Project 2:

Living in Victoria Living Safe

And Mahjong!

Authors: Zheng Qi, Heesu Ha, Tom Peddlesden, Welan Chu

Project Overview

We endeavoured to make an interactive visualisation of crime in each area of Victoria to assist in deciding the safest areas to live in .

- Victoria has some of the fastest growing Local Government Area (LGA) populations in Australia.
- More than 4 Million people call Victoria home.
- Victoria population grows around 20,000 to 150,000 people each year.

Crime is a major Factor when finding a place to live!

Our Approach

- Crime data is grouped by Local Government Area (LGA) and displays crime amounts and suburbs within.
- Use a choropleth visualisation, higher instances of crime in an LGA will result in a darker fill colour.
- Time slider plugin allows visualisation of data between 2011 to 2020
- Pop ups will display the crime type, number of occurrences in the year selected and suburbs within the LGA
- A table with filtering forms should allow postcodes, LGAs, suburbs and years to narrow scope

Creating an Interactive Visualisation

Data Sources

[Victorian Crime Data from data.vic.gov.au](#)

- https://discover.data.vic.gov.au/dataset/crime-by-location-data-table

[Suburb locality boundary](#)

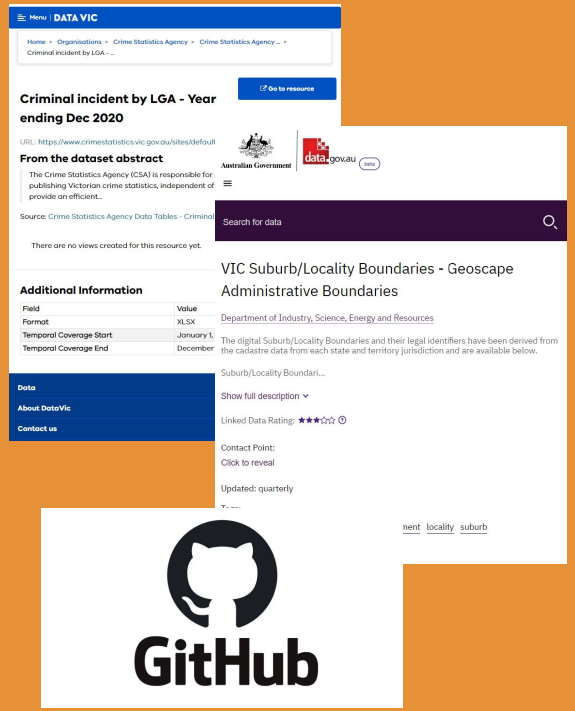
- https://data.gov.au/dataset/ds-dga-af33dd8c-0534-4e18-9245-fc64440f742e/details

[Victorian Local Government Area Boundaries:](#)

- https://data.gov.au/dataset/ds-dga-bdf92691-c6fe-42b9-a0e2-a4cd716fa811/details

[GitHub Repository: ETL Project](#)

- https://github.com/tomjp90/ETL-Project



Technologies and their roles

Python:

- Pandas
  - Data manipulation, cleaning and filtering
- Filtered data exported to mongoDB atlas using PyMongo

MongoDB Atlas:

- A remote server that stores our filtered data from python
- MongoDB Atlas as postgreSQL didn't hold enough data!!!

Flask and PyMongo:

- PyMongo used to read in data from MongoDB Atlas and do more filtering and aggregations
- Flask returns jsonified data that can be read into JavaScript through an API call
- Renders a html templates

Technologies and their roles

HTML:

- Links JavaScript with the flask app
- **CSS**
  - Bootstrap used to style and configure html elements

JavaScript:

- JS logic file renders the Mapbox map using d3 and leaflet using the following plugins:
  - Choropleth
  - Timeslider
- Uses d3.json() method to fetch JSON data via the relevant flask app route

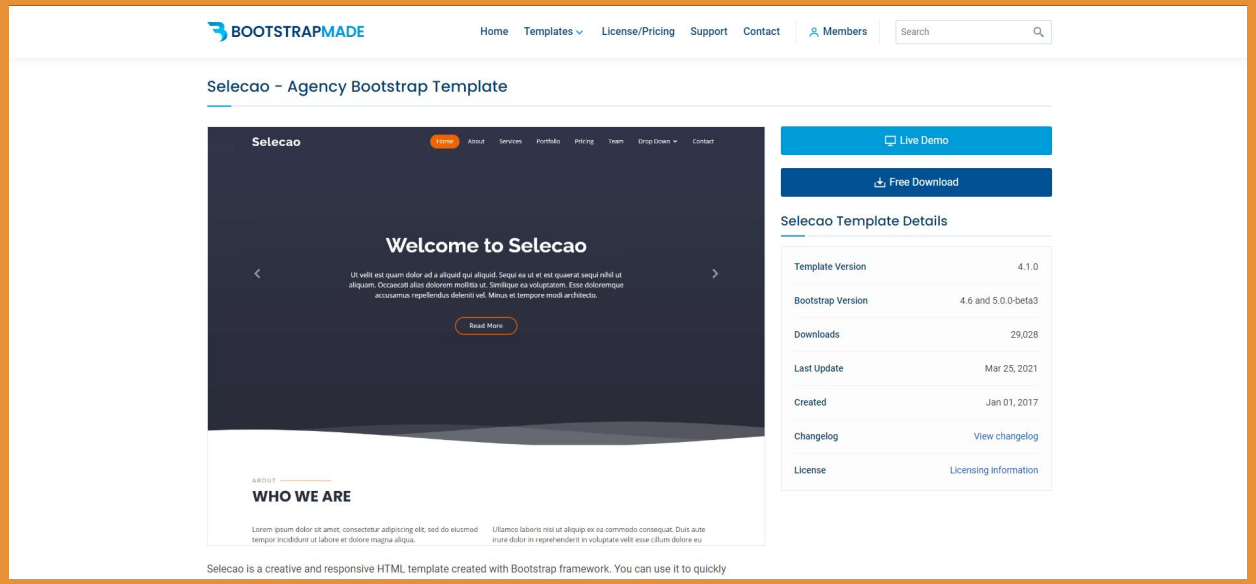
Heroku:

- App to be hosted on heroku allowing access through the web

Visualisation Ideas

our grand plans...

Selecao Bootstrap Template

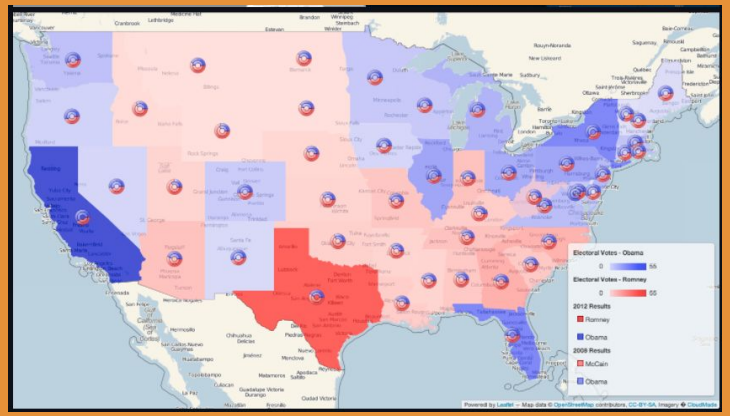


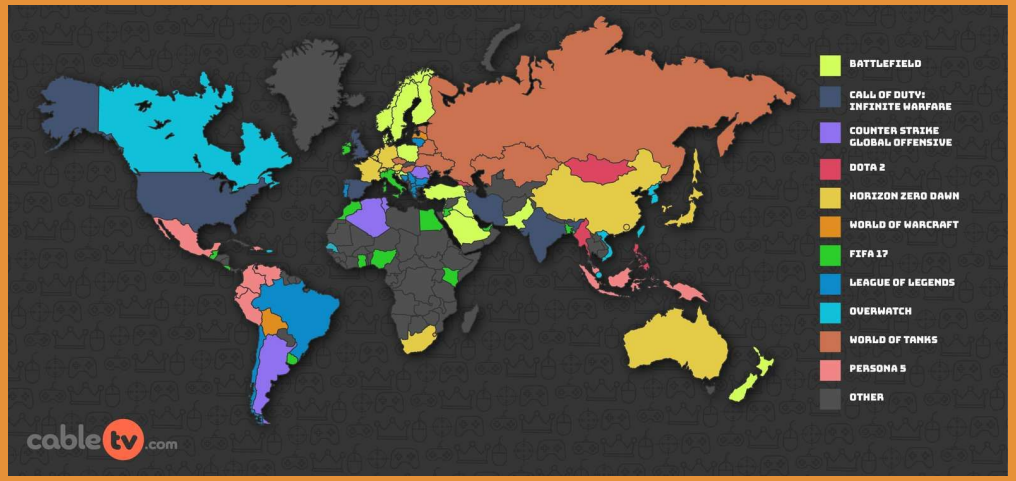
https://bootstrapmade.com/selecao-bootstrap-template/

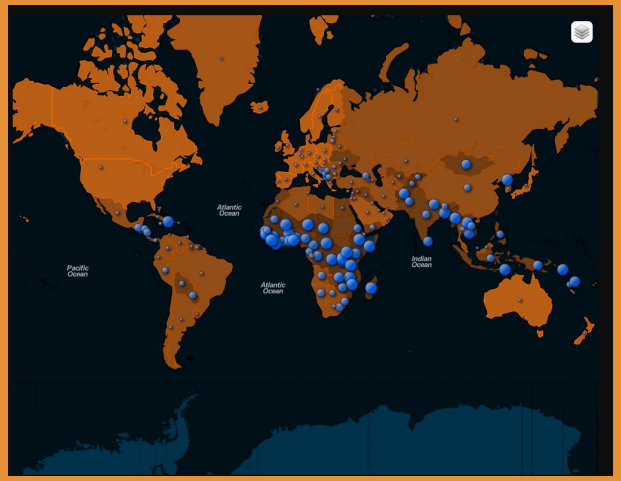
Leaflet Plugins

[Choropleth]

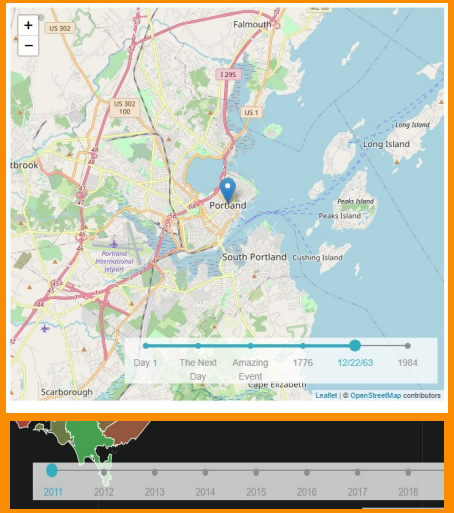
[Data Visualisation Framework]





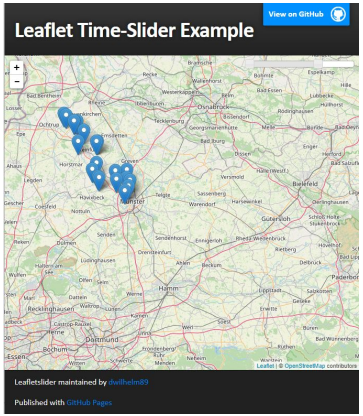


Leaflet Plugins



Timeline slider:

Map refreshes/updates based on the selected time (chosen plugin)

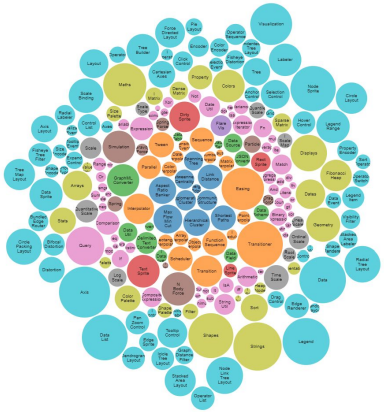


Time slider:

Markers appear based on a sliding time scale

Zoomable sunburst:

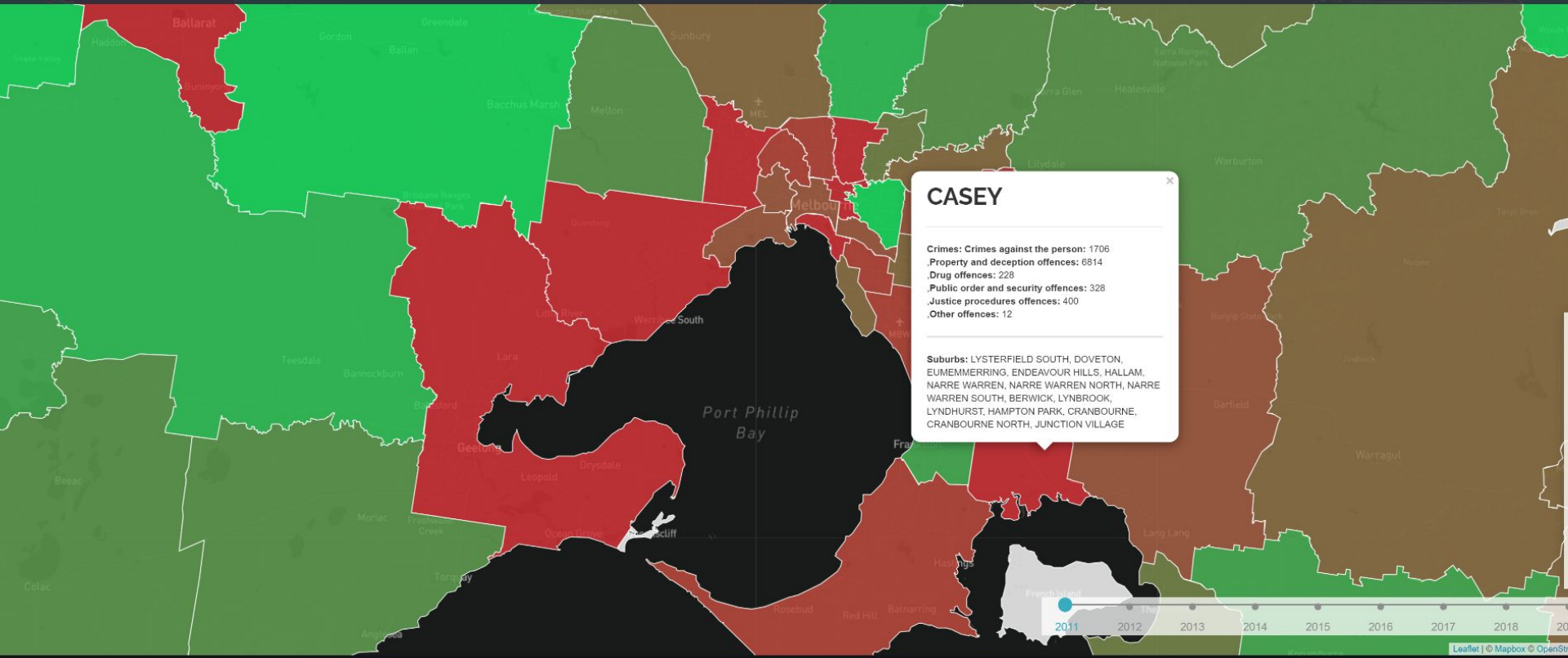
Multi-level pie chart visualisation with further details within each wedge (e.g. each wedge represents a suburb/LGA, click in to discover the applicable crime statistics



Bubble chart:

Displays data in “bubbles” based on values (e.g. radius dependent on number of crimes, etc.)

PREVIEW



Monash University Data Analytics Bootcamp 2020-2021

Project 2 - Team Members: Zheng Qi, Tom Peddlesden, Heesu Ha, Welan Chu.

Table

Filter Search

Enter a postcode

3000

Enter a suburb

melbourne

Enter a local government area

Melbourne

Enter a Year

2011

Filter Table

Year	Suburb	Postcode	LGA	A	B	C	D	E	F	Total crime
2011	melbourne	3000	Melbourne	1414	7331	404	3764	1210	52	14175
2012	melbourne	3000	Melbourne	1602	7241	465	3714	1498	39	14559
2013	melbourne	3000	Melbourne	1530	7446	793	3557	1602	29	14957
2014	melbourne	3000	Melbourne	1537	7089	703	2821	2105	40	14295
2015	melbourne	3000	Melbourne	1502	7020	614	2379	2533	30	14078
2016	melbourne	3000	Melbourne	1970	8014	620	2093	2755	33	15485
2017	melbourne	3000	Melbourne	2110	8236	673	2105	2011	27	15162
2018	melbourne	3000	Melbourne	2152	7468	616	2292	2224	28	14780
2019	melbourne	3000	Melbourne	2102	7688	682	2250	2146	21	14889
2020	melbourne	3000	Melbourne	1835	6471	765	1314	1805	1984	14174

Offence Division code	Offence Division
A	Crimes against the person
B	Property and deception offences
C	Drug offences
D	Public order and security offences

Our Code

Python

Pandas, PyMongo, Flask

Python Data Wrangling and Loading

- Crime data and geographic data are combined together.
- Crime data are classified to different types.
- Suburbs are linked to each LGA.

```
// Pseudo code of data wrangling
crimerateDF
```

	Year	Year ending	Local Government Area	postcode	suburb	Offence Division	Offence Subdivision	Offence Subgroup	Incidents Recorded
0	2020	September	Alpine	3691	dederang	B Property and deception offences	B40 Theft	B41 Motor vehicle theft	1
1	2020	September	Alpine	3691	glen creek	C Drug offences	C30 Drug use and possession	C32 Drug possession	1
2	2020	September	Alpine	3691	glen creek	F Other offences	F90 Miscellaneous offences	F93 Cruelty to animals	1

```
vicsuburbLGA_df
```

	postcode	suburb	lat	lon	Local Government Area	Region
0	3000	melbourne	-37.814563	144.970267	Melbourne	Northern Metropolitan
1	3002	east melbourne	-37.816640	144.987811	Melbourne	Northern Metropolitan
2	3003	west melbourne	-37.806255	144.941123	Melbourne	Northern Metropolitan

```
// Pseudo code of data wrangling
crimetype=crimerateDF[["Offence Division","Offence Subdivision"]]
crimetypeDF=crimetype.groupby(["Offence Division","Offence Subdivision"]).count()
```

Offence Division	Offence Subdivision
A Crimes against the person	A20 Assault and related offences
	A50 Robbery
	A70 Stalking, harassment and threatening behaviour
	A80 Dangerous and negligent acts endangering people
	Other crimes against the person

Group the crime data by offence classifications

Split the name and the code of division/subdivisions to separate columns

```
crimetypeDF.reset_index(level=["Offence Division", "Offence Subdivision"])
crimetypeDF["Offence Subdivision"].str.split(" ", 1, expand=True)
crimetypeDF["Offence Division"].str.split(" ", 1, expand=True)
```

	Offence Subdivision code	Offence Subdivision	Offence Division code	Offence Division
0	A20	Assault and related offences	A	Crimes against the person
1	A50	Robbery	A	Crimes against the person
2	A70	Stalking, harassment and threatening behaviour	A	Crimes against the person

```
// Pseudo code of data wrangling
CrimeSumDFPivot=CrimeSumDFreset.pivot(index=['Year', 'Local Government Area', 'postcode', 'suburb']
                                      columns='Offence Code', values='Incidents Recorded')
CrimeSumDFPivot.reset_index/fillna(0)/astype(int)
```

	Year	Local Government Area	postcode	suburb	A20	A50	A70	A80	A90	B10	...	D10	D20	D30	D40	E10
0	2011	Alpine	3691	dederang	1	0	0	0	0	0	...	0	1	0	0	0
1	2011	Alpine	3691	kancoona	0	0	0	0	0	0	...	0	0	0	0	0
2	2011	Alpine	3691	upper gundowring	0	0	0	0	0	0	...	0	0	0	0	0

Use pivot table to unstack the crime data of each division

Clean data by index reset/fillna/astype

```
CrimeDF=[]
for year in range(2011,2021):
    CrimeDF.append(CrimeSumDFPivotreset[CrimeSumDFPivotreset['Year']==year])
for idx in range(len(CrimeDF)):
    CrimeDF[idx]=vicsuburbLGA_df.merge(CrimeDF[idx],how='left',on=['postcode','suburb'])
CrimeSuburbYearDF=pd.concat(CrimeDF, ignore_index=True)

for subdiv_code in subdiv_code_list:
    CrimeSuburbYearDF[div_code]+=CrimeSuburbYearDF[subdiv_code]
    CrimeSuburbYearDF['Total']+=CrimeSuburbYearDF[subdiv_code]
```

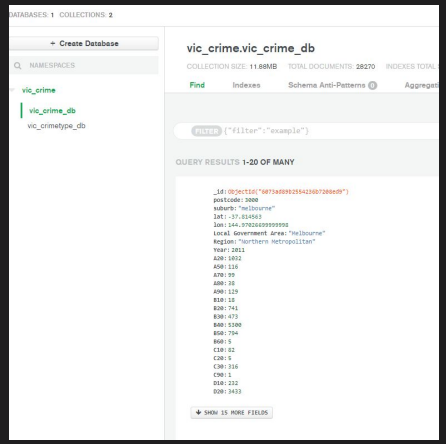
Merge crime data with suburb information data

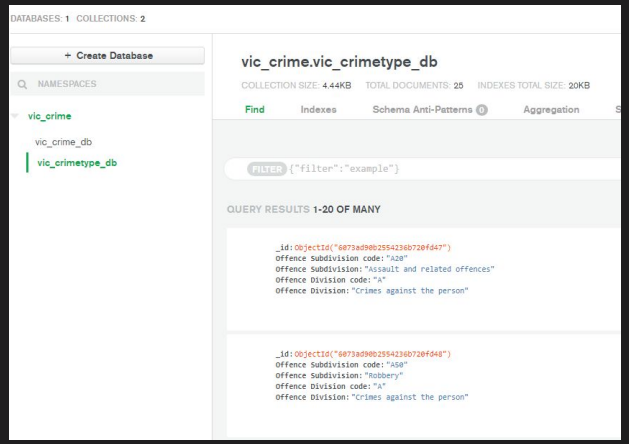
	postcode	suburb	lat	lon	Local Government Area	Region	Year	A20	A50	A70	...	F20	F30	F90	Total	A	B	C	D	E	F
0	3000	melbourne	-37.814563	144.970267	Melbourne	Northern Metropolitan	2011	1032	116	99	...	13	36	3	14175	1414	7331	404	3764	1210	52
1	3002	east melbourne	-37.816640	144.987811	Melbourne	Northern Metropolitan	2011	53	12	4	...	0	9	0	753	76	476	32	149	11	9
2	3003	west melbourne	-37.806255	144.941123	Melbourne	Northern Metropolitan	2011	54	9	3	...	2	1	2	633	80	403	32	107	6	5

Sum up the division/total crime data for each suburb

```
// Pseudo code of data loading to Mongodb Atlas
client =
pymongo.MongoClient("mongodb+srv://<username>:<password>@cluster0.pyqix.mongodb.net/vic_crime?retryWrites=true&w
=majority")
vic_db = client['vic_crime']

vic_db.vic_crime_db.drop()
vic_db.vic_crimetype_db.drop()

vic_db.vic_crime_db.insert_many(CrimeSuburbYearDF.to_dict('records'))
vic_db.vic_crimetype_db.insert_many(crimetypeDFreset.to_dict('records'))
```

Upload the clean data to Mongodb Atlas Cluster



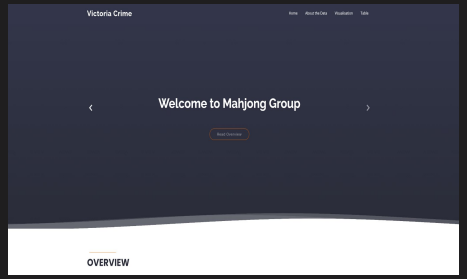


Flask and Pymongo

- Renders a html template
- PyMongo used to read in data from MongoDB Atlas and do more filtering and aggregations
- Flask returns jsonified data that can be read into JavaScript through an API call

```python
// Pseudo code for flask html rendering in flask
API_KEY=os.getenv("API_KEY")

@app.route("/")
@app.route("/index")
def Welcome():
    return render_template("index.html")

@app.route("/data")
def data():
    return render_template("data.html")

@app.route("/visualisation")
def visualisation():
    return render_template("visualisation.html", API_KEY=API_KEY)

@app.route("/table")
def table():
    return render_template("table.html")
```

Rendering html files



Passing API_KEY to html and javascript file for leaflet/map loading

```python
// Pseudo code for /api/v3.0/lga/all
@app.route("/api/v3.0/lga/all")
def lga_all_crime_3():
    off_field=request.args.getlist('off_field')
    groupby = ["Year","Local Government Area","Region"]
    group = {
        '_id': ["$%s" % (x if x else None) for x in groupby],
        'Total': {'$sum': "$Total"}
        }
    if "div" not in off_field:
        for code in crimetp_dic["Offence Division code"]:
            group[code]={'$sum': "$%s"%code}
    if "subdiv" not in off_field:
        for code in crimetp_dic["Offence Subdivision code"]:
            group[code]={'$sum': "$%s"%code}

    crime=vic_db.vic_crime_db.aggregate([{"$group":group}])
    crimetp=vic_db.vic_crimetype_db.find({},{"_id":0})

    """Return a list of all crime sum by lga/year"""
    lga_crime=function (crime, crimetp)
    return jsonify(lga_crime)
```

Similar to the following scheme in SQL

```sql
SELECT year, lga, region,
Sum(Total) AS Total,
Sum(Div) AS Div,
Sum(Subdiv) AS Subdiv
FROM crime
GROUP BY Date(date)
ORDER BY year, lga, region
```

Use mongodb group/aggregate function

```python
// Pseudo code for /api/v3.0/crime_data
@app.route('/api/v3.0/crime_data')
def crime_data_json():
    postcode=request.args.getlist('postcode')
    suburb=request.args.getlist('suburb')
    lga=request.args.getlist('lga')
    region=request.args.getlist('region')
    year=request.args.getlist('year')
    query={}
    if (len(postcode)): query["postcode"]={"$in": postcode}
    if (len(suburb)): query["suburb"]={"$in": suburb}
    if (len(lga)): query["Local Government Area"]={"$in": lga}
    if (len(region)): query["Region"]={"$in": region}
    if (len(year)): query["Year"]={"$in": year}

    crime=vic_db.vic_crime_db.find(query,{"_id":0})
    crimetp=vic_db.vic_crimetype_db.find({},{"_id":0})
    """Return a dict of filtered data"""
    all_crime=function (crime, crimetp)
    return jsonify(all_crime)
```

https://vic-crime.herokuapp.com/api/v3.0/crime_data?suburb=melbourne&year=2011&year=2020

Get input query list

Use mongodb query function

Similar to the following scheme in SQL

```sql
SELECT *
FROM crime
WHERE
Postcode in postcode_query
Suburb in suburb_query
Lga in lga_query
Region in region_query
Year in year_query
```

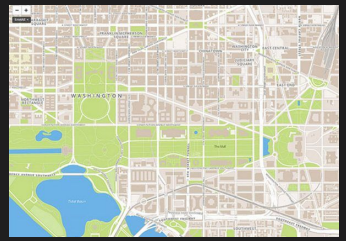
Javascript

Leaflet, Geomap..... others

```
// Creating map object
var myMap = L.map("map", {
    center: [-37.814563, 144.97026699999998],
    zoom: 20
});

// Adding tile layer
L.tileLayer("https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}", {
    attribution: "© <a href='https://www.mapbox.com/about/maps/'>Mapbox</a> © <a
href='http://www.openstreetmap.org/copyright'>OpenStreetMap</a> <strong><a
href='https://www.mapbox.com/map-feedback/' target='_blank'>Improve this map</a></strong>",
    tileSize: 512,
    maxZoom: 18,
    zoomOffset: -1,
    id: "mapbox/streets-v11",
    accessToken: API_KEY
}).addTo(myMap);
```

Variable to initialise leaflet.js map and place into html where a tag with the id “map” exists

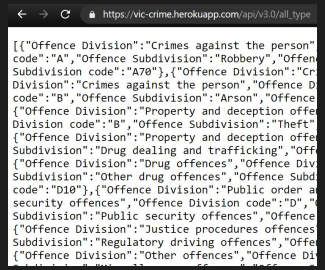
Adding base map tile layer to the leaflet.js map variable “myMap”



```
// Use this link to get the geojson data.
const lgaAPI = "https://opendata.arcgis.com/datasets/0f6f122c3ad04cc9bb97b025661c31bd_0.geojson";
// const lgaAPI = "../static/data/LGA.geojson" // Only use this if the variable above does not work!
const suburbAPI =
"https://data.gov.au/geoserver/vic-suburb-locality-boundaries-psma-administrative-boundaries/wfs?request=GetFea
ture&typeName=ckan_af33dd8c_0534_4e18_9245_fc64440f742e&outputFormat=json";
var geojson;

const lgaCrimeData = "/api/v3.0/lga/all?off_field=subdiv";
const crimeTypes = "/api/v3.0/all_type";
const suburbsLga = "/api/v3.0/lga/all_suburb"
```

API calls to GeoJSONs: LGA API for Local Government Area (LGA) boundaries and suburbAPI for suburb boundaries, both as polylines

lgaCrimeData & crimeTypes calls our flask app routes to retrieve the necessary crime data

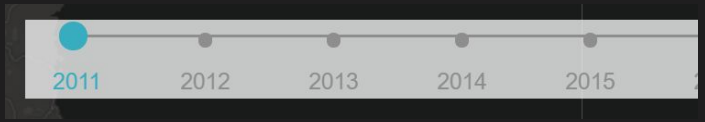


```
// Custom function to change the map per our API calls
// Timeline slider plugin for leaflet, can be placed within a function
// Refer to docs: https://github.com/svitkin/leaflet-timeline-slider
L.control.timelineSlider({
      timelineItems: [
            "2011", "2012", "2013",
            "2014", "2015", "2016",
            "2017", "2018", "2019",
            "2020"
      ],      // timeline dates are created using an array of strings
      changeMap: getDataAddMarkers,       // custom function to update the map based on the timeline items
      position: "bottomright"      // default is "bottomright" if this is preferred
      // extraChangeMapParams: {exclamation: "Hello World!" }       // extra parameters that can be read by the
function in changeMap
}).addTo(myMap);
```

Timeline Slider Plugin

Sets the times in the timeline, a function that changes the map according to time selected (and other options)

Add to the map reference in the HTML



```
function getDataAddMarkers({ label, value, map }) {
    console.log(`Timeline slider is set to ${parseInt(label)}`);

    // Clear the choropleth layer at the start of every timeline slider change (including init)
      map.eachLayer(function (layer) {
            if (geojson) {
                  map.removeLayer(geojson);
            }
      })

      var legendRemove = d3.select(".legend");
            legendRemove.remove();
```

Function that the Timeline Slider calls to create the Choropleth

Clear existing choropleth layer and legend (useful for when the user changes to different years in the data)

```
// Append crime data to the LGA GeoJSON
d3.json(lgaCrimeData, function (cData) {
    var crimeDivisions = {};
    var crimeJSON = cData[label];
    var valueCrime = [];
        for (let i = 0; i < data.features.length; i++) {
            var lgaProperties = data.features[i].properties;

            for (let lga in crimeJSON) {
                if (lgaProperties.ABB_NAME == lga.toUpperCase()) {

                    lgaProperties.CRIME_TOTAL = crimeJSON[lga].crime.Total;

                    valueCrime.push(parseInt(crimeJSON[lga].crime.Total))
                }
                else {
                    continue;
                }
            };
        };
```

Access our flask app.py, make API calls and retrieve a JSON of our data using d3.js
The variable *label* contains the year that we are filtering our data on, set by the timeline slider.

In order to display the data as a choropleth, we need to append our crime JSON data to the LGA.geoJSON (external source)

Achieved by creating a new object with the key *CRIME_TOTAL* and assigning the value from our data

```
// Create a new choropleth layer
geojson = L.choropleth(data, {
    // Define what  property in the features to use
    valueProperty: "CRIME_TOTAL",
    // Set color scale
    scale: ['#00ffad','#e9002c'],
    // Number of breaks in step range
    steps: 10,
    // q for quartile, e for equidistant, k for k-means
    mode: "q",
    style: { color: "#fff", weight: 1, fillOpacity: 0.8 },
    // Binding a pop-up to each layer
    onEachFeature: function (feature, layer) {
        // Set mouse events to change map styling
        layer.on({
            mouseover: function (event) { layer = event.target;
                layer.setStyle({ fillOpacity: 1 });
            },
            // When the cursor no longer hovers over a map feature - when the mouseout event occurs - the feature's opacity reverts back to 50%
            mouseout: function (event) {
                layer = event.target;
                layer.setStyle({ fillOpacity: 0.8 });
            },
            // When a feature (neighborhood) is clicked, it is enlarged to fit the screen
            click: function (event) {
                myMap.fitBounds(event.target.getBounds());
            }
        });
        // call getLGACrime function, parse LGA Name in capitalised format to match our lgaCrimeData json
        layer.bindPopup(`<h><b> ${feature.properties.ABB_NAME} </b></h2>
            <hr><h5>${getLGACrime(feature.properties.ABB_NAME.trim().toLowerCase().replace(/\w\S*/g, (w) => (
c.toUpperCase()))))}</h5><hr>`
        );
    }
}).addTo(myMap)
```

Using choropleth plugin, create the choropleth and add detailed crime data to each LGA

Mouse events to enrich the visualisation experience, including revealing detailed crime data

Demonstration

https://vic-crime.herokuapp.com/

Questions ?