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 № 3

**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.

Необходимо создать приложение Plotly Dash, позволяющее пользователям выполнять интерактивный визуальный анализ данных запуска кораблей SpaceX.

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/"

"IBM-DS0321EN-SkillsNetwork/datasets/spacex\_launch\_dash.csv")

df = pd.read\_csv(url)

# Создание приложения Dash с внешними стилями

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

# Layout приложения

app.layout = html.Div([

html.H1("Анализ запусков SpaceX", style={'text-align': 'center'}),

# Выпадающий список для выбора стартового комплекса

dcc.Dropdown(

id='site-dropdown',

options=[

{'label': 'Все стартовые комплексы', 'value': 'ALL'},

\* [{'label': site, 'value': site} for site in df['Launch Site'].unique()]

],

value='ALL',

placeholder="Выберите стартовый комплекс",

searchable=True,

style={'width': '50%', 'margin': 'auto'}

),

# Круговая диаграмма

html.Div(dcc.Graph(id='success-pie-chart'), style={'margin': '20px'}),

# Ползунок для выбора диапазона полезной нагрузки

html.P("Диапазон массы полезной нагрузки (кг):"),

dcc.RangeSlider(

id='payload-slider',

min=df['Payload Mass (kg)'].min(),

max=df['Payload Mass (kg)'].max(),

step=1000,

marks={str(i): str(i) for i in range(0, 10001, 2500)},

value=[df['Payload Mass (kg)'].min(), df['Payload Mass (kg)'].max()]

),

# Точечная диаграмма

html.Div(dcc.Graph(id='success-payload-scatter-chart'), style={'margin': '20px'})

])

# Callback для обновления круговой диаграммы

@app.callback(

Output('success-pie-chart', 'figure'),

[Input('site-dropdown', 'value')]

)

def update\_pie\_chart(selected\_site):

if selected\_site == 'ALL':

# Фильтруем только успешные запуски (class == 1)

success\_only\_df = df[df['class'] == 1]

# Группировка данных по стартовым комплексам и подсчет успешных запусков

success\_counts = success\_only\_df.groupby('Launch Site').size().reset\_index(name='count')

# Создание списка уникальных стартовых комплексов

launch\_sites = success\_counts['Launch Site'].unique()

# Определение цветовой палитры для стартовых комплексов

color\_map = px.colors.qualitative.Plotly[:len(launch\_sites)] # Берем столько цветов, сколько комплексов

# Создание круговой диаграммы для всех стартовых комплексов с учетом только успешных запусков

fig = px.pie(

success\_counts,

values='count',

names='Launch Site',

title='Общая успешность запусков по всем стартовым комплексам',

color='Launch Site', # Используем стартовые комплексы как категории для цветов

color\_discrete\_sequence=color\_map, # Уникальные цвета для каждого комплекса

labels={'Launch Site': 'Стартовый комплекс'}

)

fig.update\_traces(textinfo='percent+label') # Показывать проценты и метки

else:

# Фильтрация данных для выбранного стартового комплекса

filtered\_df = df[df['Launch Site'] == selected\_site]

success\_counts = filtered\_df['class'].value\_counts().reset\_index()

success\_counts.columns = ['class', 'count']

# Создание круговой диаграммы для конкретного стартового комплекса

fig = px.pie(

success\_counts,

values='count',

names='class',

title=f'Успешность запусков для {selected\_site}',

color='class',

color\_discrete\_map={0: 'red', 1: 'green'}, # Цветовая кодировка для успешности

labels={'class': 'Результат миссии'}

)

return fig

# Callback для обновления точечной диаграммы

@app.callback(

Output('success-payload-scatter-chart', 'figure'),

[Input('site-dropdown', 'value'), Input('payload-slider', 'value')]

)

def update\_scatter\_chart(selected\_site, payload\_range):

low, high = payload\_range

filtered\_df = df[(df['Payload Mass (kg)'] >= low) & (df['Payload Mass (kg)'] <= high)]

if selected\_site != 'ALL':

filtered\_df = filtered\_df[filtered\_df['Launch Site'] == selected\_site]

fig = px.scatter(

filtered\_df,

x='Payload Mass (kg)',

y='class',

color='Booster Version Category',

title='Зависимость успешности запуска от массы полезной нагрузки',

labels={

'Payload Mass (kg)': 'Масса полезной нагрузки (кг)',

'class': 'Результат миссии',

'Booster Version Category': 'Категория ускорителя'

}

)

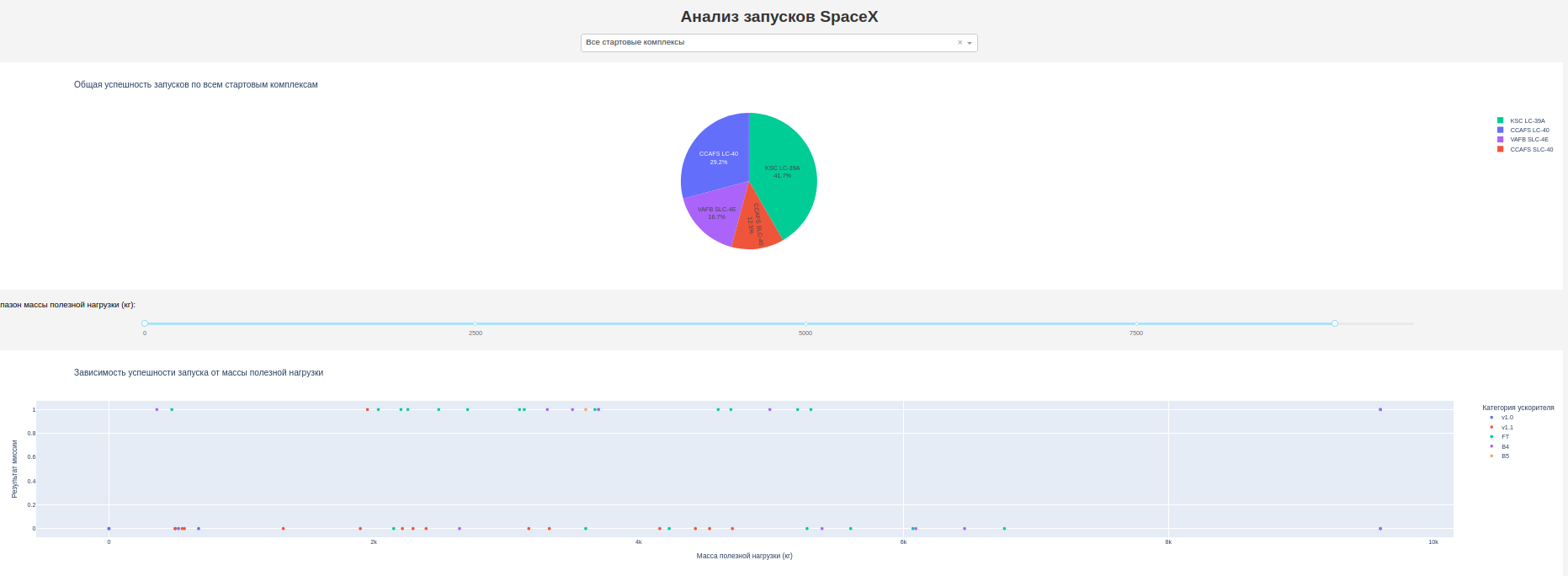
return fig

# Запуск сервера

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

app.run\_server(debug=True)

Скриншот программы:



Ответы:

1. Больше всего успешных запусков было со стартового комплекса KSC LC-39A;
2. На стартовом комплексе KSC LC-39A были успешные запуски с самой большой полезной нагрузкой;
3. У стартового комплекса KSC LC-39A (76,9%) самый высокий показатель успешности запуска;
4. FT версия F9 Booster имеет самый высокий показатель успешности запуска.

Задание 2

Создайте веб-приложение, в котором будет отображаться текущий курс валют

относительно выбранной базовой валюты

Исходный код:

import dash

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

import requests

import pandas as pd

import plotly.express as px

# Инициализация Dash приложения с подключением внешнего CSS

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

# Словарь для перевода валют на русский язык

currency\_names\_ru = {

'USD': 'Доллар США',

'EUR': 'Евро',

'RUB': 'Российский рубль',

'GBP': 'Фунт стерлингов',

'JPY': 'Японская иена',

'CAD': 'Канадский доллар',

'AUD': 'Австралийский доллар',

'CHF': 'Швейцарский франк',

'CNY': 'Китайский юань',

'SEK': 'Шведская крона'

}

# Функция для получения курсов валют

def get\_exchange\_rates(base\_currency):

url = f"https://api.exchangerate-api.com/v4/latest/{base\_currency}"

response = requests.get(url)

if response.status\_code == 200:

data = response.json()

return data['rates']

else:

return None

# Создание DataFrame из курсов валют с русскими названиями

def create\_dataframe(rates, base\_currency):

df = pd.DataFrame(list(rates.items()), columns=['Currency', 'Rate'])

df['Currency'] = df['Currency'].map(currency\_names\_ru) # Переводим названия валют

df['Base Currency'] = currency\_names\_ru.get(base\_currency, base\_currency) # Базовая валюта

return df

# Layout приложения

app.layout = html.Div([

html.H1("Конвертер валют и график курсов"),

# Выбор базовой валюты

html.Label("Выберите базовую валюту:"),

dcc.Dropdown(

id='base-currency',

options=[

{'label': 'Доллар США (USD)', 'value': 'USD'},

{'label': 'Евро (EUR)', 'value': 'EUR'},

{'label': 'Российский рубль (RUB)', 'value': 'RUB'},

{'label': 'Фунт стерлингов (GBP)', 'value': 'GBP'}

],

value='USD'

),

# График курсов валют

dcc.Graph(id='currency-chart'),

# Конвертер валют

html.Div([

html.H3("Конвертер валют"),

html.Label("Из валюты:"),

dcc.Input(id="from-currency", type="text", value="USD"),

html.Label("В валюту:"),

dcc.Input(id="to-currency", type="text", value="RUB"),

html.Label("Сумма:"),

dcc.Input(id="amount", type="number", value=1),

html.Button("Конвертировать", id="convert-button", n\_clicks=0),

html.Div(id="conversion-result")

])

])

# Callback для обновления графика курсов

@app.callback(

Output('currency-chart', 'figure'),

[Input('base-currency', 'value')]

)

def update\_graph(base\_currency):

rates = get\_exchange\_rates(base\_currency)

if rates:

df = create\_dataframe(rates, base\_currency)

# Создаем гистограмму

fig = px.histogram(

df,

x="Currency", # Ось X - названия валют

y="Rate", # Ось Y - значения курсов

log\_y=True, # Логарифмическая шкала по Y

title=f"Курс валют относительно {currency\_names\_ru.get(base\_currency, base\_currency)}"

)

# Настройка русских подписей

fig.update\_layout(

xaxis\_title="Валюта", # Подпись оси X

yaxis\_title="Курс (логарифмическая шкала)", # Подпись оси Y

font=dict(family="Arial", size=14), # Шрифт для текста

title\_font=dict(size=20) # Размер шрифта для заголовка

)

# Добавляем аннотации для значений курсов

annotations = []

for i in range(len(df)):

annotations.append(

dict(

x=df.iloc[i]['Currency'],

y=df.iloc[i]['Rate'],

text=f"{df.iloc[i]['Rate']:.2f}",

showarrow=False,

font=dict(color="black", size=10),

xanchor='center',

yanchor='bottom'

)

)

fig.update\_layout(annotations=annotations)

return fig

else:

return {}

# Callback для конвертации валют

@app.callback(

Output('conversion-result', 'children'),

[Input('convert-button', 'n\_clicks')],

[State('from-currency', 'value'),

State('to-currency', 'value'),

State('amount', 'value')]

)

def convert\_currency(n\_clicks, from\_currency, to\_currency, amount):

if n\_clicks > 0:

rates = get\_exchange\_rates(from\_currency)

if rates and to\_currency in rates:

converted\_amount = amount \* rates[to\_currency]

return f"{amount} {from\_currency} = {converted\_amount:.2f} {to\_currency}"

else:

return "Ошибка: Не удалось получить данные о курсах валют."

return ""

# Запуск сервера

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

app.run\_server(debug=True)

Скриншот программы:



Выводы:

Создана программа выводящая гистограмму курсов валют относительно базовой валюты, а также конвертор валют.