**CODE for creating dashboard using Python**

# Import required packages

import pandas as pd

import plotly.express as px

import dash

from dash import dcc

from dash import html

# Read the airline data into pandas dataframe

airline\_data =  pd.read\_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/airline\_data.csv',

                            encoding = "ISO-8859-1",

                            dtype={'Div1Airport': str, 'Div1TailNum': str,

                                   'Div2Airport': str, 'Div2TailNum': str})

# Randomly sample 500 data points. Setting the random state to be 42 so that we get same result.

data = airline\_data.sample(n=500, random\_state=42)

# Pie Chart Creation

fig = px.pie(data, values='Flights', names='DistanceGroup', title='Distance group proportion by flights')

# Create a dash application

app = dash.Dash(\_\_name\_\_)

# Get the layout of the application and adjust it.

# Create an outer division using html.Div and add title to the dashboard using html.H1 component

# Add description about the graph using HTML P (paragraph) component

# Finally, add graph component.

app.layout = html.Div(children=[html.H1('Airline Dashboard', style={'textAlign': 'center', 'color': '#503D36', 'font-size': 40}),

                                html.P('Proportion of distance group (250 mile distance interval group) by flights.', style={'textAlign':'center', 'color': '#F57241'}),

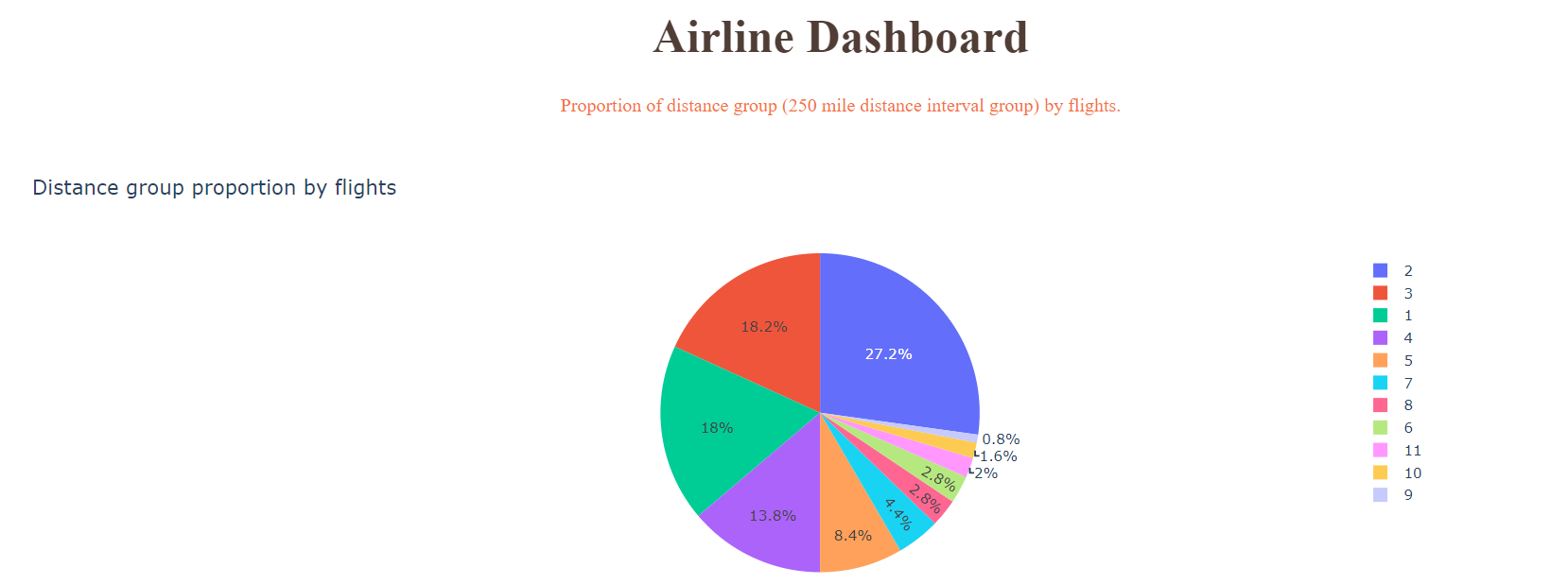
                                dcc.Graph(figure=fig),

                    ])

# Run the application

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

    app.run\_server()



**Installation and execution of python code in terminal**

python3.8 -m pip install packaging

python3.8 -m pip install pandas dash

pip3 install httpx==0.20 dash plotly

* Run the python file using the following command in the terminal

python3.8 dash\_basics.py

# Import required libraries

import pandas as pd

import plotly.graph\_objects as go

import dash

from dash import dcc

from dash import html

from dash.dependencies import Input, Output

# Read the airline data into the pandas dataframe

airline\_data = pd.read\_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/airline\_data.csv',

encoding = "ISO-8859-1",

dtype={'Div1Airport': str, 'Div1TailNum': str,

'Div2Airport': str, 'Div2TailNum': str})

# Create a dash application

app = dash.Dash(\_\_name\_\_)

app.layout = html.Div(children=[ html.H1('Airline Performance Dashboard',style={'textAlign': 'center', 'color': '#503D36', 'font-size': 40}),

html.Div(["Input Year: ", dcc.Input(id='input-year', value='2010',

type='number', style={'height':'50px', 'font-size': 35}),],

style={'font-size': 40}),

html.Br(),

html.Br(),

html.Div(dcc.Graph(id='line-plot')),

])

# add callback decorator

@app.callback( Output(component\_id='line-plot', component\_property='figure'),

Input(component\_id='input-year', component\_property='value'))

# Add computation to callback function and return graph

def get\_graph(entered\_year):

# Select 2019 data

df = airline\_data[airline\_data['Year']==int(entered\_year)]

# Group the data by Month and compute average over arrival delay time.

line\_data = df.groupby('Month')['ArrDelay'].mean().reset\_index()

fig = go.Figure(data=go.Scatter(x=line\_data['Month'], y=line\_data['ArrDelay'], mode='lines', marker=dict(color='green')))

fig.update\_layout(title='Month vs Average Flight Delay Time', xaxis\_title='Month', yaxis\_title='ArrDelay')

return fig

# Run the app

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

app.run\_server()

A graph showing a green line

Description automatically generated

# Import required libraries

import pandas as pd

import dash

from dash import dcc

from dash import html

from dash.dependencies import Input, Output

import plotly.express as px

# Read the airline data into pandas dataframe

airline\_data = pd.read\_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/airline\_data.csv',

encoding = "ISO-8859-1",

dtype={'Div1Airport': str, 'Div1TailNum': str,

'Div2Airport': str, 'Div2TailNum': str})

# Create a dash application

app = dash.Dash(\_\_name\_\_)

# Build dash app layout

app.layout = html.Div(children=[ html.H1('Flight Delay Time Statistics',

style={'textAlign': 'center', 'color': '#503D36',

'font-size': 30}),

html.Div(["Input Year: ", dcc.Input(id='input-year', value='2010',

type='number', style={'height':'35px', 'font-size': 30}),],

style={'font-size': 30}),

html.Br(),

html.Br(),

# Segment 1

html.Div([

html.Div(dcc.Graph(id='carrier-plot')),

html.Div(dcc.Graph(id='weather-plot'))

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

# Segment 2

html.Div([

html.Div(dcc.Graph(id='nas-plot')),

html.Div(dcc.Graph(id='security-plot'))

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

# Segment 3

html.Div(dcc.Graph(id='late-plot'), style={'width':'65%'})

])

""" Compute\_info function description

This function takes in airline data and selected year as an input and performs computation for creating charts and plots.

Arguments:

airline\_data: Input airline data.

entered\_year: Input year for which computation needs to be performed.

Returns:

Computed average dataframes for carrier delay, weather delay, NAS delay, security delay, and late aircraft delay.

"""

def compute\_info(airline\_data, entered\_year):

# Select data

df = airline\_data[airline\_data['Year']==int(entered\_year)]

# Compute delay averages

avg\_car = df.groupby(['Month','Reporting\_Airline'])['CarrierDelay'].mean().reset\_index()

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

avg\_NAS = df.groupby(['Month','Reporting\_Airline'])['NASDelay'].mean().reset\_index()

avg\_sec = df.groupby(['Month','Reporting\_Airline'])['SecurityDelay'].mean().reset\_index()

avg\_late = df.groupby(['Month','Reporting\_Airline'])['LateAircraftDelay'].mean().reset\_index()

return avg\_car, avg\_weather, avg\_NAS, avg\_sec, avg\_late

"""Callback Function

Function that returns fugures using the provided input year.

Arguments:

entered\_year: Input year provided by the user.

Returns:

List of figures computed using the provided helper function `compute\_info`.

"""

# Callback decorator

@app.callback( [

Output(component\_id='carrier-plot', component\_property='figure'),

Output(component\_id='weather-plot', component\_property='figure'),

Output(component\_id='nas-plot', component\_property='figure'),

Output(component\_id='security-plot', component\_property='figure'),

Output(component\_id='late-plot', component\_property='figure')

],

Input(component\_id='input-year', component\_property='value'))

# Computation to callback function and return graph

def get\_graph(entered\_year):

# Compute required information for creating graph from the data

avg\_car, avg\_weather, avg\_NAS, avg\_sec, avg\_late = compute\_info(airline\_data, entered\_year)

# Line plot for carrier delay

carrier\_fig = px.line(avg\_car, x='Month', y='CarrierDelay', color='Reporting\_Airline', title='Average carrrier delay time (minutes) by airline')

# Line plot for weather delay

weather\_fig = px.line(avg\_weather, x='Month', y='WeatherDelay', color='Reporting\_Airline', title='Average weather delay time (minutes) by airline')

# Line plot for nas delay

nas\_fig = px.line(avg\_NAS, x='Month', y='NASDelay', color='Reporting\_Airline', title='Average NAS delay time (minutes) by airline')

# Line plot for security delay

sec\_fig = px.line(avg\_sec, x='Month', y='SecurityDelay', color='Reporting\_Airline', title='Average security delay time (minutes) by airline')

# Line plot for late aircraft delay

late\_fig = px.line(avg\_late, x='Month', y='LateAircraftDelay', color='Reporting\_Airline', title='Average late aircraft delay time (minutes) by airline')

return[carrier\_fig, weather\_fig, nas\_fig, sec\_fig, late\_fig]

# Run the app

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

app.run\_server()

A screenshot of a graph

Description automatically generated

A screenshot of a graph

Description automatically generated

