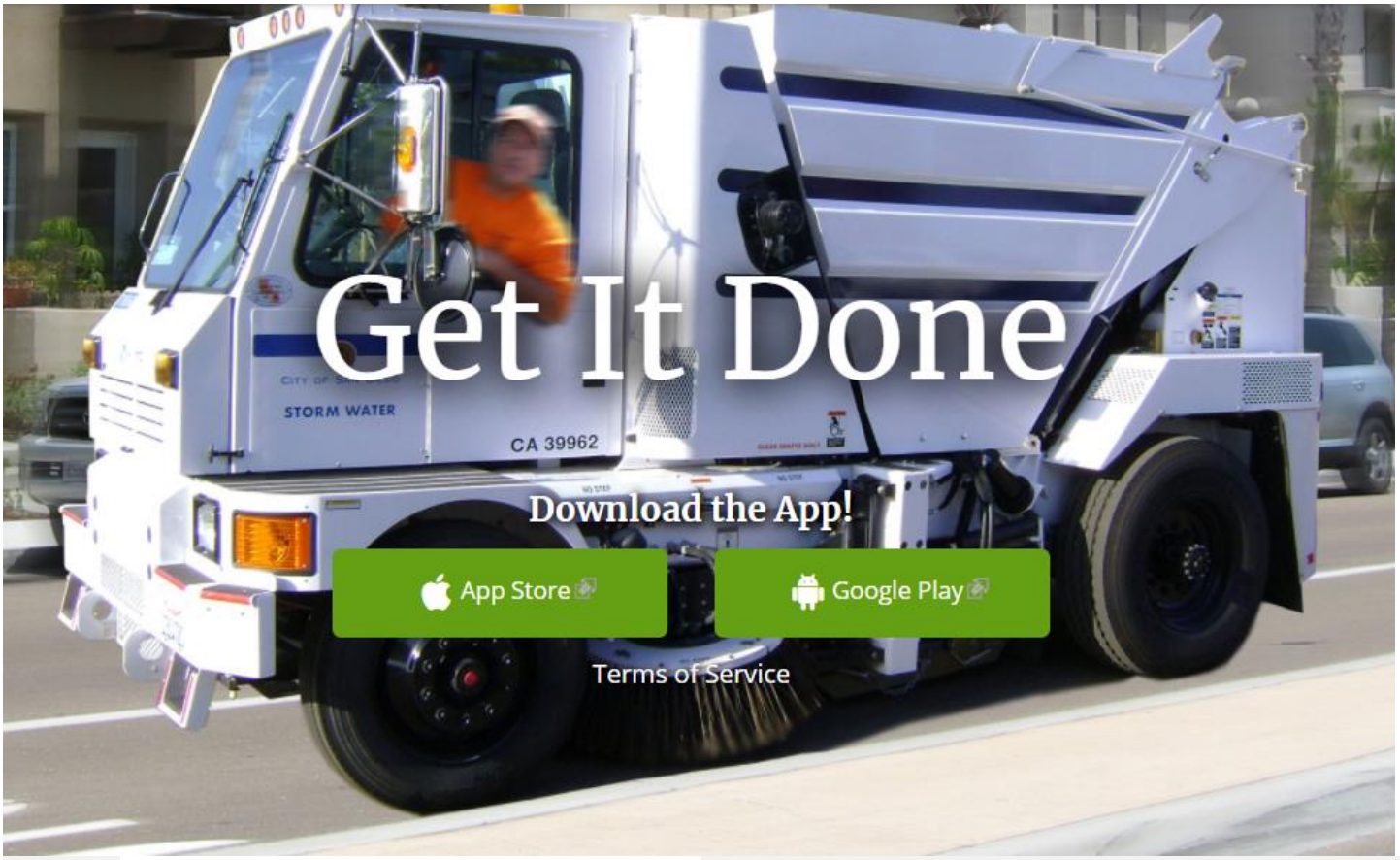


CITY OF SAN DIEGO GET-IT-DONE DASHBOARD

FINAL REPORT FOR PROJECT 2

UCSD Data Science and Visualization Bootcamp
April 2020

Team Git it On:
Melissa Monroe | Architecture, design, data, Leaflet map, art
Andrew Bankston | Data, content, analysis, plots
Jacob Zacarias | Data, content, analysis, plots, maps
Annette Broeren | Powerpoint, final report, ReadMe

Table of Contents

ARTICULATION AND RATIONALE

The objective of this report is to describe the goal, process and observations of transforming a dataset into a dynamic dashboard for business metrics. For this project, City of San Diego's Get-It-Done app data was used.

Get It Done San Diego is the official app for reporting non-emergency problems to the City of San Diego. App users can report problems like potholes or graffiti and connect directly to the City's work tracking system.

The app has been in use since 2016 and the City of San Diego publishes annual Get-It-Done datasets in CSV format for historical data, and JSON format for the current year to date.

Intended audience: City of San Diego Leadership

Decision: Strategic/Operational – identify problem-areas, monitor and gage effectiveness of the Get-It-Done program overall and by Council district, using:

- Service request volume by period, type, council district
- Average response time from open to closed
- Open/closed deltas
- Service request volumes by type by council district

Decision timing: Ongoing for the life of the app

Importance: The dashboard can assist City Leadership in quickly identifying problem areas or problem periods, such as holidays, tourist season, etc.

What actions can be taken: The dashboard is a monitoring tool for City Leadership allowing for further investigation, remedying or use a pro-active approach as appropriate for the issue. For instance, during tourist season or while large conventions are in town, it is possible that mobile service requests for scooters spike. If the City had historical data to confirm this, the City could take a pro-active approach and request that the scooter service vendors increase their collection efforts during those times, thereby reducing the number of service requests, the amount of resouces the City would otherwise have to dedicate, and last but not least: reduce the nuisance factor for the public.

SETTING UP

SOURCES

City of San Diego Get It Done program files:
https://data.sandiego.gov/datasets/get-it-done-311/

Used for Project:

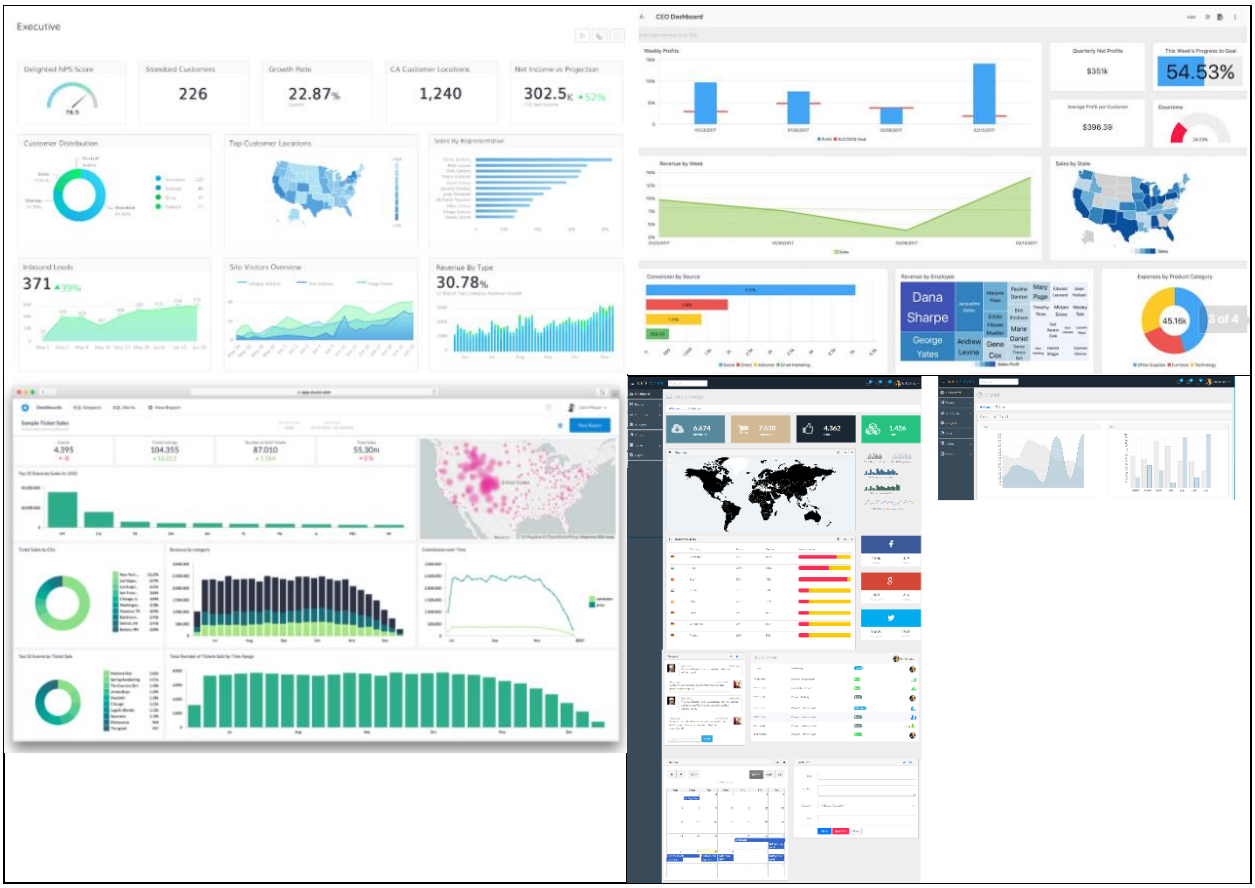
- Get It Done Requests JSON API – json
- Get It Done Requests year-to-date – csv
- Get It Done Requests 2020 – csv
- Get It Done Requests 2019 – csv

Not used but listed for future consideration:

- Get It Done Requests 2018 – csv
- Get It Done Requests 2017 – csv
- Get It Done Requests 2016 – csv9 CSV files

City Council Districts geojson:
https://data.sandiego.gov/datasets/city-council-districts/

INSPIRING VISUALIZATIONS

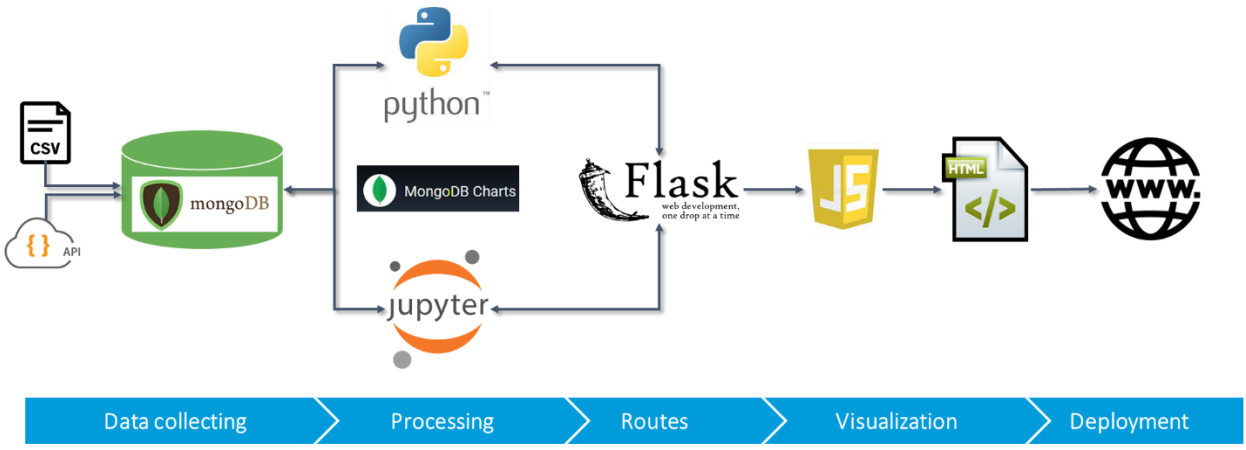


ORIGINAL SKETCHES

EXECUTION

The approach consisted of a two-tier environment where data from the CSVs is imported into a MongoDB database, and the year-to-date data published in JSON-format is called in via API.

- Download the source files to a Github repository with branches for all team members
- Create Atlas Cloud MongoDB
- Create documents from the CSV files in MongoDB converting strings to numbers or date values where appropriate
- Connect to mongodb, read collections, add group by and count service name, filter by date
- Create graphs in MongoDB Charts
- Create pre-populated summary and chart documents in MongoDB to reduce dashboard loading time
- Use NiceAdmin dashboard template and modify for purpose
- Use Python, Jupyter, Flask, Javascript, HTML/CCS to process and load data
- Populate and deploy dashboard



As of time of writing, deployment is in flux.

LIBRARIES

New and mention-worthy:

DNS Enables connection to Atlas MongoDB.

Flask CORS Cross origin resource sharing (enables running API calls when origin is different from web origin).

Pytz Accurate cross-platform time calculation.

Django Stores datetime information in UTC, uses time-zone-aware datetime objects internally, and translates them to the end user's time zone.

All libraries / dependencies

App.py – Flask Routes

```
import os
from flask import (Flask,
  render_template, jsonify, request,
  redirect)
import pymongo
from src import config
from bson.json_util import dumps
import json
import requests
import dns
import datetime
from datetime import datetime
import pytz
from django.utils import timezone
from flask import CORS
```

Jupiter Notebook– Data gathering

```
import requests
import pymongo
import dns
import datetime
from datetime import datetime
from dateutil import parser
import pytz
from django.utils import timezone
from time import sleep
import json
import config
import requests

#Project defined dependencies
import config
from pprint import pprint
```

Python – Class Visualizer

```
import pymongo
import dns
import datetime
import json
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import config
```

Jupiter Notebook – Visualizations

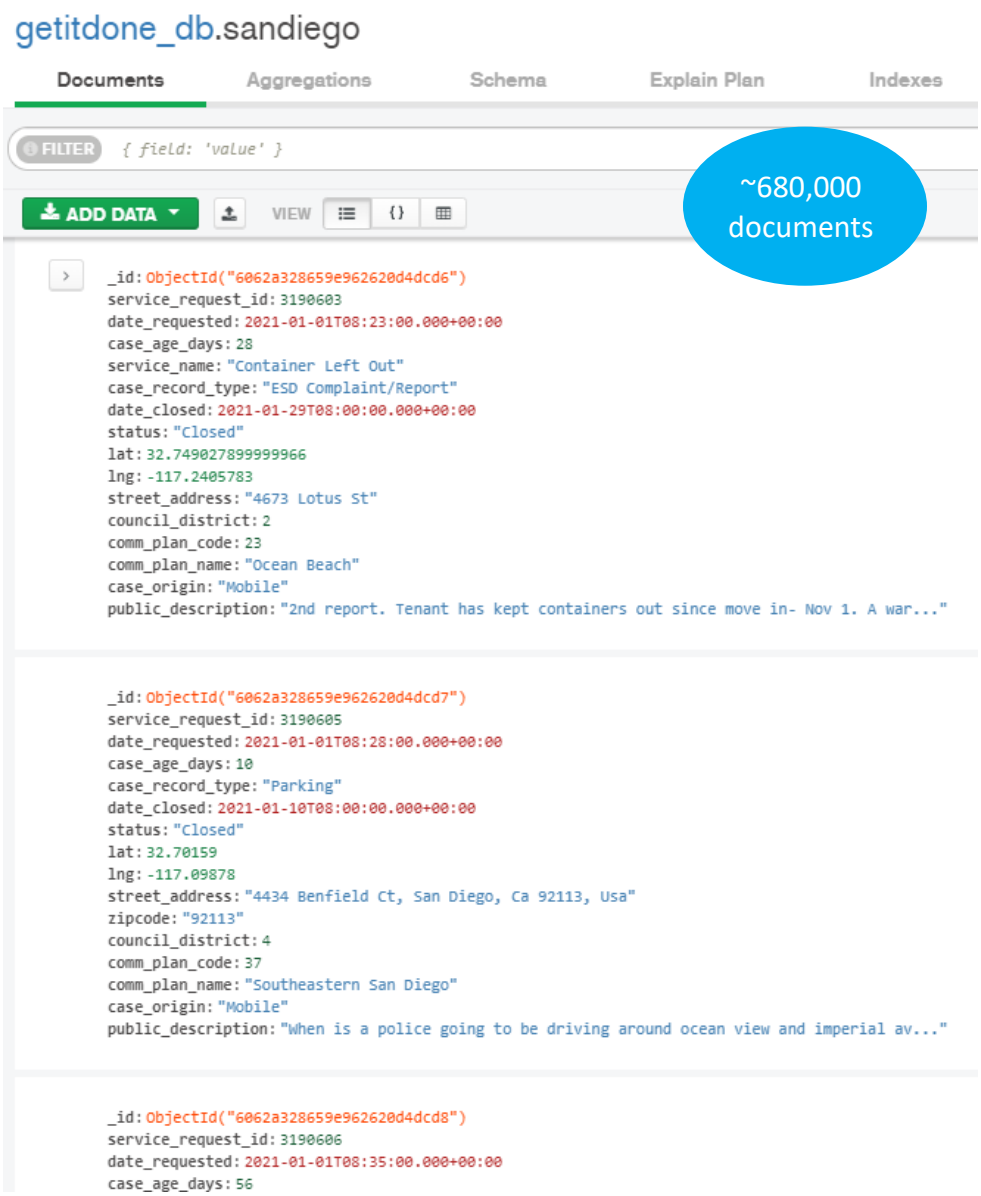
```
from matplotlib import pyplot as
  plt
import pandas as pd

#Project defined modules
import config
import visualization as viz
```

MONGODB

Approximately 680,000 service requests for 2019 and 2020 were imported into MongoDB. Strings were converted to numbers or dates where appropriate.

Column1	Column2	Column3
num	service_request_id	json
num	service_request_parent_id	
num	sap_notification_number	
date	date_requested	requested_c
num	case_age_days	calculate
	service_name	json
	case_record_type	service_cod
date	date_closed	calculate ba
	status	json
double	lat	json
double	lng	json
	street_address	address
	zipcode	
	council_district	
num	comm_plan_code	
num	comm_plan_name	
	park_name	
	case_origin	
	referred	
	public_description	
	iamfloc	
	floc	

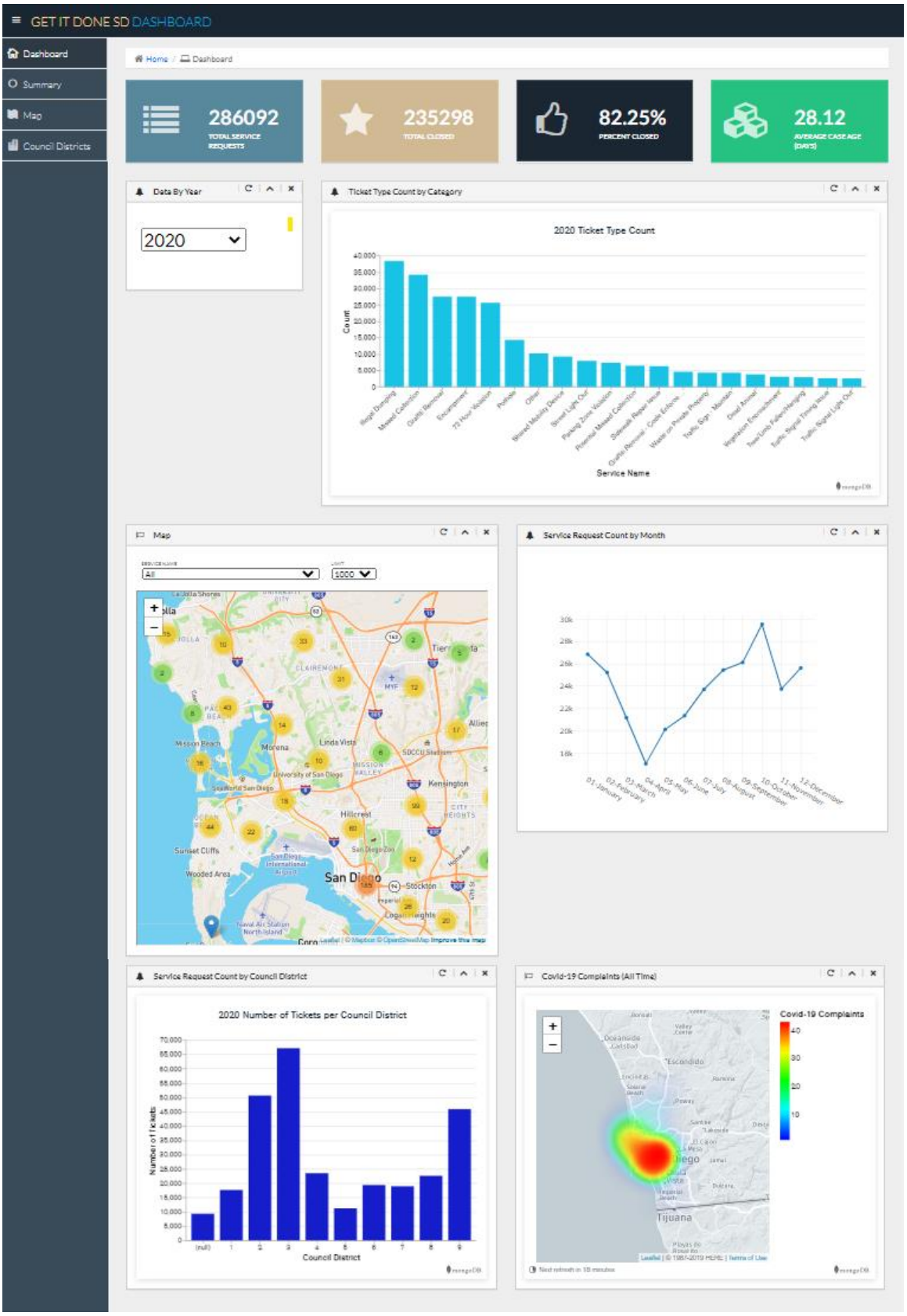

getitdone_db.sandiego
Documents
Aggregations
Schema
Explain Plan
Indexes
FILTER
{ field: 'value' }
ADD DATA
VIEW
~680,000 documents
_id: ObjectId("6062a328659e962620d4dcd6")
service_request_id: 3190603
date_requested: 2021-01-01T08:23:00.000+00:00
case_age_days: 28
service_name: "Container Left Out"
case_record_type: "ESD Complaint/Report"
date_closed: 2021-01-29T08:00:00.000+00:00
status: "Closed"
lat: 32.749027899999966
lng: -117.2405783
street_address: "4673 Lotus St"
council_district: 2
comm_plan_code: 23
comm_plan_name: "Ocean Beach"
case_origin: "Mobile"
public_description: "2nd report. Tenant has kept containers out since move in- Nov 1. A war..."
_id: ObjectId("6062a328659e962620d4dcd7")
service_request_id: 3190605
date_requested: 2021-01-01T08:28:00.000+00:00
case_age_days: 10
case_record_type: "Parking"
date_closed: 2021-01-10T08:00:00.000+00:00
status: "Closed"
lat: 32.70159
lng: -117.09878
street_address: "4434 Benfield Ct, San Diego, Ca 92113, Usa"
zipcode: "92113"
council_district: 4
comm_plan_code: 37
comm_plan_name: "Southeastern San Diego"
case_origin: "Mobile"
public_description: "When is a police going to be driving around ocean view and imperial av..."
_id: ObjectId("6062a328659e962620d4dcd8")
service_request_id: 3190606
date_requested: 2021-01-01T08:35:00.000+00:00
case_age_days: 56

DASHBOARD

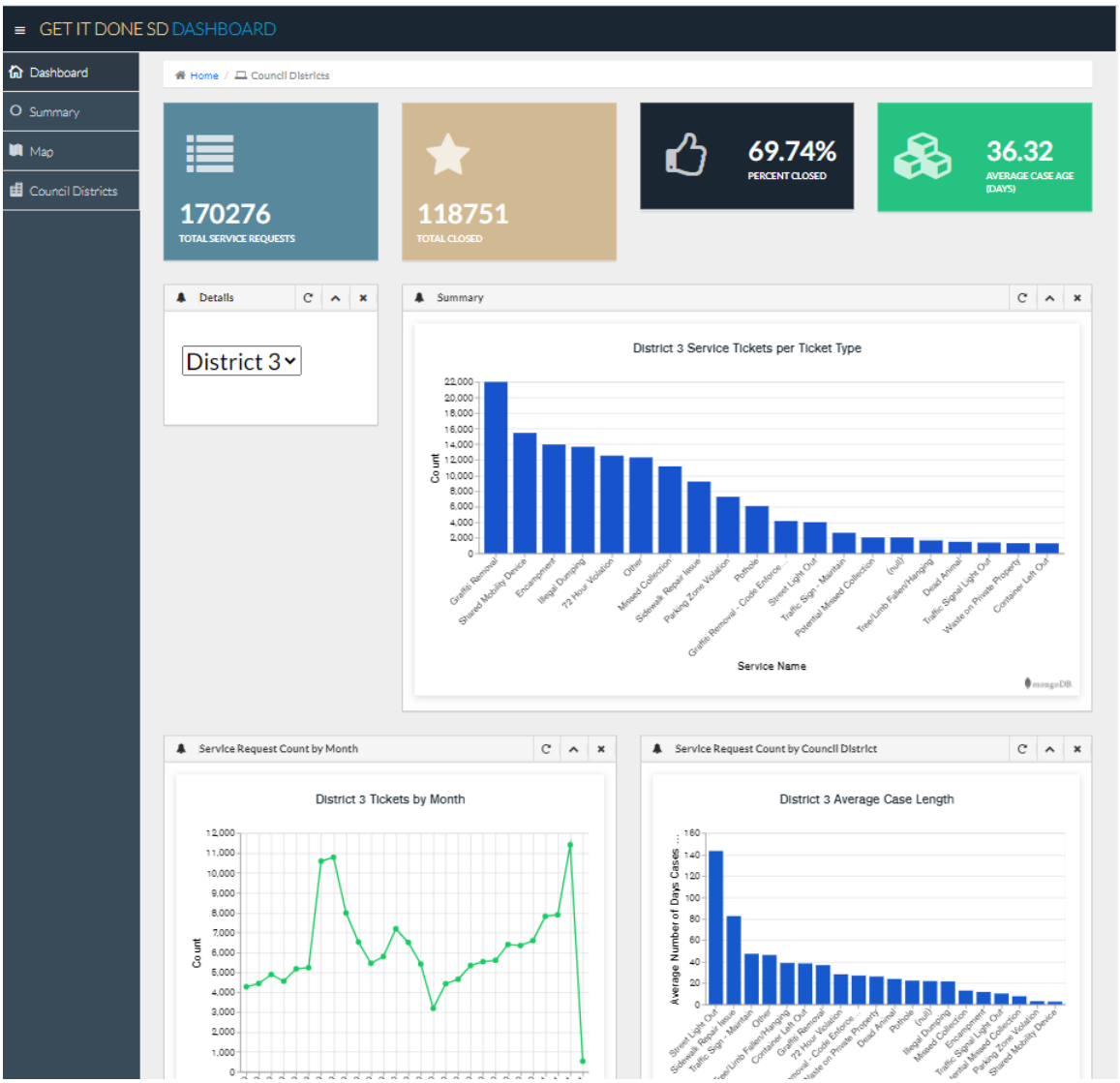
FEATURES

- ✓ Year selector
- ✓ Summary Status at a glance
- ✓ Service requests by service type
- ✓ Leaflet mapbox with popup tool tip and filter (service type, limit)
- ✓ Service request count by month
- ✓ Top 20 service requests by type
- ✓ Service requests by council district
- ✓ Average case length by council district
- ✓ COVID-19 heatmap

MAIN PAGE

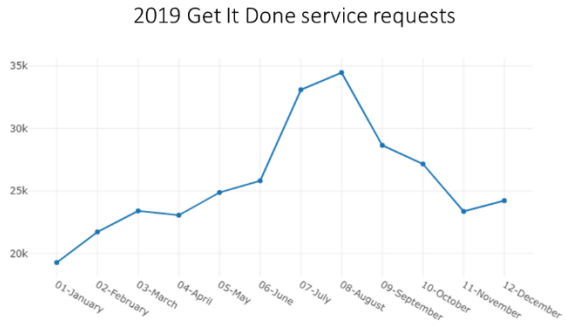

GET IT DONE SD DASHBOARD
Dashboard
Summary
Map
Council Districts
Home / Dashboard
286092
TOTAL SERVICE REQUESTS
235298
TOTAL CLOSED
82.25%
PERCENT CLOSED
28.12
AVERAGE CASE AGE (DAYS)
Data By Year
2020
Ticket Type Count by Category
2020 Ticket Type Count
Count
Service Name
Map
Service Request Count by Month
Service Request Count by Council District
2020 Number of Tickets per Council District
Number of Tickets
Council District
Covid-19 Complaints (All Time)
Covid-19 Complaints

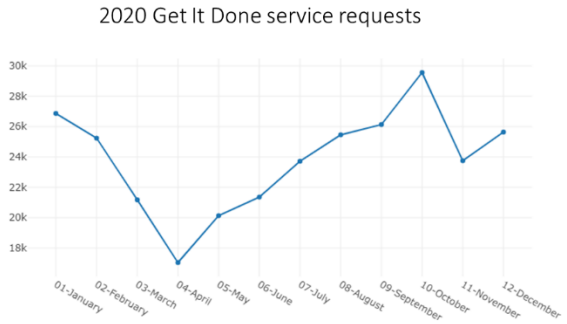
COUNCIL DISTRICT PAGE



CONCLUSION

NUMBER OF SERVICE REQUESTS BY YEAR – PLOTLY





In 2019, service requests peeked in July and August. District data should be verified to see if spikes could possibly occur in more popular tourist areas.

In 2020, service requests dipped significantly in April which correlates with the start of COVID-19 restrictions, and peeked in October.

COUNCIL DISTRICTS PERFORMANCE

Council Districts	2019	2020	2021 to date
Most service requests	District 3 (75K)	District 3 (66K)	District 3 (25K)
Fewest service requests	District 5 (10K)	District 5 (10K)	District 5 (3K)
Average case length across districts	46-60 days	20-35 days	10-16 days
Longest average case length	District 5 (70 days+)	District 5 (40 days+)	District 5 (21 days+)

The data shows that the Council District 5 has the fewest service requests and the longest average case length. Further review disclosed that District 5 has several long-duration service request types that possibly skew the average case length:

- Evaluate for resurfacing: 280 cases at 400 days to close
- Sidewalk repair issues: 750 cases at 200 days to close

It is recommended to remove the above two items from the data and reevaluate the average case length of District 5. If the data still shows a significant higher average case length, possible culprits such as high volume service requests should be reviewed to determine if there are training or resource issues.

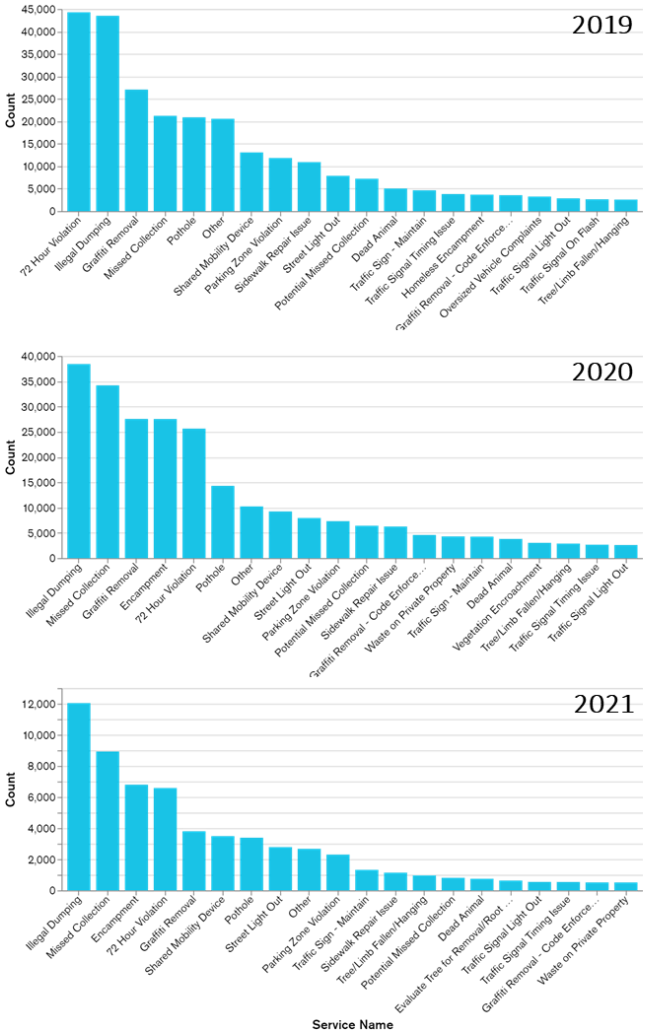
SERVICE TYPES

When looking at the annual summary statistics, the following is noted:

Service Type	2019	2020	2021	Trend
Illegal Dumping	44K	38K	12K	↑
Missed Collection	21K	34K	9K	↑
Encampment	4K	27K	7K	◄ ►

Since 2019, illegal dumping has been the most popular service request in the City of San Diego. Further analysis is required to see if locations of the occurrences are repetitive or clustered.

Missed collections is experiencing a significant increase since 2019 and is still trending upwards. Council district data should be reviewed to determine if this is a city-wide trend or if this is occurring in specific areas. Causes could be things such as an aging vehicle fleet or area population growth. Possible remedies could be additional training if the issue is area-specific, additional resources if the issue is city-wide.



GOING FORWARD

The dashboard could be expanded with the following data:

- Are dips/spikes occurring during certain times of the year? Certain districts?
    - Average days from open to close, grouped by category, by council district compared to overall
    - Is average time to resolve a service request same across categories? Across districts?
    - Dips/spikes? Certain areas? Certain times of year?
- Group the data by major categories (less then 20) instead of every category

CODE

FLASK SETUP

```
# Flask Setup
#################################################
app = Flask(__name__)
CORS(app)

#################################################
# Database Setup
#################################################
mongo = pymongo.MongoClient(config.mongo_conn, maxPoolSize=50, connect=False)
db = pymongo.database.Database(mongo, config.db_name)
col = pymongo.collection.Collection(db, 'sandiego')
collection_summary = pymongo.collection.Collection(db, 'summary_counts')


# create route that renders index.html template
@app.route("/")
def home():
    return render_template("index.html")

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

@app.route("/api/data")
def data():
    results = json.loads(dumps(col.find().limit(500).sort("time", -1)))
    return jsonify(results)

@app.route("/api/daterequested/<year>!<name>!<limit>")
def daterequested(year, name, limit):
    year_int = 2021
    limit_int = 1000
    try:
        year_int = int(year)
        limit_int = int(limit)
    except ValueError:
        # Handle the exception
        "Invalid Year"


    local = pytz.timezone("America/Los_Angeles")
    dt_start = datetime.strptime(str(year_int) + "-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")
    dt_start_local = local.localize(dt_start, is_dst=None)
    dt_start_utc = dt_start_local.astimezone(pytz.utc)

    dt_end = datetime.strptime(str(year_int + 1) + "-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")
    dt_end_local = local.localize(dt_end, is_dst=None)
```

```
    dt_end_utc = dt_end_local.astimezone(pytz.utc)

    print(dt_start_utc, dt_end_utc)
    if name != "All":
        filter={
            'date_requested': {
                '$gte': dt_start_utc,
                '$lt': dt_end_utc
            },
            'service_name':name
        }
    else:
        filter={
            'date_requested': {
                '$gte': dt_start_utc,
                '$lt': dt_end_utc
            }
        }
    results = json.loads(dumps(col.find(filter=filter).limit(limit_int).sort("time", -
1)))

    return jsonify(results)

@app.route("/api/summary/<year>")
def summary(year):
    filter= {"year":int(year)}

    results = json.loads(dumps(collection_summary.find(filter=filter)))

    return jsonify(results)

@app.route("/api/cdSummary/<district>")
def cdSummary(district):
    filter= {"district":int(district)}

    results = json.loads(dumps(collection_summary.find(filter=filter)))

    return jsonify(results)

@app.route("/api/servicenames")
def servicenames():

    results = json.loads(dumps(col.distinct("service_name")))

    return jsonify(results)




if __name__ == '__main__':
    app.run(debug=True, port=5104)
```

MAIN_MBM.JPYNB TO PROCESS THE DATA

In [ ]:
```
# Initialize PyMongo to work with MongoDBs
client = pymongo.MongoClient(config.mongo_conn)
```

In [ ]:
```
# Define database and collection
db = client.getitdone_db
SR_collection = db.sandiego
summary_collection = db.summary_counts
```

In [ ]:
```
for SR in SR_collection.find():
    print(SR['service_request_id'])
    break
```

In [ ]:
```
summary_collection.count_documents({'year':2021})
```

In [ ]:
```
summary_collection.count_documents({'year':2021})
```

In [ ]:
```
for summary_obj in summary_collection.find():
    print(summary_obj)
    break
```

In [ ]:
```
currentMonth = datetime.now().month
print(currentMonth)
type(currentMonth)
```

In [ ]:
```
local = pytz.timezone("America/Los_Angeles")
dt_start = datetime.strptime("2021-" + str(currentMonth) + "-1 00:00:00", "%Y-%m-%d %H:%M:%S")
dt_start_local = local.localize(dt_start, is_dst=None)
dt_start_utc = dt_start_local.astimezone(pytz.utc)

dt_end = datetime.strptime("2021-" + str(currentMonth + 1) + "-1 00:00:00", "%Y-%m-%d %H:%M:%S")
dt_end_local = local.localize(dt_end, is_dst=None)
dt_end_utc = dt_end_local.astimezone(pytz.utc)


print(dt_start_utc, dt_end_utc)
```

In [ ]:
```
year = 2020
local = pytz.timezone("America/Los_Angeles")
dt_start = datetime.strptime(str(year) + "-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")
dt_start_local = local.localize(dt_start, is_dst=None)
dt_start_utc = dt_start_local.astimezone(pytz.utc)

dt_end = datetime.strptime(str(year + 1) +"-1-1 00:00:00", "%Y-%m-%d %H:%M:%S")
dt_end_local = local.localize(dt_end, is_dst=None)
dt_end_utc = dt_end_local.astimezone(pytz.utc)

filter_requested ={
    'date_requested': {
        '$gte': dt_start_utc,
        '$lt': dt_end_utc
    }
}
filter_closed ={
    'date_requested': {
        '$gte': dt_start_utc,
        '$lt': dt_end_utc
    }, 'status':"Closed"
}


total_requested = SR_collection.count_documents(filter=filter_requested)

sum_case_age_days = 0
for SR in SR_collection.find(filter=filter_requested):
    if 'case_age_days' in SR:
        sum_case_age_days += int(SR["case_age_days"])

total_closed = SR_collection.count_documents(filter=filter_closed)
```

In [ ]:
```
print(dt_start_utc, dt_end_utc)
print(total_requested, total_closed)
print(sum_case_age_days)
```

In [ ]:
```
            update_doc = summary_collection.find_one_and_update(
                {'year' : year},
                {'$set':
                    {
                        "total_requested":int(total_requested),
                        "total_closed":int(total_closed),
                        "percent_closed": float(total_closed/total_requested),
                        "average_case_age_days": float(sum_case_age_days/total_requested)

                    }

                },upsert=True
            )
```

In [ ]:
```
def updateDistricts(district):
    filter_requested ={'council_district': district}
    filter_closed ={'council_district': district, 'status':"Closed"}

    total_requested = SR_collection.count_documents(filter=filter_requested)

    sum_case_age_days = 0
    for SR in SR_collection.find(filter=filter_requested):
        if 'case_age_days' in SR:
            sum_case_age_days += int(SR["case_age_days"])

    total_closed = SR_collection.count_documents(filter=filter_closed)

    update_doc = summary_collection.find_one_and_update(
                {'district' : district},
                {'$set':
                    {
                        "total_requested":int(total_requested),
                        "total_closed":int(total_closed),
                        "percent_closed": float(total_closed/total_requested),
                        "average_case_age_days": float(sum_case_age_days/total_requested)

                    }

                },upsert=True
            )

districts = [1,2,3,4,5,6,7,8,9]
for district in districts:
    updateDistricts(district)
```

In [ ]:
```
filter={
    'date_requested': {
        '$gte': dt_start_utc,
        '$lt': dt_end_utc
    }
}

result = client['getitdone_db']['sandiego'].find(filter=filter)
```

In [ ]:
```
i = 0
date_requested_count = {}
date_closed_count = {}

for SR in result:

    if 'date_requested' in SR:
        date_requested_month = SR['date_requested'].month
        if (date_requested_month in date_requested_count):
            date_requested_count[date_requested_month] += 1
        else:
            date_requested_count[date_requested_month] = 1
    if 'date_closed' in SR:
        date_closed_month = SR['date_closed'].month
        if (date_closed_month in date_closed_count):
            date_closed_count[date_closed_month] += 1
        else:
            date_closed_count[date_closed_month] = 1

    # SR['case_age_days']
    # SR['service_name']
    # SR['case_record_type']
    # SR['status']
    # SR['lat']
    # SR['lng']
    # SR['street_address']
    # SR['council_district']
    # SR['comm_plan_code']
    # SR['comm_plan_name']
    # SR['case_origin']
    # SR['public_description']

    # i += 1
    # if i > 5:
    #     break
```

In [ ]:
```
date_requested_count
```

In [ ]:
```
date_closed_count
```

In [ ]:
```
def get_sd_api_data(sd_api_url):
    print("import SR from get it done api")
    response = requests.get(sd_api_url)
    response_json = response.json()

    count_insert_SR = 0
    count_update_SR = 0

    for SR in response_json:
        public_description = ""
        media_url = ""

        # Get the data from the results
        service_request_id = int(SR["service_request_id"])
        date_requested_string = SR["requested_datetime"] # or any date sting of differing formats.
        date_requested = parser.parse(date_requested_string)
        updated_datetime_string = SR["updated_datetime"] # or any date sting of differing formats.
        updated_datetime = parser.parse(updated_datetime_string)

        if "description" in SR:
            public_description = SR["description"]
        if 'media_url' in SR:
            media_url = SR["media_url"]

        if SR_collection.count_documents({'service_request_id': service_request_id}) == 0:
            print("Insert new service request! ")
            insert_doc = {
                "service_request_id": service_request_id,
                "date_requested": date_requested,
                "date_updated": updated_datetime,
                "status": SR["status"],
                "service_code": SR["service_code"],
                "service_name": SR["service_name"],
                "public_description": public_description,
                "street_address": SR["address"],
                "lat": float(SR["lat"]),
                "lng": float(SR["long"]),
                "media_url": media_url
                }
            doc = SR_collection.insert_one(insert_doc)
            count_insert_SR += 1
            # if config.debug:
            #     print(doc)

        else:
            print(f"SR {service_request_id} already exists, update existing service request!")
            update_doc = SR_collection.find_one_and_update(
                {'service_request_id' : service_request_id},
                {'$set':
                    {
                        "date_requested": date_requested,
                        "date_updated": updated_datetime,
                        "status": SR["status"],
                        "service_code": SR["service_code"],
                        "service_name": SR["service_name"],
                        "public_description": public_description,
                        "street_address": SR["address"],
                        "lat": float(SR["lat"]),
                        "lng": float(SR["long"]),
                        "media_url": media_url
                    }

                },upsert=True
            )
            count_update_SR += 1
            # if config.debug:
            #     print(update_doc)

    return f"{str(count_insert_SR)} SR created. {str(count_update_SR)} SR updated."
```

In [ ]:
```
summary = get_sd_api_data("http://san-diego.spotreporters.com/open311/v2/requests.json")
```

VISUALIZATION.PY FOR THE SUMMARY STATISTICS

```
In [ ]: from matplotlib import pyplot as plt
        import numpy as np
        import scipy.stats as stats
        import pandas as pd

        # Project defined modules
        import config
        import visualization as viz
```

```
In [ ]: # Create visualization object
        visuals = viz.Visualizer()
        base_df = visuals.get_clean_data()
        base_df.head(5)
```

```
In [ ]: grouped_service_name_df = base_df.groupby(["service_name"])

        grouped_service_name_df.head()
```

```
In [ ]: SR_count = grouped_service_name_df["service_name"].count()
```

```
In [ ]: # Assemble the resulting series into a single summary dataframe.

        summary_stats_df = pd.DataFrame({
                "Service Name": grouped_service_name_df["service_name"].unique(),
                "Count": SR_count
            })
        summary_stats_df
```

```
In [ ]:  summary_stats_sorted_df = summary_stats_df.sort_values(['Count'], ascending=False)
```

```
In [ ]: summary_stats_sorted_df.head()
```

```
In [ ]:
```

APP.PY TO POPULATE MAIN DASHBOARD

```
d3.selectAll("body").on("change", populateDashboard);

function populateServicesNames() {
  url_servicenames = "api/servicenames";
  d3.json(url_servicenames).then(function(response) {
    console.log(response)
    var serviceNameArr = response
    // select inputs
    var inputElementDate = d3.select("#selServiceName");

    // auto populate available filter days and add blank option to search without date filter
    serviceNameArr.forEach(servicename => {
         inputElementDate.append('option').text(servicename);
    });
  });




}

function populateDashboard() {
  console.log("loading summary data...")
```

```
  // Use D3 to select the dropdown menu
  var CB_Year = d3.select("#selYear");
  // Assign the value of the dropdown menu option to a variable
  var year = CB_Year.node().value;

  console.log(year);

  /* data route */
  const url = "api/summary" + "/" + year;
  d3.json(url).then(function(response) {
    ////////////////////////////////////////////////
    // Multiline Plot SR over time
    ////////////////////////////////////////////////
    console.log(year);
    const data = response[0].summary;
    //console.log(data);

    month = []
    count = []
    for (const [key, value] of Object.entries(data)) {
      // console.log(`${key}: ${value}`);
      month.push(key);
      count.push(value);
    }

    countbymonth = [{
      x: month,
      y: count }];

    var lineplot = d3.selectAll("#line-plot").node();

    Plotly.newPlot(lineplot, countbymonth);
    console.log("Summary data loaded.")


    // bar chart count by service request type
    var chart_url = response[0].chart_url;
    chart_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-
radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url + "'>
</iframe>"
    d3.select("#bar-plot").html(chart_html)

    // bar chart count by council district
    var chart_url_council_dist = response[0].chart_url_council_dist
    chart_council_dist_html = "<iframe id='bar-
count' style='background: #FFFFFF;border: none;border-radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url_counc
il_dist + "'></iframe>"
    d3.select("#bar-plot-council-dist").html(chart_council_dist_html)

    var sr_name = "All"
    var limit = 1000

    // Use D3 to select the dropdown menu
```

```
    var CB_SRName = d3.select("#selServiceName");
    // Assign the value of the dropdown menu option to a variable
    sr_name = CB_SRName.node().value;
    // Use D3 to select the dropdown menu
    var CB_Limit = d3.select("#selMapLimit");
    // Assign the value of the dropdown menu option to a variable
    limit = CB_Limit.node().value;
    console.log(year, sr_name, limit);



    map_html = "<iframe src='http://127.0.0.1:5500/pages/index.html?year=" + year + "&name=" +
 sr_name + "&limit=" + limit + "' height='600px' width='100%' title='Service Request Cluster M
ap'></iframe>";
    d3.select("#map").html(map_html)

    d3.select("#total_requested").html("<span>" + response[0].total_requested + "</span>")
    d3.select("#total_closed").html("<span>" + response[0].total_closed + "</span>")
    d3.select("#percent_closed").html("<span>" + (parseFloat(response[0].percent_closed)*100).
toFixed(2).toString() + "%</span>")
    d3.select("#average_case_age").html("<span>" + parseFloat(response[0].average_case_age_day
s).toFixed(2).toString() + "</span>")

  });
}

function populateDistricts() {
  console.log("loading summary data...")

  // Use D3 to select the dropdown menu
  var CB_District = d3.select("#selDistrict");
  // Assign the value of the dropdown menu option to a variable
  var district = CB_District.node().value;

  console.log(district);

  /* data route */
  const cdurl = "api/summary" + "/" + district;
  d3.json(cdurl).then(function(response) {

    console.log(district);
    var line_url = response[0].chart_url_cd_tickets_over_time;
    line_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-
radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + line_url + "'><
/iframe>"
    d3.select("#cd-line-plot").html(line_html)


    // bar chart count by service request type
    var chart_url = response[0].chart_url;
    chart_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-
radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url + "'>
</iframe>"
```

```
    d3.select("#cd-bar-plot").html(chart_html)

    // bar chart count by council district
    var chart_url_council_dist = response[0].chart_url_council_dist
    chart_council_dist_html = "<iframe id='bar-
count' style='background: #FFFFFF;border: none;border-radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url_counc
il_dist + "'></iframe>"
    d3.select("#bar-plot-council-dist").html(chart_council_dist_html)

    d3.select("#total_requested").html("<span>" + response[0].total_requested + "</span>")
    d3.select("#total_closed").html("<span>" + response[0].total_closed + "</span>")
    d3.select("#percent_closed").html("<span>" + (parseFloat(response[0].percent_closed)*100).
toFixed(2).toString() + "%</span>")
    d3.select("#average_case_age").html("<span>" + parseFloat(response[0].average_case_age_day
s).toFixed(2).toString() + "</span>")

  });
}

function init() {
  populateServicesNames();
  populateDashboard();
  populateDistricts();
};

init();
```

CDAPP.JS TO POPULATE COUNCIL DISTRICTS DASHBOARD

```
d3.selectAll("body").on("change", populateDistricts);


function populateDistricts() {
  console.log("loading summary data...")

  // Use D3 to select the dropdown menu
  var CB_District = d3.select("#selDistrict");
  // Assign the value of the dropdown menu option to a variable
  var district = CB_District.node().value;

  console.log(district);

  /* data route */
  const cdurl = "api/cdSummary" + "/" + district;
  d3.json(cdurl).then(function(response) {

    console.log(district);
    var line_url = response[0].chart_url_cd_tickets_over_time;
    line_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-
radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + line_url + "'><
/iframe>"
```

```
    d3.select("#cd-line-plot").html(line_html)


    // bar chart count by service request type
    var chart_url = response[0].chart_url;
    chart_html = "<iframe id='bar-count' style='background: #FFFFFF;border: none;border-
radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url + "'>
</iframe>"
    d3.select("#cd-bar-plot").html(chart_html)

    // bar chart count by council district
    var chart_url_council_dist = response[0].chart_url_cd_average_case_length;
    chart_council_dist_html = "<iframe id='bar-
count' style='background: #FFFFFF;border: none;border-radius: 2px;box-
shadow: 0 2px 10px 0 rgba(70, 76, 79, .2);' width='100%' height='480' src='" + chart_url_counc
il_dist + "'></iframe>"
    d3.select("#bar-plot-council-dist").html(chart_council_dist_html)

    d3.select("#total_requested").html("<span>" + response[0].total_requested + "</span>")
    d3.select("#total_closed").html("<span>" + response[0].total_closed + "</span>")
    d3.select("#percent_closed").html("<span>" + (parseFloat(response[0].percent_closed)*100).
toFixed(2).toString() + "%</span>")
    d3.select("#average_case_age").html("<span>" + parseFloat(response[0].average_case_age_day
s).toFixed(2).toString() + "</span>")

  });
}

function init() {
  populateDdistricts();
};

init();
```

LOGIC.JS TO CREATE MAP

```
// Creating map object
var myMap = L.map("map", {
  center: [32.7157, -117.1611],
  zoom: 11
});

// Adding tile layer to the map
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://ww
w.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);

const urlParams = new URLSearchParams(window.location.search);

// Store API query variables
var baseURL = "http://127.0.0.1:5104/api/daterequested/";
var year = urlParams.get('year');
var sr_name = "!" + urlParams.get('name');
var limit = "!" + urlParams.get('limit');

console.log(year);

// Assemble API query URL
var url = baseURL + year + sr_name + limit
console.log(url);
// Grab the data with d3
d3.json(url, function(response) {

  // Create a new marker cluster group
  var markers = L.markerClusterGroup();


  // Loop through data
  for (var i = 0; i < response.length; i++) {
      lat = response[i]["lat"]
      lng = response[i]["lng"]

      var details = "service_request_id: " + response[i]["service_request_id"];

      if (response[i]["date_requested"]) {
        details += "<br>date_requested: " + response[i]["date_requested"]
        console.log(response[i]["date_requested"]);
      } if (response[i]["case_age_days"]) {
        details += "<br>case_age_days: " + response[i]["case_age_days"]
      } if (response[i]["service_name"]) {
        details += "<br>service_name: " + response[i]["service_name"]
      } if (response[i]["case_record_type"]) {
        details += "<br>case_record_type: " + response[i]["case_record_type"]
      } if (response[i]["date_closed"]) {
        details += "<br>date_closed: " + response[i]["date_closed"]
        console.log(response[i]["date_closed"]);
      } if (response[i]["status"]) {
        details += "<br>status: " + response[i]["status"]
      } if (response[i]["street_address"]) {
        details += "<br>street_address: " + response[i]["street_address"]
      } if (response[i]["council_district"]) {
        details += "<br>council_district: " + response[i]["council_district"]
      } if (response[i]["comm_plan_code"]) {
        details += "<br>comm_plan_code: " + response[i]["comm_plan_code"]
      } if (response[i]["comm_plan_name"]) {
        details += "<br>comm_plan_name: " + response[i]["comm_plan_name"]
      } if (response[i]["case_origin"]) {
        details += "<br>case_origin: " + response[i]["case_origin"]
      } if (response[i]["public_description"]) {
        details += "<br>public_description: " + response[i]["public_description"]
```

```
      } if (response[i]["media_url"]) {
        details += "<img src='" + response[i]["media_url"] + "' style='height:200px;float:righ
t'/>";
      }

     // Check for location property
     if ((lat) && (lng)) {
      // Add a new marker to the cluster group and bind a pop-up
      markers.addLayer(L.marker([lat, lng])
        .bindPopup(details));
    }

   }

  // // Add our marker cluster layer to the map
  myMap.addLayer(markers);

});
```

CHARTS

Charts were created in MondoDB charts. The links were stored in a MongoDB collection and pulled into the dashboard from there.