
        })
        .then(data =>{
            // do something with the data
        })
}
// we will put this comment to remember to call our function later!
loadData(dataUrl)
```

There is no lab assignment for this week, but you should check to make sure your lab is working because you will use this lab for next week's assignment!

The week's final template is provided for you to double check if something isn't working.

Last update: 2023-04-27

✅ Final Template Code

index.html

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
                <iframe
src="https://docs.google.com/forms/d/e/1FAIpQLSdqVT10bEbUrULMu6Etwj4ZBXGf-
LAxcKohAINFbIdZmHS6OA/viewform?embedded=true" width="640" height="654"
frameborder="0" marginheight="0" marginwidth="0">Loading…</iframe>
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2023)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

js/init.js

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

const dataUrl = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSpOaH94y-
oguAqbtcvZRyKdrEYiT1JOzW0jmmreznYS8THdQTYQ6cUB7J_68SZLgjpXbB_FY_nDf2A/pub?
output=csv"

function loadData(url){
    fetch(url)
        .then(response => {
            console.log(response)
            return response
        })
        .then(data =>{
            // do something with the data
        })
}
// we will put this comment to remember to call our function later!
loadData(dataUrl)
```

styles/style.css

```
body{
    display: grid;
    grid-auto-rows: auto 1fr;
    grid-template-areas: "header" "main_content" "footer";
    background-color: aqua;
}

header{
    grid-area: header;
}

#footer{
```

```
    grid-area: footer;
}

.main{
    grid-area: main_content;
    grid-template-columns: 1fr 1fr;
    grid-template-areas: "main_map content";
    display: grid;
}

#contents{
    grid-area: content;
}

#the_map{
    height:80vh;
    grid-area: main_map;
}
```

Last update: 2023-04-27