```
# Dash is a code framework to display data and plots. Plotly is a
visualization tool
# to create nice plots which are more interactive than what we
typically get from
# matplotlib.pyplot
# Dash is used by companys, e.g. Nordea use it to create dashboards
and data displays

# Two ways to import dash, either like we do here or from dash
import Dash, html etc
import dash
import plotly.express as px
import pandas as pd
#%%
# To display Plotly interactively in browser
import plotly.io as pio
pio.renderers.default='browser'

#%%
# The data set consists of 50 samples from each of three species of
Iris (Iris setosa
# , Iris virginica and Iris versicolor). Four features were measured
from each
# sample: the length and the width of the sepals and petals, in
centimeters.

# Based on the combination of these four features, Fisher developed
a linear
# discriminant model to distinguish the species from each other



# From plotly we can get some standard data-sets to practice plots
(also common for ML)
df_iris = px.data.iris()

# We let x and y be the sepal lenght and width, we let size of
scatter points
# be petal lenght, and we let color be determined by species.
fig = px.scatter(df_iris, x = 'sepal_width',
                 y = 'sepal_length',
                 size = 'petal_length',
                 color = 'species')



fig.show()
#%%
# We will utilize the iris-figure in our dash-app.


# Simple Dash dashboard (1)


# Constructor, layout, main
# The dashboard:
# This is the constructor of the dashboard, i.e. our initialization
of the app
# sort of like creating a class object
app = dash.Dash(__name__)

# The layout:
# This is the visual components that should be displayed in the web
browser.
# This is where we typically "design" the web browser with html
code.
# We have the typical html-tags like .Div, .H1 and so forth which we
use to
# create the layout.
# We sort of create a family "tree" with a parameter called children
to declare
# What components that should belong where.

# Initialize children
app.layout = dash.html.Div(

    children=[
    # Create title
    dash.html.H1(children='Iris Graph Display'
                 ),
    # create under-text
    dash.html.Div(children='''
        Standard dataset visualization by LINC_STEM'''
        ),
    # include graph
    # id: unique identification of component. useful when we trigger
some kind of event
    # such as user input, tells us what component to interact with

    # Also, useful for callbacks which we will get to later
    dash.dcc.Graph(id='example-graph',
        figure=fig
        )
    ]
)


# Flask: web framework for Python
# Allows us to combine html and python, also allows for a built-in
development
# server to test web app locally

# runs the app:
# If the script is being run as main program, i.e. not imported as a
module, the
# __name__ variable gets set to main and we run the app.
if __name__ == '__main__':
    app.run_server(debug=False)
```

```
#%%
# # Simple Dash dashboard (2): added customizations

# Customize the plot
colors = {'background' : '#004d4d',
          'text' : '#ccffff'}

fig.update_layout(
    #plot_bgcolor=colors['background'],
    paper_bgcolor=colors['background'],
    font_color=colors['text']
    )

# The dashboard:

app = dash.Dash(__name__)

# The layout, customized:
app.layout = dash.html.Div(
    # Add colors
    style={'backgroundColor': colors['background']},
    children=[
    # Add color and alignment
    dash.html.H1(children='Iris Graph Display',
                  style={'textAlign': 'center',
                        'color': colors['text']}
                  ),
    dash.html.Div(children='''
        Standard dataset visualization by LINC_STEM'''

        ),
    # dcc := dash core components
    dash.dcc.Graph(id='example-graph',
        figure=fig
        )
    ]
    )

# runs the app:
if __name__ == '__main__':
    app.run_server(debug=False)


#%% Add input

colors = {'background' : '#004d4d',
          'text' : '#ccffff'}

fig.update_layout(
    #plot_bgcolor=colors['background'],
    paper_bgcolor=colors['background'],
    font_color=colors['text']
    )
```

```
app = dash.Dash(__name__)

app.layout = dash.html.Div(
    style={'backgroundColor':colors['background']},
    children=[
    dash.html.H1(
        children = 'Iris Graph Display',
        style={'color':colors['text'],
                'textAlign':'Center'}),
    dash.html.Div(
        children = '''
        Standard dataset visualization by LINC_STEM
        '''),

    # Add input
    dash.html.Div(
        children = ["Input - your name:",
                    dash.dcc.Input( # input creates input box
                        id='my-input',
                        value='your name',
                        type='text')]), # type specifies format of
input
    # Adding output of user interaction.
    dash.html.Div(
        id='my-output'
        ),

    dash.dcc.Graph(
        id = 'example-graph',
        figure=fig)
    ]
)

# So far we have built a static app, now we will add the possibility
to interact with the program
# by using callbacks. We give id's to input and output components,
which will be
# used by the callback function to identify the components.
# the function decorator facilitating the callback functionality

# A decorator in Python is a design pattern that allows you to
modify or extend the
# behavior of functions or classes. Decorators are essentially
functions that take
# another function or class as input, add some functionality to it,
and return it.

# Callbacks are functions in Dash that are automatically called
whenever an input
# component's property changes.

# component property: specifies the output of the callback
# Input: It indicates that the callback function will be triggered
whenever the
# value property of the component with the id 'my-input' changes.
@app.callback(dash.Output(component_id='my-output',
component_property='children'),
              dash.Input(component_id='my-input',
component_property='value'))
# The parameter input_value refers to the component of the Input, so
we create the output inside
# the update_name function.
def update_name(input_value):
    return f'Your name is {input_value}'

# Same thing as:
app.callback(dash.Output('my-output', 'children'), dash.Input('my-
input', 'value'))(update_name)

# Note that we can add a bunch of callbacks after each other.

if __name__ == '__main__':
    app.run_server(debug=True)

#%%

# Customize the plot
colors = {'background' : '#004d4d',
          'text' : '#ccffff'}

fig.update_layout(
    #plot_bgcolor=colors['background'],
    paper_bgcolor=colors['background'],
    font_color=colors['text']
    )

app = dash.Dash(__name__)

app.layout = dash.html.Div(
    style={'backgroundColor':colors['background']},
    children=[
    dash.html.H1(
        children = 'Iris Graph Display',
        style={'color':colors['text'],
                'textAlign':'Center'}),
    dash.html.Div(
        children = '''
        Standard dataset visualization by LINC_STEM
        '''),

    # Add input
    dash.html.Div(
        children = ["Input - your name:",
                    dash.dcc.Input( # input creates input box
                        id='my-input',
                        value='your name',
                        type='text',  # type specifies format of
input
                        debounce= True)]), # Add debounce to reduce
callback invocations
    # Adding output of user interaction.
    dash.html.Div(
        id='my-output'
        ),

    dash.dcc.Graph(
        id = 'example-graph',
        figure=fig)
    ]
)


@app.callback(dash.Output(component_id='my-output',
component_property='children'),
              # Checks if enter has been pressed
              dash.Input(component_id='my-input',
component_property='n_submit'),
              dash.State(component_id='my-input',
component_property='value'))

def update_name(n_submit, input_value):
    if n_submit is not None:
        return f'Your name is {input_value}'

# Note that we can add a bunch of callbacks after each other.

if __name__ == '__main__':
    app.run_server(debug=True)


#%%
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is
called.")
        func()
        print("Something is happening after the function is
called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```