МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
 ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

**по дисциплине**

**“Прикладное программное обеспечение в научных исследованиях”**

**Компоненты Dash**

Выполнил: студент гр. ПИм-1301-03-00 Бояринцев Д. Ф.\_\_\_\_\_\_\_\_\_\_

Проверил: к.ф.-м.н., доцент кафедры ПМИ Бызов В. А. \_\_\_\_\_\_\_\_\_\_\_

Киров

2024

Задание 1.

Анализ задержки вылетов самолетов в дашборде.

Исходный код:

import dash

from dash import dcc, html, Input, Output

import pandas as pd

import plotly.express as px

# Загрузка данных

url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/airline\_data.csv"

airline\_data = pd.read\_csv(url)

# Создание приложения Dash с внешними стилями

app = dash.Dash(\_\_name\_\_, external\_stylesheets=['./assets/styles.css'])

# Дизайн макета приложения

app.layout = html.Div([

# Заголовок

html.H1("Анализ задержки вылетов самолетов"),

# Ввод года

html.Div([

html.Label("Выберите год:", style={'font-weight': 'bold'}),

dcc.Dropdown(

id="year-dropdown",

options=[{'label': str(year), 'value': year} for year in range(2010, 2021)],

value=2010

)

]),

# Блок для линейных графиков

html.Div([

# Первый и второй графики

html.Div([

dcc.Graph(id="carrier-delay-graph"),

dcc.Graph(id="weather-delay-graph")

], className="graph-container"),

# Третий и четвертый графики

html.Div([

dcc.Graph(id="nas-delay-graph"),

dcc.Graph(id="security-delay-graph")

], className="graph-container")

]),

# Пятый график

html.Div([

dcc.Graph(id="late-aircraft-delay-graph")

])

])

# Функция для фильтрации данных по году

def filter\_data\_by\_year(data, year):

return data[data['Year'] == int(year)]

# Обратный вызов для обновления графиков

@app.callback(

[Output("carrier-delay-graph", "figure"),

Output("weather-delay-graph", "figure"),

Output("nas-delay-graph", "figure"),

Output("security-delay-graph", "figure"),

Output("late-aircraft-delay-graph", "figure")],

[Input("year-dropdown", "value")]

)

def update\_graphs(selected\_year):

filtered\_data = filter\_data\_by\_year(airline\_data, selected\_year)

# Среднемесячные значения задержек

avg\_carrier\_delay = filtered\_data.groupby(['Month', 'Reporting\_Airline'])['CarrierDelay'].mean().reset\_index()

avg\_weather\_delay = filtered\_data.groupby(['Month', 'Reporting\_Airline'])['WeatherDelay'].mean().reset\_index()

avg\_nas\_delay = filtered\_data.groupby(['Month', 'Reporting\_Airline'])['NASDelay'].mean().reset\_index()

avg\_security\_delay = filtered\_data.groupby(['Month', 'Reporting\_Airline'])['SecurityDelay'].mean().reset\_index()

avg\_late\_aircraft\_delay = filtered\_data.groupby(['Month', 'Reporting\_Airline'])['LateAircraftDelay'].mean().reset\_index()

# График CarrierDelay

carrier\_figure = px.line(avg\_carrier\_delay, x='Month', y='CarrierDelay', color='Reporting\_Airline',

title='Среднемесячная задержка по вине перевозчика')

# График WeatherDelay

weather\_figure = px.line(avg\_weather\_delay, x='Month', y='WeatherDelay', color='Reporting\_Airline',

title='Среднемесячная задержка из-за погоды')

# График NASDelay

nas\_figure = px.line(avg\_nas\_delay, x='Month', y='NASDelay', color='Reporting\_Airline',

title='Среднемесячная задержка из-за НВС')

# График SecurityDelay

security\_figure = px.line(avg\_security\_delay, x='Month', y='SecurityDelay', color='Reporting\_Airline',

title='Среднемесячная задержка из-за безопасности')

# График LateAircraftDelay

late\_aircraft\_figure = px.line(avg\_late\_aircraft\_delay, x='Month', y='LateAircraftDelay', color='Reporting\_Airline',

title='Среднемесячная задержка из-за опозданий самолета')

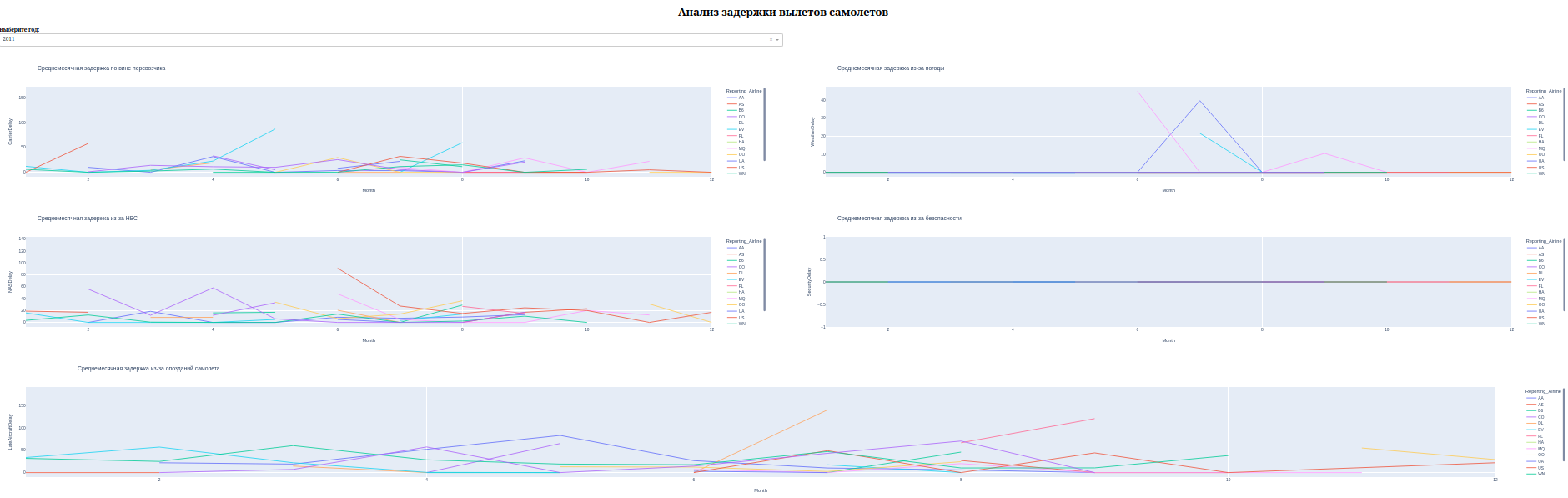
return carrier\_figure, weather\_figure, nas\_figure, security\_figure, late\_aircraft\_figure

# Запуск приложения

if \_\_name\_\_ == "\_\_main\_\_":

app.run\_server(debug=True)

Скриншот работы программы:



Выводы:

1. Разработана программа, которая по данным из датасета строит диаграммы задержки вылета самолетов по пяти параметрам;
2. На дашборд выводятся диаграммы по каждой причине задержки вылета;
3. Добавлена интерактивная возможность выбора года для отображения данных о задержках вылета

Задание 2

Визуализируйте количество заболевших и количество смертей от COVID-19 в разных регионах России

Исходный код:

import dash

from dash import dcc, html, callback, Output, Input

import pandas as pd

import plotly.express as px

import os

# Создание приложения Dash

app = dash.Dash(\_\_name\_\_, external\_stylesheets=['styles.css'])

# Путь к данным

DATA\_PATH = "po\_ni/Лабораторная работа 2/task\_2/data/covid\_polymatica.xlsx"

# Загрузка данных

if not os.path.exists(DATA\_PATH):

raise FileNotFoundError("Файл с данными не найден. "

"Убедитесь, что данные находятся в папке 'data'.")

df = pd.read\_excel(DATA\_PATH)

df = df.rename(columns={'Регион ': 'region',

'Федеральный округ': 'FederalDistrict',

'дата': 'date',

'случаи заболевания': 'casesofdisease',

'население': 'population',

'количество смертей': 'numberofdeaths'})

# df

# df.info()

# df['date'] = pd.to\_datetime(df['date'])

# df

# df.info()

# Разметка приложения

app.layout = html.Div([

html.H1("Статистика по COVID-19 в России", className="title"),

# Выбор региона

html.Label("Выберите регион:", className="label"),

dcc.Dropdown(

id='region-dropdown',

options=[{'label': region, 'value': region} for region in df['region'].unique()],

value=df['region'].iloc[0],

className="dropdown"

),

# Выбор начальной даты

html.Label("Начальная дата:", className="label"),

dcc.DatePickerSingle(

id='start-date-picker',

min\_date\_allowed=df['date'].min(),

max\_date\_allowed=df['date'].max(),

initial\_visible\_month=df['date'].min(),

date=df['date'].min(),

className="datepicker"

),

# Выбор конечной даты

html.Label("Конечная дата:", className="label"),

dcc.DatePickerSingle(

id='end-date-picker',

min\_date\_allowed=df['date'].min(),

max\_date\_allowed=df['date'].max(),

initial\_visible\_month=df['date'].max(),

date=df['date'].max(),

className="datepicker"

),

# График заболевших

dcc.Graph(id='confirmed-graph', className="graph"),

# График смертей

dcc.Graph(id='deaths-graph', className="graph")

])

# Callback для обновления графиков

@callback(

[Output('confirmed-graph', 'figure'),

Output('deaths-graph', 'figure')],

[Input('region-dropdown', 'value'),

Input('start-date-picker', 'date'),

Input('end-date-picker', 'date')]

)

def update\_graphs(region, start\_date, end\_date):

filtered\_df = df[(df['region'] == region) &

(df['date'] >= start\_date) &

(df['date'] <= end\_date)]

# График заболевших

confirmed\_fig = px.line(filtered\_df,

x='date',

y='casesofdisease',

title=f"Количество заболевших в {region}")

confirmed\_fig.update\_layout(xaxis\_title="Дата", yaxis\_title="Заболевшие")

# График смертей

deaths\_fig = px.line(filtered\_df,

x='date',

y='numberofdeaths',

title=f"Количество смертей в {region}")

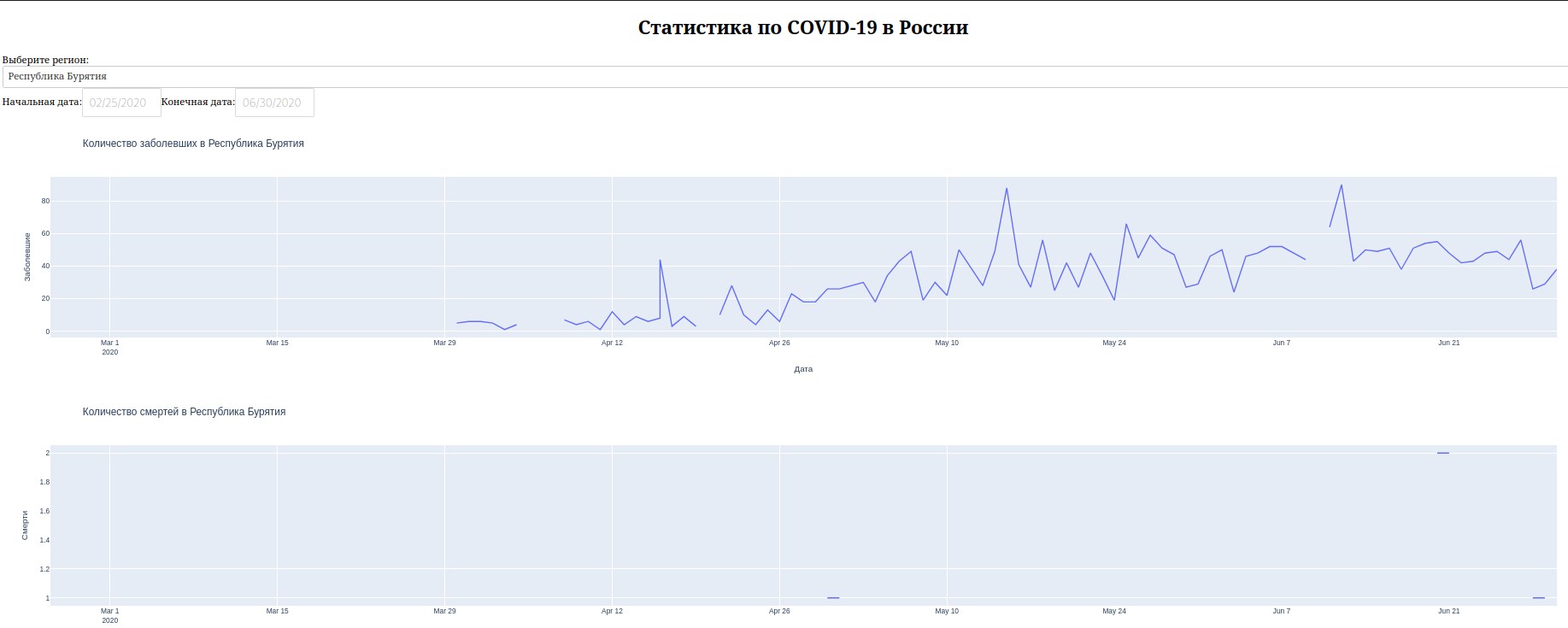
deaths\_fig.update\_layout(xaxis\_title="Дата", yaxis\_title="Смерти")

return confirmed\_fig, deaths\_fig

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

Скриншот работы программы:



Выводы:

1. Разработана программа отображения статистики заболевших и умерших от COVID-19 в России;
2. Программы выводит на дашборд информацию о заболевших и умерших от COVID-19 по регионам России;
3. Добавлены элементы интерактивности:

* Выбор региона;
* Выбор начальной даты;
* Выбор конечной даты.