****

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана»

(национальный исследовательский университет)

(МГТУ им. Н.Э. Баумана)

**Домашнее задание по курсу**

**"Технологии мультимедиа"**

**(ИУ5 / ГУИМЦ)**

**«Синтез 3D изображения цветного автомобиля**

**с использованием dash библиотеки python»**

**Москва, МГТУ - 2021 год**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Задание:** Синтез 3D изображения цветного автомобиля с использованием dash библиотеки python.

# Краткое описание платформы Dash:

Dash — это веб-фреймворк, построенный на основе Plotly.js, React и Flask, который можно использовать для создания аналитических веб-приложений. Приложения Dash состоят из двух частей. Первая часть — «layout» описывает то, как выглядит наше приложение. Вторая часть описывает интерактивность приложения, о ней мы поговорим в следующей статье.

Dash предоставляет Python классы для всех визуальных компонентов приложения. Разработчики предоставляют набор компонентов в так называемых dash\_core\_components и dash\_html\_components.

**Важно:**

В dash\_core\_components содержатся различные динамические формы такие как, например, выпадающие списки, графики и чек-боксы.

В dash\_html\_components содержатся html конструкции, которыми можно завернуть наши формы. Например Div блоки или теги заголовков H1, H2, и так далее. Разработчики предоставляют нам некую абстракцию от html с помощью словарей Python.

## О работе с Plotly:

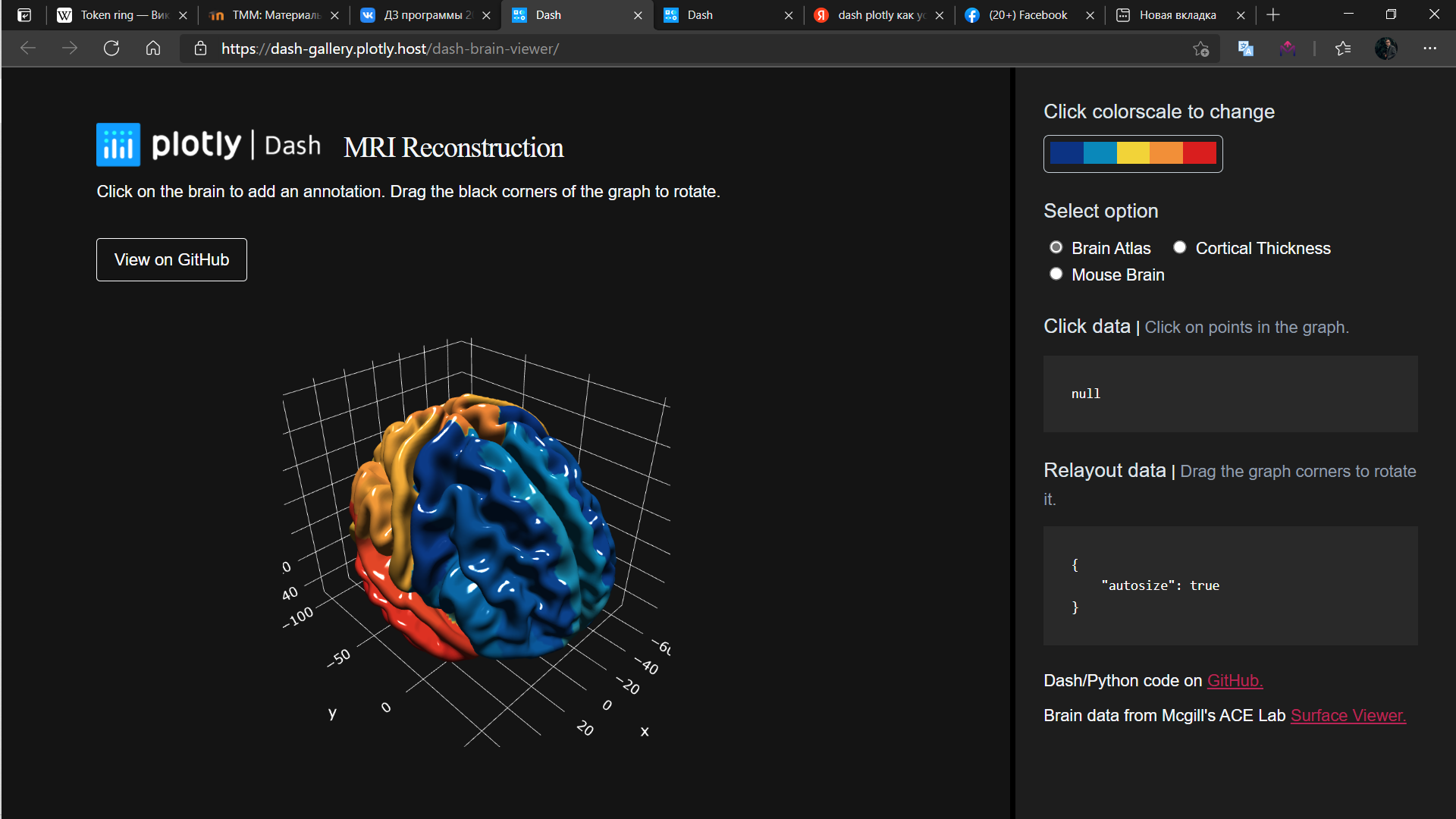
При помощи Dash можно быстро создать интерактивный многостраничный многопользовательский интерфейс полностью в Python, наполнить его красивыми визуализациями Plotly и, используя свои CSS-стили, сделать кастомный дизайн. Так как весь процесс создания происходит в Python, то веб-приложение можно с легкостью связать со всеми самыми продвинутыми библиотеками машинного обучения, нейронными сетями и анализом данных. Такой подход значительно ускоряет время создания веб-приложений без ущерба для дизайна — созданные аналитические интерфейсы и интерактивные отчеты.

Несмотря на новизну библиотеки, уже сейчас в ней присутствует достаточно широкий базовый набор веб-компонент. Также существует возможность создать свои собственные компоненты, правда, для этого уже понадобится владение Javascript, React и прочими веб-технологиями.

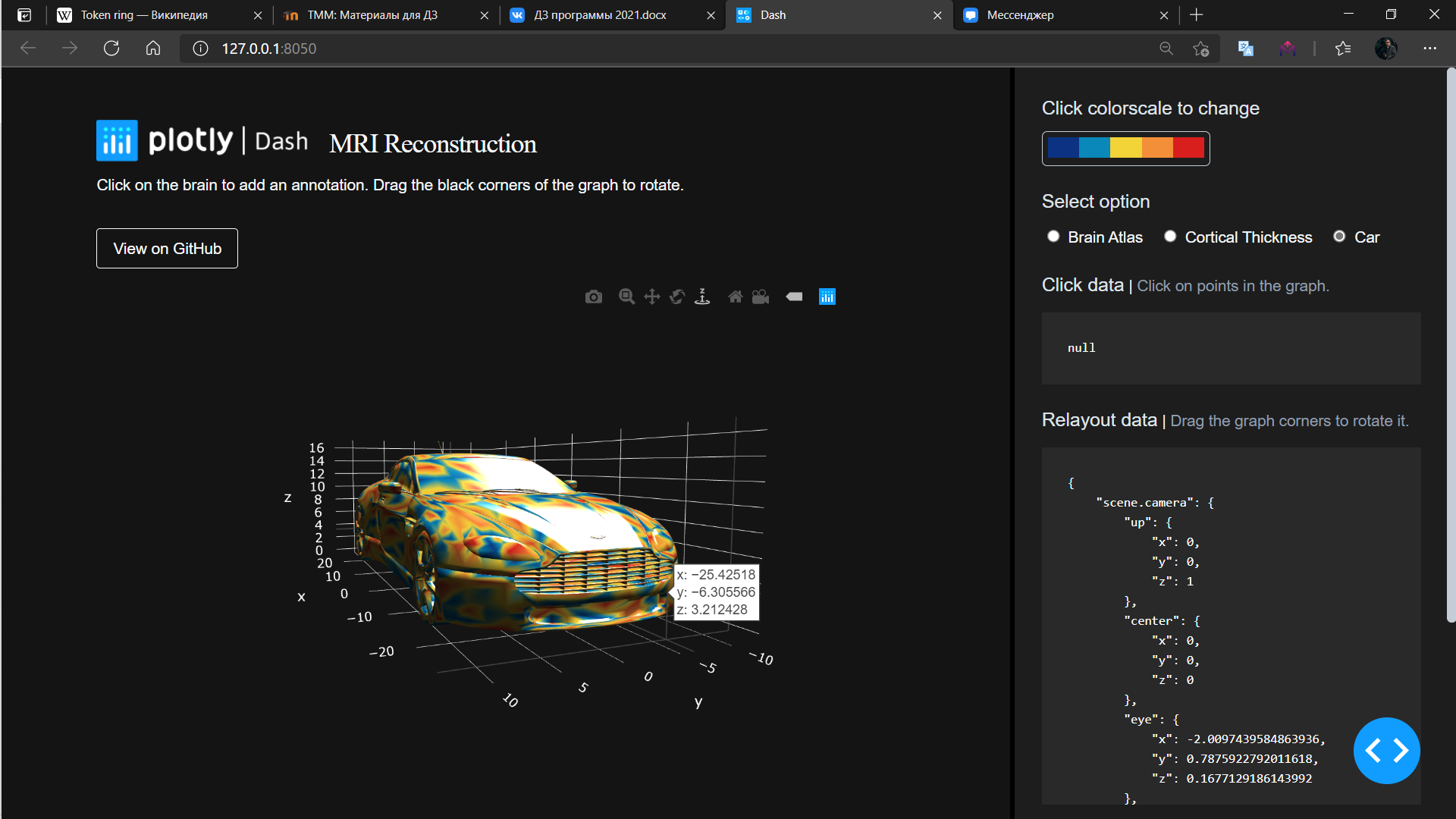
На данный момент можно с уверенностью утверждать, что Dash хорошо подходит для быстрого создания легковесных интерфейсов и дэшбордов. Так как Dash полностью инегрируется с Plotly, то любая аналитика и модели машинного обучения, реализованные в Python, могут быть отражены в веб-приложении с помощью красивых и интерактивных графиков, отчётов, моделей.

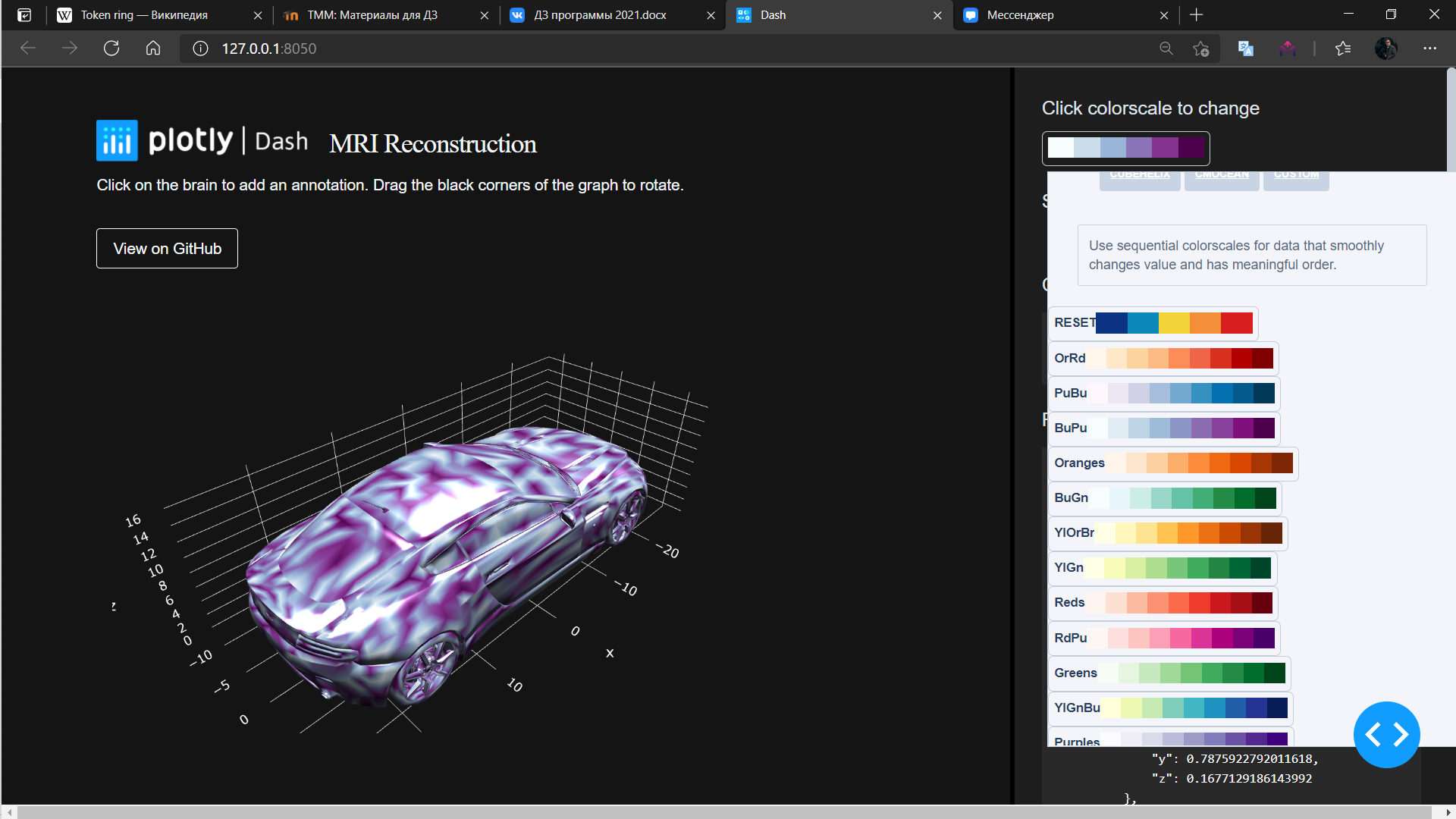
## Реализация задания

За основу был взят проект реализации модели мозга человека на платформе Plotly.



# Результаты работы программы





# Код программы

**App.py:**

import os  
import json  
  
import pip  
  
import dash  
import dash\_core\_components as dcc  
import dash\_html\_components as html  
import dash\_colorscales as dcs  
from dash.dependencies import Input, Output, State  
from dash.exceptions import PreventUpdate  
from mni import create\_mesh\_data, default\_colorscale  
  
  
app = dash.Dash(  
 \_\_name\_\_,  
 meta\_tags=[{"name": "viewport", "content": "width=device-width, initial-scale=1"}],  
)  
  
server = app.server  
  
GITHUB\_LINK = os.environ.get(  
 "GITHUB\_LINK",  
 "https://github.com/plotly/dash-sample-apps/tree/master/apps/dash-brain-viewer",  
)  
  
default\_colorscale\_index = [ea[1] for ea in default\_colorscale]  
  
axis\_template = {  
 "showbackground": True,  
 "backgroundcolor": "#141414",  
 "gridcolor": "rgb(255, 255, 255)",  
 "zerolinecolor": "rgb(255, 255, 255)",  
}  
  
plot\_layout = {  
 "title": "",  
 "margin": {"t": 0, "b": 0, "l": 0, "r": 0},  
 "font": {"size": 12, "color": "white"},  
 "showlegend": False,  
 "plot\_bgcolor": "#141414",  
 "paper\_bgcolor": "#141414",  
 "scene": {  
 "xaxis": axis\_template,  
 "yaxis": axis\_template,  
 "zaxis": axis\_template,  
 #"aspectratio": {"x": 1, "y": 0.5, "z": 0.4},  
 "camera": {"eye": {"x": 1.25, "y": 1.25, "z": 1.25}},  
 "annotations": [],  
 },  
}  
  
app.layout = html.Div(  
 [  
 html.Div(  
 [  
 html.Div(  
 [  
 html.Div(  
 [  
 html.Div(  
 [  
 html.Img(  
 src=app.get\_asset\_url("dash-logo.png")  
 ),  
 html.H4("MRI Reconstruction"),  
 ],  
 className="header\_\_title",  
 ),  
 html.Div(  
 [  
 html.P(  
 "Click on the brain to add an annotation. Drag the black corners of the graph to rotate."  
 )  
 ],  
 className="header\_\_info pb-20",  
 ),  
 html.Div(  
 [  
 html.A(  
 "View on GitHub",  
 href=GITHUB\_LINK,  
 target="\_blank",  
 )  
 ],  
 className="header\_\_button",  
 ),  
 ],  
 className="header pb-20",  
 ),  
 html.Div(  
 [  
 dcc.Graph(  
 id="brain-graph",  
 figure={  
 "data": create\_mesh\_data("car"),  
 "layout": plot\_layout,  
 },  
 config={"editable": True, "scrollZoom": False},  
 )  
 ],  
 className="graph\_\_container",  
 ),  
 ],  
 className="container",  
 )  
 ],  
 className="two-thirds column app\_\_left\_\_section",  
 ),  
 html.Div(  
 [  
 html.Div(  
 [  
 html.Div(  
 [  
 html.P(  
 "Click colorscale to change", className="subheader"  
 ),  
 dcs.DashColorscales(  
 id="colorscale-picker",  
 colorscale=default\_colorscale\_index,  
 ),  
 ]  
 )  
 ],  
 className="colorscale pb-20",  
 ),  
 html.Div(  
 [  
 html.P("Select option", className="subheader"),  
 dcc.RadioItems(  
 options=[  
 {"label": "Brain Atlas", "value": "human\_atlas"},  
 {"label": "Cortical Thickness", "value": "human"},  
 {"label": "Car", "value": "car"},  
 ],  
 value="car",  
 id="radio-options",  
 labelClassName="label\_\_option",  
 inputClassName="input\_\_option",  
 ),  
 ],  
 className="pb-20",  
 ),  
 html.Div(  
 [  
 html.Span("Click data", className="subheader"),  
 html.Span(" | "),  
 html.Span(  
 "Click on points in the graph.", className="small-text"  
 ),  
 dcc.Loading(  
 html.Pre(id="click-data", className="info\_\_container"),  
 type="dot",  
 ),  
 ],  
 className="pb-20",  
 ),  
 html.Div(  
 [  
 html.Span("Relayout data", className="subheader"),  
 html.Span(" | "),  
 html.Span(  
 "Drag the graph corners to rotate it.",  
 className="small-text",  
 ),  
 dcc.Loading(  
 html.Pre(id="relayout-data", className="info\_\_container"),  
 type="dot",  
 ),  
 ],  
 className="pb-20",  
 ),  
 html.Div(  
 [  
 html.P(  
 [  
 "Dash/Python code on ",  
 html.A(  
 children="GitHub.",  
 target="\_blank",  
 href=GITHUB\_LINK,  
 className="red-ish",  
 ),  
 ]  
 ),  
 html.P(  
 [  
 "Brain data from Mcgill's ACE Lab ",  
 html.A(  
 children="Surface Viewer.",  
 target="\_blank",  
 href="https://brainbrowser.cbrain.mcgill.ca/surface-viewer#ct",  
 className="red-ish",  
 ),  
 ]  
 ),  
 ]  
 ),  
 ],  
 className="one-third column app\_\_right\_\_section",  
 ),  
 dcc.Store(id="annotation\_storage"),  
 ]  
)  
  
  
def add\_marker(x, y, z):  
 *""" Create a plotly marker dict. """* return {  
 "x": [x],  
 "y": [y],  
 "z": [z],  
 "mode": "markers",  
 "marker": {"size": 25, "line": {"width": 3}},  
 "name": "Marker",  
 "type": "scatter3d",  
 "text": ["Click point to remove annotation"],  
 }  
  
  
def add\_annotation(x, y, z):  
 *""" Create plotly annotation dict. """* return {  
 "x": x,  
 "y": y,  
 "z": z,  
 "font": {"color": "black"},  
 "bgcolor": "white",  
 "borderpad": 5,  
 "bordercolor": "black",  
 "borderwidth": 1,  
 "captureevents": True,  
 "ay": -100,  
 "arrowcolor": "white",  
 "arrowwidth": 2,  
 "arrowhead": 0,  
 "text": "Click here to annotate<br>(Click point to remove)",  
 }  
  
  
def marker\_in\_points(points, marker):  
 *"""  
 Checks if the marker is in the list of points.  
   
 :params points: a list of dict that contains x, y, z  
 :params marker: a dict that contains x, y, z* ***:returns****: index of the matching marker in list  
 """* for index, point in enumerate(points):  
 if (  
 point["x"] == marker["x"]  
 and point["y"] == marker["y"]  
 and point["z"] == marker["z"]  
 ):  
 return index  
 return None  
  
  
@app.callback(  
 Output("brain-graph", "figure"),  
 [  
 Input("brain-graph", "clickData"),  
 Input("radio-options", "value"),  
 Input("colorscale-picker", "colorscale"),  
 ],  
 [State("brain-graph", "figure"), State("annotation\_storage", "data")],  
)  
def brain\_graph\_handler(click\_data, val, colorscale, figure, current\_anno):  
 *""" Listener on colorscale, option picker, and graph on click to update the graph. """* # new option select  
 if figure["data"][0]["name"] != val:  
 figure["data"] = create\_mesh\_data(val)  
 figure["layout"] = plot\_layout  
 cs = [[i / (len(colorscale) - 1), rgb] for i, rgb in enumerate(colorscale)]  
 figure["data"][0]["colorscale"] = cs  
 return figure  
  
 # modify graph markers  
 if click\_data is not None and "points" in click\_data:  
  
 y\_value = click\_data["points"][0]["y"]  
 x\_value = click\_data["points"][0]["x"]  
 z\_value = click\_data["points"][0]["z"]  
  
 marker = add\_marker(x\_value, y\_value, z\_value)  
 point\_index = marker\_in\_points(figure["data"], marker)  
  
 # delete graph markers  
 if len(figure["data"]) > 1 and point\_index is not None:  
  
 figure["data"].pop(point\_index)  
 anno\_index\_offset = 2 if val == "car" else 1  
 try:  
 figure["layout"]["scene"]["annotations"].pop(  
 point\_index - anno\_index\_offset  
 )  
 except Exception as error:  
 print(error)  
 pass  
  
 # append graph markers  
 else:  
  
 # iterate through the store annotations and save it into figure data  
 if current\_anno is not None:  
 for index, annotations in enumerate(  
 figure["layout"]["scene"]["annotations"]  
 ):  
 for key in current\_anno.keys():  
 if str(index) in key:  
 figure["layout"]["scene"]["annotations"][index][  
 "text"  
 ] = current\_anno[key]  
  
 figure["data"].append(marker)  
 figure["layout"]["scene"]["annotations"].append(  
 add\_annotation(x\_value, y\_value, z\_value)  
 )  
  
 cs = [[i / (len(colorscale) - 1), rgb] for i, rgb in enumerate(colorscale)]  
 figure["data"][0]["colorscale"] = cs  
  
 return figure  
  
  
@app.callback(Output("click-data", "children"), [Input("brain-graph", "clickData")])  
def display\_click\_data(click\_data):  
 return json.dumps(click\_data, indent=4)  
  
  
@app.callback(  
 Output("relayout-data", "children"), [Input("brain-graph", "relayoutData")]  
)  
def display\_relayout\_data(relayout\_data):  
 return json.dumps(relayout\_data, indent=4)  
  
  
@app.callback(  
 Output("annotation\_storage", "data"),  
 [Input("brain-graph", "relayoutData")],  
 [State("annotation\_storage", "data")],  
)  
def save\_annotations(relayout\_data, current\_data):  
 *""" Update the annotations in the dcc store. """* if relayout\_data is None:  
 raise PreventUpdate  
  
 if current\_data is None:  
 return {}  
  
 for key in relayout\_data.keys():  
  
 # to determine if the relayout has to do with annotations  
 if "scene.annotations" in key:  
 current\_data[key] = relayout\_data[key]  
  
 return current\_data  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run\_server(debug=True)

**mni.py:**

import pathlib  
import numpy as np  
  
  
DATA\_PATH = pathlib.Path(\_\_file\_\_).parent.joinpath("data").resolve()  
  
default\_colorscale = [  
 [0, "rgb(12,51,131)"],  
 [0.25, "rgb(10,136,186)"],  
 [0.5, "rgb(242,211,56)"],  
 [0.75, "rgb(242,143,56)"],  
 [1, "rgb(217,30,30)"],  
]  
  
  
def read\_mniobj(file):  
 *"""  
 Parses an obj file.  
   
 :params file: file name in data folder* ***:returns****: a tuple  
 """* def triangulate\_polygons(list\_vertex\_indices):  
 for k in range(0, len(list\_vertex\_indices), 3):  
 yield list\_vertex\_indices[k : k + 3]  
  
 with open(DATA\_PATH.joinpath(file)) as fp:  
 num\_vertices = 0  
 matrix\_vertices = []  
 k = 0  
 list\_indices = []  
  
 for i, line in enumerate(fp):  
 if i == 0:  
 num\_vertices = int(line.split()[6])  
 matrix\_vertices = np.zeros([num\_vertices, 3])  
 elif i <= num\_vertices:  
 matrix\_vertices[i - 1] = list(map(float, line.split()))  
 elif i > 2 \* num\_vertices + 5:  
 if not line.strip():  
 k = 1  
 elif k == 1:  
 list\_indices.extend(line.split())  
  
 list\_indices = [int(i) for i in list\_indices]  
 faces = np.array(list(triangulate\_polygons(list\_indices)))  
 return matrix\_vertices, faces  
  
  
def read\_obj(file):  
 *"""  
 Parses an obj file.  
   
 :params file: file name in data folder* ***:returns****: a tuple  
 """* def getIndicy(raw):  
 return int(raw.split('/')[0]) - 1  
  
 with open(DATA\_PATH.joinpath(file)) as fp:  
 vertices = []  
 indices = []  
  
 for i, line in enumerate(fp):  
 s = line.split()  
  
 if len(s) < 1:  
 continue  
  
 lineType = s[0]  
 values = s[1 : 4]  
  
 if lineType == 'v':  
 vertices.append(list(map(float, values)))  
 elif lineType == 'f':  
 indices.append(list(map(getIndicy, values)))  
  
 matrix\_vertices = np.array(vertices)  
 faces = np.array(indices)  
 return matrix\_vertices, faces  
  
  
def plotly\_triangular\_mesh(  
 vertices,  
 faces,  
 intensities=None,  
 colorscale="Viridis",  
 flatshading=False,  
 showscale=False,  
 plot\_edges=False,  
):  
  
 x, y, z = vertices.T  
 I, J, K = faces.T  
  
 if intensities is None:  
 intensities = z  
  
 mesh = {  
 "type": "mesh3d",  
 "x": x,  
 "y": y,  
 "z": z,  
 "colorscale": colorscale,  
 "intensity": intensities,  
 "flatshading": flatshading,  
 "i": I,  
 "j": J,  
 "k": K,  
 "name": "",  
 "showscale": showscale,  
 "lighting": {  
 "ambient": 0.18,  
 "diffuse": 1,  
 "fresnel": 0.1,  
 "specular": 1,  
 "roughness": 0.1,  
 "facenormalsepsilon": 1e-6,  
 "vertexnormalsepsilon": 1e-12,  
 },  
 "lightposition": {"x": 100, "y": 200, "z": 0},  
 }  
  
 if showscale:  
 mesh["colorbar"] = {"thickness": 20, "ticklen": 4, "len": 0.75}  
  
 if plot\_edges is False:  
 return [mesh]  
  
 lines = create\_plot\_edges\_lines(vertices, faces)  
 return [mesh, lines]  
  
  
def create\_plot\_edges\_lines(vertices, faces):  
 tri\_vertices = vertices[faces]  
 Xe = []  
 Ye = []  
 Ze = []  
 for T in tri\_vertices:  
 Xe += [T[k % 3][0] for k in range(4)] + [None]  
 Ye += [T[k % 3][1] for k in range(4)] + [None]  
 Ze += [T[k % 3][2] for k in range(4)] + [None]  
  
 # define the lines to be plotted  
 lines = {  
 "type": "scatter3d",  
 "x": Xe,  
 "y": Ye,  
 "z": Ze,  
 "mode": "lines",  
 "name": "",  
 "line": {"color": "rgb(70,70,70)", "width": 1},  
 }  
 return lines  
  
  
def create\_mesh\_data(option):  
  
 data = []  
 if option == "human":  
 vertices, faces = read\_mniobj("realct.obj")  
 intensities = np.loadtxt(DATA\_PATH.joinpath("realct.txt"))  
 elif option == "human\_atlas":  
 vertices, faces = read\_mniobj("surf\_reg\_model\_both.obj")  
 intensities = np.loadtxt(DATA\_PATH.joinpath("aal\_atlas.txt"))  
 elif option == "car":  
 vertices, faces = read\_obj("car.obj")  
 #intensities = np.loadtxt(DATA\_PATH.joinpath("car.txt"))  
 intensities = np.random.rand(len(vertices))  
 else:  
 raise ValueError  
  
 data = plotly\_triangular\_mesh(  
 vertices, faces, intensities, colorscale=default\_colorscale  
 )  
  
 # if option == "car":  
 # vertices, faces = read\_obj("car.obj")  
 # outer\_mesh = plotly\_triangular\_mesh(vertices, faces)[0]  
 # outer\_mesh["opacity"] = 0.5  
 # outer\_mesh["colorscale"] = "Greys"  
 # data.append(outer\_mesh)  
  
 data[0]["name"] = option  
 return data

**default.css**

/\* Grid  
–––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
.container {  
 position: relative;  
 width: 100%;  
 max-width: 960px;  
 margin: 0 auto;  
 padding: 0 20px;  
 box-sizing: border-box; }  
 .column,  
 .columns {  
 width: 100%;  
 float: left;  
 box-sizing: border-box; }  
   
 /\* For devices larger than 400px \*/  
 @media (min-width: 400px) {  
 .container {  
 width: 85%;  
 padding: 0; }  
 }  
   
 /\* For devices larger than 550px \*/  
 @media (min-width: 1024px) {  
 .container {  
 width: 80%; }  
 .column,  
 .columns {  
 margin-left: 4%; }  
 .column:first-child,  
 .columns:first-child {  
 margin-left: 0; }  
   
 .one.column,  
 .one.columns { width: 4.66666666667%; }  
 .two.columns { width: 13.3333333333%; }  
 .three.columns { width: 22%; }  
 .four.columns { width: 30.6666666667%; }  
 .five.columns { width: 39.3333333333%; }  
 .six.columns { width: 48%; }  
 .seven.columns { width: 56.6666666667%; }  
 .eight.columns { width: 65.3333333333%; }  
 .nine.columns { width: 74.0%; }  
 .ten.columns { width: 82.6666666667%; }  
 .eleven.columns { width: 91.3333333333%; }  
 .twelve.columns { width: 100%; margin-left: 0; }  
   
 .one-third.column { width: 30.6666666667%; }  
 .two-thirds.column { width: 65.3333333333%; }  
   
 .one-half.column { width: 48%; }  
   
 /\* Offsets \*/  
 .offset-by-one.column,  
 .offset-by-one.columns { margin-left: 8.66666666667%; }  
 .offset-by-two.column,  
 .offset-by-two.columns { margin-left: 17.3333333333%; }  
 .offset-by-three.column,  
 .offset-by-three.columns { margin-left: 26%; }  
 .offset-by-four.column,  
 .offset-by-four.columns { margin-left: 34.6666666667%; }  
 .offset-by-five.column,  
 .offset-by-five.columns { margin-left: 43.3333333333%; }  
 .offset-by-six.column,  
 .offset-by-six.columns { margin-left: 52%; }  
 .offset-by-seven.column,  
 .offset-by-seven.columns { margin-left: 60.6666666667%; }  
 .offset-by-eight.column,  
 .offset-by-eight.columns { margin-left: 69.3333333333%; }  
 .offset-by-nine.column,  
 .offset-by-nine.columns { margin-left: 78.0%; }  
 .offset-by-ten.column,  
 .offset-by-ten.columns { margin-left: 86.6666666667%; }  
 .offset-by-eleven.column,  
 .offset-by-eleven.columns { margin-left: 95.3333333333%; }  
   
 .offset-by-one-third.column,  
 .offset-by-one-third.columns { margin-left: 34.6666666667%; }  
 .offset-by-two-thirds.column,  
 .offset-by-two-thirds.columns { margin-left: 69.3333333333%; }  
   
 .offset-by-one-half.column,  
 .offset-by-one-half.columns { margin-left: 52%; }  
   
 }  
   
   
 /\* Base Styles  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 /\* NOTE  
 html is set to 62.5% so that all the REM measurements throughout Skeleton  
 are based on 10px sizing. So basically 1.5rem = 15px :) \*/  
 html {  
 font-size: 62.5%; }  
 body {  
 font-size: 1.5em; /\* currently ems cause chrome bug misinterpreting rems on body element \*/  
 line-height: 1.6;  
 font-weight: 400;  
 font-family: "Open Sans", "HelveticaNeue", "Helvetica Neue", Helvetica, Arial, sans-serif;  
 color: rgb(50, 50, 50); }  
   
   
 /\* Typography  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 h1, h2, h3, h4, h5, h6 {  
 margin-top: 0;  
 margin-bottom: 0;  
 font-weight: 300; }  
 h1 { font-size: 4.5rem; line-height: 1.2; letter-spacing: -.1rem; margin-bottom: 2rem; }  
 h2 { font-size: 3.6rem; line-height: 1.25; letter-spacing: -.1rem; margin-bottom: 1.8rem; margin-top: 1.8rem;}  
 h3 { font-size: 3.0rem; line-height: 1.3; letter-spacing: -.1rem; margin-bottom: 1.5rem; margin-top: 1.5rem;}  
 h4 { font-size: 2.6rem; line-height: 1.35; letter-spacing: -.08rem; margin-bottom: 1.2rem; margin-top: 1.2rem;}  
 h5 { font-size: 2.2rem; line-height: 1.5; letter-spacing: -.05rem; margin-bottom: 0.6rem; margin-top: 0.6rem;}  
 h6 { font-size: 2.0rem; line-height: 1.6; letter-spacing: 0; margin-bottom: 0.75rem; margin-top: 0.75rem;}  
   
 p {  
 margin-top: 0; }  
   
   
 /\* Blockquotes  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 blockquote {  
 border-left: 4px lightgrey solid;  
 padding-left: 1rem;  
 margin-top: 2rem;  
 margin-bottom: 2rem;  
 margin-left: 0rem;  
 }  
   
   
 /\* Links  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 a {  
 color: #1EAEDB;   
 text-decoration: underline;  
 cursor: pointer;}  
 a:hover {  
 color: #0FA0CE; }  
   
   
 /\* Buttons  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 .button,  
 button,  
 input[type="submit"],  
 input[type="reset"],  
 input[type="button"] {  
 display: inline-block;  
 height: 38px;  
 padding: 0 30px;  
 color: #555;  
 text-align: center;  
 font-size: 11px;  
 font-weight: 600;  
 line-height: 38px;  
 letter-spacing: .1rem;  
 text-transform: uppercase;  
 text-decoration: none;  
 white-space: nowrap;  
 background-color: transparent;  
 border-radius: 4px;  
 border: 1px solid #bbb;  
 cursor: pointer;  
 box-sizing: border-box; }  
 .button:hover,  
 button:hover,  
 input[type="submit"]:hover,  
 input[type="reset"]:hover,  
 input[type="button"]:hover,  
 .button:focus,  
 button:focus,  
 input[type="submit"]:focus,  
 input[type="reset"]:focus,  
 input[type="button"]:focus {  
 color: #333;  
 border-color: #888;  
 outline: 0; }  
 .button.button-primary,  
 button.button-primary,  
 input[type="submit"].button-primary,  
 input[type="reset"].button-primary,  
 input[type="button"].button-primary {  
 color: #FFF;  
 background-color: #33C3F0;  
 border-color: #33C3F0; }  
 .button.button-primary:hover,  
 button.button-primary:hover,  
 input[type="submit"].button-primary:hover,  
 input[type="reset"].button-primary:hover,  
 input[type="button"].button-primary:hover,  
 .button.button-primary:focus,  
 button.button-primary:focus,  
 input[type="submit"].button-primary:focus,  
 input[type="reset"].button-primary:focus,  
 input[type="button"].button-primary:focus {  
 color: #FFF;  
 background-color: #1EAEDB;  
 border-color: #1EAEDB; }  
   
   
 /\* Forms  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 input[type="email"],  
 input[type="number"],  
 input[type="search"],  
 input[type="text"],  
 input[type="tel"],  
 input[type="url"],  
 input[type="password"],  
 textarea,  
 select {  
 height: 38px;  
 padding: 6px 10px; /\* The 6px vertically centers text on FF, ignored by Webkit \*/  
 background-color: #fff;  
 border: 1px solid #D1D1D1;  
 border-radius: 4px;  
 box-shadow: none;  
 box-sizing: border-box;   
 font-family: inherit;  
 font-size: inherit; /\*https://stackoverflow.com/questions/6080413/why-doesnt-input-inherit-the-font-from-body\*/}  
 /\* Removes awkward default styles on some inputs for iOS \*/  
 input[type="email"],  
 input[type="number"],  
 input[type="search"],  
 input[type="text"],  
 input[type="tel"],  
 input[type="url"],  
 input[type="password"],  
 textarea {  
 -webkit-appearance: none;  
 -moz-appearance: none;  
 appearance: none; }  
 textarea {  
 min-height: 65px;  
 padding-top: 6px;  
 padding-bottom: 6px; }  
 input[type="email"]:focus,  
 input[type="number"]:focus,  
 input[type="search"]:focus,  
 input[type="text"]:focus,  
 input[type="tel"]:focus,  
 input[type="url"]:focus,  
 input[type="password"]:focus,  
 textarea:focus,  
 select:focus {  
 border: 1px solid #33C3F0;  
 outline: 0; }  
 label,  
 legend {  
 display: block;  
 margin-bottom: 0px; }  
 fieldset {  
 padding: 0;  
 border-width: 0; }  
 input[type="checkbox"],  
 input[type="radio"] {  
 display: inline; }  
 label > .label-body {  
 display: inline-block;  
 margin-left: .5rem;  
 font-weight: normal; }  
   
   
 /\* Lists  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 ul {  
 list-style: circle inside; }  
 ol {  
 list-style: decimal inside; }  
 ol, ul {  
 padding-left: 0;  
 margin-top: 0; }  
 ul ul,  
 ul ol,  
 ol ol,  
 ol ul {  
 margin: 1.5rem 0 1.5rem 3rem;  
 font-size: 90%; }  
 li {  
 margin-bottom: 1rem; }  
   
   
 /\* Tables  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 table {  
 border-collapse: collapse;  
 }  
 th,  
 td {  
 padding: 12px 15px;  
 text-align: left;  
 border-bottom: 1px solid #E1E1E1; }  
 th:first-child,  
 td:first-child {  
 padding-left: 0; }  
 th:last-child,  
 td:last-child {  
 padding-right: 0; }  
   
   
 /\* Spacing  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 button,  
 .button {  
 margin-bottom: 0rem; }  
 input,  
 textarea,  
 select,  
 fieldset {  
 margin-bottom: 0rem; }  
 pre,  
 dl,  
 figure,  
 table,  
 form {  
 margin-bottom: 0rem; }  
 p,  
 ul,  
 ol {  
 margin-bottom: 0.75rem; }  
   
 /\* Utilities  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 .u-full-width {  
 width: 100%;  
 box-sizing: border-box; }  
 .u-max-full-width {  
 max-width: 100%;  
 box-sizing: border-box; }  
 .u-pull-right {  
 float: right; }  
 .u-pull-left {  
 float: left; }  
   
   
 /\* Misc  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 hr {  
 margin-top: 3rem;  
 margin-bottom: 3.5rem;  
 border-width: 0;  
 border-top: 1px solid #E1E1E1; }  
   
   
 /\* Clearing  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
   
 /\* Self Clearing Goodness \*/  
 .container:after,  
 .row:after,  
 .u-cf {  
 content: "";  
 display: table;  
 clear: both; }  
   
   
 /\* Media Queries  
 –––––––––––––––––––––––––––––––––––––––––––––––––– \*/  
 /\*  
 Note: The best way to structure the use of media queries is to create the queries  
 near the relevant code. For example, if you wanted to change the styles for buttons  
 on small devices, paste the mobile query code up in the buttons section and style it  
 there.  
 \*/  
   
   
 /\* Larger than mobile \*/  
 @media (min-width: 400px) {}  
   
 /\* Larger than phablet (also point when grid becomes active) \*/  
 @media (min-width: 550px) {}  
   
 /\* Larger than tablet \*/  
 @media (min-width: 750px) {}  
   
 /\* Larger than desktop \*/  
 @media (min-width: 1000px) {}  
   
 /\* Larger than Desktop HD \*/  
 @media (min-width: 1200px) {}

**Style.css**

body {  
 background-color: #141414;  
 color: #F4F6F8;  
 font-family: "Open Sans", sans-serif;  
 margin: 0;  
}  
  
.pb-20 {  
 padding-bottom: 20px;  
}  
  
.red-ish {  
 color: #BA2456;  
}  
  
a:hover {  
 opacity: 0.5;  
}  
  
.header {  
 display: flex;  
 flex-direction: column;  
}  
  
.header\_\_title {  
 display: flex;  
 flex-direction: row;  
 align-items: center;  
 margin-top: 50px;  
}  
  
.header img {  
 height: 40px;  
 width: auto;  
}  
  
.header p {  
 padding: 10px 0px;  
}  
  
.header h4 {  
 font-family: "serif";  
 margin: 5px 0px 0px 20px;  
}  
  
.header a {  
 padding: 10px 15px;  
 background: transparent;  
 color: #F9FAFB;  
 text-decoration: none;  
 border-radius: 0.35rem;  
 border: 1px solid #F9FAFB;;  
 margin: 15px 10px 0px 0px;  
}  
  
.graph\_\_container {  
 display: flex;  
 justify-content: center;  
}  
  
  
.app\_\_right\_\_section {  
 background-color: #1D1D1D;  
 min-height: 100vh;  
 max-height: 100vh;  
 overflow-y: scroll;  
 overflow: scroll;  
 padding: 25px;  
 border-left: black solid 5px;  
}  
  
.app\_\_right\_\_section::-webkit-scrollbar-thumb {  
 border-radius: 5px;  
 background-color: #C4CDD5;  
 --webkit-box-shadow: 0 0 1px #C4CDD5;  
}  
  
.app\_\_right\_\_section::-webkit-scrollbar {  
 --webkit-appearance: none;  
 width: 10px;  
}  
  
.app\_\_right\_\_section::-webkit-scrollbar-corner {  
 background: rgba(0,0,0,0);  
}  
  
.subheader {  
 font-size: 1.2em;  
 color: #DFE3E8;  
}  
  
.colorscale-block {  
 margin: 0 !important;  
 border: 1px solid #C4CDD5;  
 padding: 5px 5px 0px 5px;  
 border-radius: 0.5rem;  
}  
  
.colorscale-block div {  
 margin: 0 !important;  
}  
  
.label\_\_option {  
 display: inline-block;  
 padding-right: 15px;  
}  
  
.input\_\_option {  
 margin-right: 8px;  
}  
  
.small-text {  
 font-size: 1em;  
 color: #919EAB;  
}  
  
.info\_\_container {  
 background-color: #292929;  
 color: white;  
 padding: 25px;  
}  
  
.colorscalePickerContainer {  
 background: #F4F5FA !important;  
}  
  
#brain-graph {  
 user-select: none;  
 margin: auto;  
 height: 60vh;  
}  
  
@media all and (max-width: 768px) {  
 .header\_\_title {  
 display: block;  
 margin-top: 25px;  
 }  
 .header\_\_title h4 {  
 text-align: center;  
 }  
 .header\_\_title img {  
 display: flex;  
 height: 30px;  
 width: auto;  
 padding-bottom: 10px;  
 }  
 .header\_\_info p {  
 text-align: center;  
 font-size: 14px;  
 }  
 .header\_\_button {  
 text-align: center;  
 }  
 .app\_\_right\_\_section {  
 border: none;  
 }  
 .graph\_\_container {  
 padding-top: 50px;  
 padding-bottom: 50px;  
 }  
 .app\_\_right\_\_section {  
 overflow-y: hidden;  
 overflow: hidden;  
 min-height:initial;  
 max-height: initial;  
 }  
 #radio-options {  
 font-size: 11px;  
 }  
 #brain-graph {  
 height: 350px;  
 width: 350px;  
 }  
}

# Вывод

В процессе выполнения домашнего задания мной была реализована 3D модель автомобиля с помощью Dash.