# How to incorporate the plot to the application?

**requirements.txt**

bokeh

pmdarima

json

(additional librairies to be installed, or by using pip instal *library\_name*)

**app.py**

* **Libraries to import**

import requests

import json

import bokeh

import numpy as np

import pandas as pd

from bokeh.plotting import figure, output\_file

from bokeh.models import DatetimeTickFormatter, PrintfTickFormatter, ColumnDataSource, Grid, Line, Dropdown, LinearAxis, Plot, HoverTool, CustomJS, Slider, Select, DateRangeSlider

from bokeh.io import show, curdoc

from bokeh.layouts import gridplot, column

from bokeh.embed import components

from bokeh.resources import CDN

from datetime import date, timedelta

from bokeh.transform import factor\_cmap

from pmdarima import auto\_arima

from statsmodels.tsa.arima.model import ARIMA

import warnings

from backend.gas import average\_price\_plot, ReverseGeo

* Code to paste (**right after @app.route('/', methods=('GET', 'POST'))):**

def plot():

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

    # 1. Get all the data we need to build our interactive plot   #

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

    # To get the data we need as source to build our interactive plot

    plot\_data = average\_price\_plot()

    plot\_source = plot\_data.query()

    df = json.loads(plot\_source)

    df = pd.json\_normalize(df, 'records')

    # We only keep the data required to construct our plot: gas type, the date and the related price

    df = df[['fields.prix\_nom','fields.prix\_valeur','fields.prix\_maj']]

    df = df.rename(columns={'fields.prix\_nom': "Carburant", 'fields.prix\_valeur': "Prix",'fields.prix\_maj': "Date"})

    # # We remove timestamps to keep only the date, with date format

    for i in range(len(df)):

        df.iloc[i, 2] = str(df.iloc[i, 2])[:10]

    df['Date'] = pd.to\_datetime(df['Date'], format = '%Y-%m-%d')

    df = df.sort\_values(by = ['Date'])

    # #to keep only the value we want, anc convert Carburant to columns, to make it easier to manipulate afterwards

    df1 = pd.pivot\_table(df,

                        values = 'Prix',

                        index = ['Date'],

                     columns = ['Carburant']

                        ,aggfunc=np.mean)

    df1 = df1.reset\_index()

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

    # 2. Predictions over the next 5 days #

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

    #We use the pdm library to perform a time series prediction of the average gas price for the next 5 days

    gas = ['SP95', 'SP98', 'E85','Gazole','GPLc','E10']

    J = [(date.today() + timedelta(days = i+1)).strftime("%Y-%m-%d") for i in range(5)]

    # We remove the NAN values for each type of gas (which can be very different)

    no\_nan = [df1[['Date', i]].dropna() for i in gas]

    # We obtain the ARMA(p,d,q) parameters for each gas

    parameters = [auto\_arima(no\_nan[count][value], suppress\_warnings = True, seasonal = True, m = 1).get\_params().get("order") for count, value in enumerate(gas)]

    # We fit each ARMA model on the training value (for simplicity, training values macth the historical data for each gas type)

    models = [ARIMA(endog = no\_nan[count][value], order = parameters[count]).fit() for count, value in enumerate(gas)]

    start, end = len(df), len(df) + 5

    forecast = [models[i].predict(start = start , end =  end) for i in range(len(models))]

    predictions = pd.DataFrame(0, index = np.arange(len(J)), columns = ['Date'] + gas)

    #We store the day+5 predictions into a dataframe, with the corresponding dates

    for count, value in enumerate(J):

        predictions['Date'].iloc[-count -1] = J[-count-1]

    for i in range(len(gas)):

        for j in range(len(J)):

            h = pd.DataFrame(forecast[i])

            predictions.iloc[j, i+1] = h.iloc[-j+5]

    #Reduces vizulization (and prediction base) to the past 8 weeks

    df1 = df1[df1['Date'] > str(date.today() - timedelta(weeks= 8))]

    #We eventually merge the historical data as well as the forecasted ones, which will be used as our database for plotting

    df1['Category'] = 'Historique'

    predictions['Category'] = 'Prévision'

    df2 = pd.concat([df1, predictions])

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

    #       3. Datavizualization tool     #

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

    source = ColumnDataSource(df2)

    plot = figure(width = 600,

               height = 300,

               x\_axis\_type='datetime',

               title = str('Prix journalier moyen - '+ 'SP95'))

    plot.grid.grid\_line\_alpha = 0.4

    plot.axis.minor\_tick\_out = 0

    plot.axis.major\_tick\_out = 4

    plot.legend.background\_fill\_alpha = 0.0

    plot.legend.border\_line\_alpha = 0.0

    plot.toolbar.logo = None

    plot.toolbar\_location = None

    plot.title.align = 'center'

    plot.yaxis[0].formatter = PrintfTickFormatter(format = '€ %0.2f')

    plot.xaxis[0].formatter = DatetimeTickFormatter(months=["%b %d"])

    # Creates a javascript hovertool that changes the color

    # when the user hovers on a particular value, and displays the average price as well as the date

    plot.add\_tools(HoverTool(tooltips=[('Date', '@Date{%F}'),

                                    ('Prix moyen', '$y')],

                           formatters = {'@Date': 'datetime'}))

    gas = ['SP95', 'SP98', 'E85','Gazole','GPLc','E10']

    index\_cmap = factor\_cmap('Category', palette = ['#4292c6', '#EE6677'],

                           factors = sorted(df2.Category.unique()))

    circle = plot.circle(x = "Date", y = "SP95",

                     legend\_field = 'Category',

                     line\_color = index\_cmap,

                     fill\_color = index\_cmap,

                     source = source,

                         size = 15,

                         fill\_alpha=0.4,

                         hover\_color='limegreen')

    # Creates a dropdown menu to select the desired gas type, and then update the underlying data selection

    dropdown = Dropdown(label="Choisissez votre carburant", menu = gas, button\_type = 'primary')

    code = """

      let s = source.data

      s["SP95"] = s[this.item]

      plot.title.text = 'Prix journalier moyen - ' + this.item

      plot.change.emit()

      source.change.emit();

    """

    dropdown.js\_on\_event("menu\_item\_click", CustomJS(args=dict(source=source, plot = plot, legend = plot.legend.items[0]), code=code))

    # Creates a date-slider that allows the user to select its desired timeframe from vizualization

    startdate, enddate =df2['Date'].iat[0] , df2['Date'].iat[-1]

    date\_range\_slider = DateRangeSlider(

      title="Date", start=startdate, end=enddate,

      value=(startdate, enddate), step=1, width=300)

    callback = CustomJS(args=dict(plot = plot), code="""

      plot.x\_range.start = cb\_obj.value[0]

      plot.x\_range.end = cb\_obj.value[1]

      plot.x\_range.change.emit()

      """)

    date\_range\_slider.js\_on\_change('value', callback)

    layout = column(children = [plot, dropdown, date\_range\_slider], sizing\_mode = "stretch\_width")

#     show(layout)

    script1, div1 = components(layout)

    jss= CDN.js\_files[0]

    widget = CDN.js\_files[2]

    return render\_template('index.html',

                  script1 = script1,

                  div1 = div1,

                  jss = jss,

                  widget = widget)

**gas.py**

* **Libraries to import**

from requests.adapters import HTTPAdapter

from urllib3.util.retry import Retry

import json

* Code to paste:

class average\_price\_plot:

    def \_\_init\_\_(self) -> None:

        self.url = 'https://data.economie.gouv.fr/api/records/1.0/search/?dataset=prix-carburants-fichier-instantane-test-ods-copie'

    def query(self, filters=None):

        query = self.url + '&q=&rows=10000&facet=prix\_maj&facet=prix\_nom&facet=prix\_valeur'

        if filters is not None:

            for f in filters :

                query += '&refine.%s'%f

        r = requests.get(query)

        res = r.json()

        final = json.dumps(res)

        return final

class ReverseGeo:

    def \_\_init\_\_(self) -> None:

        self.url = 'https://api.bigdatacloud.net/data/reverse-geocode-client?'

    def query(self, location=None):

        query = self.url

        if location is not None:

            query += 'latitude=' + str(location[0]) + '&longitude=' + str(location[1]) +'&localityLanguage=fr'

        session = requests.Session()

        retry = Retry(connect=5, backoff\_factor=0.5)

        adapter = HTTPAdapter(max\_retries=retry)

        session.mount('http://', adapter)

        session.mount('https://', adapter)

        r = session.get(query)

        final = r.json()

        geoloc = pd.DataFrame.from\_dict(final, orient='columns')

        return geoloc['city'].iloc[0]

**index.html**

* Code to paste:
  + In the <head> </head> section:

    <script type="text/javascript" src={{jss | safe}}></script>

    <script type="text/javascript" src={{widget | safe}}></script>

* + In the <header> </header> section (it does not currently exist. Thus, just paste the following code, between </head> and <body>)

</head>

<header>

    {{script1 |safe }}

</header>

<body>

* + In the <body> </body> section, right after <div id = ‘sidebar’>, as follows:

<div id="content">

    <!-- we create the left side bar -->

    <div id="sidebar">

        <div id ='plot'>{{div1 | safe}}</div>

        <form method="POST" >

**universal.css**

I have slightly modified the size of the sidebar to improve the plot visibility (from 20% to 30%)

#sidebar{

    height: 100%;

    width: 30%;

    justify-content: center;

    align-items: center;

    background-color: #303030;

    position: relative;

}

I have also put the form below its prior position, to avoid the plot to overflow the form, as well as to have a better balance in the content distribution (i.e., avoid the two element in the sidebar to be next to each other), with a view to improving the front-end visual appearance.

form{

    width: 90%;

    position: absolute;

    left: 50%;

    top: 60%;

    transform: translate(-50%,-50%);

}