# CS 340 README Template

## About the Project/Project Title

## *This README document provides documentation for the CS-340 Dashboard project developed for Grazioso Salvare. The project aims to create an interactive dashboard using the Dash framework in Python. The dashboard connects to a MongoDB database containing Austin Animal Center Outcomes data, allowing users to filter and visualize the data dynamically.*

## Motivation

The motivation behind this project is to provide Grazioso Salvare with an interactive dashboard that enables efficient exploration and visualization of the Austin Animal Center Outcomes data. The dashboard allows users to filter data based on rescue types and view corresponding charts, enhancing their ability to make data-driven decisions.

**Required Functionality**

The required functionality of the project includes the following features:

1. An interactive data table displaying the unfiltered view of the Austin Animal Center Outcomes dataset.
2. Filter options for Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking, and a Reset option.
3. An interactive data table that responds to filter selections.
4. Charts (a pie chart and a geolocation chart) that dynamically update based on filter selections.
5. Dashboard branding with the Grazioso Salvare logo, a unique identifier, and required widgets.

## Getting Started

* **Dash Framework:** Dash was chosen due to its simplicity, ease of integration with Plotly for charts, and its ability to create interactive web applications with Python.
* **MongoDB:** MongoDB was selected as the database for its flexibility with JSON-like documents, scalability, and compatibility with Python through the PyMongo library.

MongoDB was chosen as the model component for its key features:

* Flexible Schema: MongoDB's schema-less nature allows easy handling of diverse data types in the Austin Animal Center Outcomes dataset.
* JSON-like Documents: The data stored in MongoDB is represented in a format (BSON) similar to JSON, making it easy to work with in Python.
* Scalability: MongoDB is highly scalable, allowing for the efficient storage and retrieval of large datasets.

Dash provides a Pythonic way to build interactive web applications, allowing for the creation of complex data visualizations and interactive user interfaces. It uses a reactive programming paradigm to update the interface dynamically based on user interactions.

## Installation

*To use this module, ensure the following tools are installed.*

* *Python 3.x*
* *PyMongo libray -pip install pymongo*
* *Jupyter*

## Usage

### Code Example

# Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

# Configure the necessary Python module imports for dashboard components

import dash\_leaflet as dl

from dash import dcc

from dash import html

import plotly.express as px

from dash import dash\_table

from dash.dependencies import Input, Output, State

import base64

# Configure OS routines

import os

# Configure the plotting routines

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from dash.dcc import RadioItems

#### FIX ME #####

# change animal\_shelter and AnimalShelter to match your CRUD Python module file name and class name

from animal\_shelter import AnimalShelter

###########################

# Data Manipulation / Model

###########################

# FIX ME update with your username and password and CRUD Python module name

username = "aacuser"

password = "Oakmont"

# Connect to database via CRUD Module

db = AnimalShelter(username, password)

# class read method must support return of list object and accept projection json input

# sending the read method an empty document requests all documents be returned

df = pd.DataFrame.from\_records(db.read({}))

# MongoDB v5+ is going to return the '\_id' column and that is going to have an

# invlaid object type of 'ObjectID' - which will cause the data\_table to crash - so we remove

# it in the dataframe here. The df.drop command allows us to drop the column. If we do not set

# inplace=True - it will reeturn a new dataframe that does not contain the dropped column(s)

df.drop(columns=['\_id'],inplace=True)

## Debug

# print(len(df.to\_dict(orient='records')))

# print(df.columns)

#########################

# Dashboard Layout / View

#########################

app = JupyterDash(\_\_name\_\_)

#FIX ME Add in Grazioso Salvare’s logo

image\_filename = 'Grazioso\_Salvare\_Logo.png' # replace with your own image

encoded\_image = base64.b64encode(open(image\_filename, 'rb').read())

#FIX ME Place the HTML image tag in the line below into the app.layout code according to your design

#FIX ME Also remember to include a unique identifier such as your name or date

#html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()))

app.layout = html.Div([

html.Div(id='hidden-div', style={'display':'none'}),

html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()), style={'height': '10%', 'width': '10%'}),

html.P('Created by: Kendal Guizado'),

html.Center(html.B(html.H1('SNHU CS-340 Dashboard'))),

html.Hr(),

html.Div([ #FIXME Add in code for the interactive filtering options. For example, Radio buttons, drop down, checkboxes, etc.

html.Label("Select Rescue Type:"),

dcc.RadioItems(

id='filter-type',

options=[

{'label': 'Water Rescue', 'value': 'Water'},

{'label': 'Mountain or Wilderness Rescue', 'value': 'Mountain'},

{'label': 'Disaster or Individual Tracking', 'value': 'Disaster'},

{'label': 'Reset', 'value': 'Reset'}

],

value='Reset', # Default value

labelStyle={'display': 'block'}

),

]),

html.Hr(),

dash\_table.DataTable(

id='datatable-id',

columns=[

{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns

],

data=df.to\_dict('records'),

#FIXME: Set up the features for your interactive data table to make it user-friendly for your client

page\_size =10,

sort\_action="native",

row\_selectable='single',

selected\_rows=[0],

),

html.Br(),

html.Hr(),

html.Div(className='row',

style={'display' : 'flex'},

children=[

html.Div(

id='graph-id',

className='col s12 m6',

),

html.Div(

id='map-id',

className='col s12 m6',

)

])

])

#############################################

# Interaction Between Components / Controller

#############################################

@app.callback(Output('datatable-id','data'),

[Input('filter-type', 'value')])

def update\_dashboard(filter\_type):

if filter\_type == 'Reset':

query = {}

else:

# Add your filter conditions based on the specifications provided

if filter\_type == 'Water':

query = {

'breed': {'$in': ['Labrador Retriever Mix', 'Chesapeake Bay Retriever', 'NewFoundland']},

'sex\_upon\_outcome': 'Intact Female',

'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}

}

elif filter\_type == 'Mountain':

query = {

'breed': {'$in': ['German Shepherd', 'Alaskan Malamute', 'Old English Sheepdog', 'Siberian Husky', 'Rottweiler']},

'sex\_upon\_outcome': 'Intact Male',

'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}

}

elif filter\_type == 'Disaster':

query = {

'breed': {'$in': ['Doberman Pinscher', 'German Shepherd', 'Golden Retriever', 'Bloodhound', 'Rottweiler']},

'sex\_upon\_outcome': 'Intact Male',

'age\_upon\_outcome\_in\_weeks': {'$gte': 20, '$lte': 300}

}

else:

query = {}

data = db.read(query)

df\_filtered = pd.DataFrame.from\_records(data)

df\_filtered.drop(columns=['\_id'], inplace=True)

return df\_filtered.to\_dict('records')

@app.callback(

Output('graph-id', "children"),

[Input('datatable-id', "data")])

def update\_graphs(viewData):

if viewData is None:

return []

#Convert the dictionary to data to a DataFrame

df = pd.DataFrame.from\_dict(viewData) #d code for chart of your choice (e.g. pie chart) #

return [

dcc.Graph(

figure = px.pie(df, names='breed', title='Preferred Animals')

)

]

#This callback will highlight a cell on the data table when the user selects it

@app.callback(

Output('datatable-id', 'style\_data\_conditional'),

[Input('datatable-id', 'selected\_columns')]

)

def update\_styles(selected\_columns):

return [{

'if': { 'column\_id': i },

'background\_color': '#D2F3FF'

} for i in selected\_columns]

# This callback will update the geo-location chart for the selected data entry

# derived\_virtual\_data will be the set of data available from the datatable in the form of

# a dictionary.

# derived\_virtual\_selected\_rows will be the selected row(s) in the table in the form of

# a list. For this application, we are only permitting single row selection so there is only

# one value in the list.

# The iloc method allows for a row, column notation to pull data from the datatable

@app.callback(Output('map-id', "children"),

[Input('datatable-id', "derived\_virtual\_data"),

Input('datatable-id', "derived\_virtual\_selected\_rows")])

def update\_map(viewData, index):

if viewData is None:

return

elif index is None:

return

dff = pd.DataFrame.from\_dict(viewData)

# Because we only allow single row selection, the list can be converted to a row index here

if index is None:

row = 0

else:

row = index[0]

# Austin TX is at [30.75,-97.48]

return [

dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75,-97.48], zoom=10, children=[

dl.TileLayer(id="base-layer-id"),

# Marker with tool tip and popup

# Column 13 and 14 define the grid-coordinates for the map

# Column 4 defines the breed for the animal

# Column 9 defines the name of the animal

dl.Marker(position=[dff.iloc[row,13],dff.iloc[row,14]], children=[

dl.Tooltip(dff.iloc[row,4]),

dl.Popup([

html.H1("Animal Name"),

html.P(dff.iloc[row,9])

])

])

])

]

app.run\_server(debug=True)

### Tests

No specific testing suite has been implemented for this project. However, the code has been thoroughly reviewed and tested during development to ensure functionality.

### Screenshots

1) Initial State

A screenshot of a computer

Description automatically generated

2) Water Rescue

A screenshot of a computer

Description automatically generated

3) Mountain Rescue

A screenshot of a computer

Description automatically generated

4) Disaster Rescue

A screenshot of a computer

Description automatically generated

5) Reset

A screenshot of a computer

Description automatically generated

## Contact

Your name: Kendal Guizado O’riley