Practice Assignment

Objectives

After completing the lab you will be able to:

- Create a dash board layout
- Add a bar chart

Estimated time needed: 45 minutes

About Skills Network Cloud IDE

This Skills Network Labs Cloud IDE (Integrated Development Environment) provides a hands-on environment in your web browser for completing course and project related labs. It utilizes Theia, an open-source IDE platform, that can be run on desktop or on the cloud. So far in the course you have been using Jupyter notebooks to run your python code. This IDE provides an alternative for editing and running your Python code. In this lab you will be using this alternative Python runtime to create and launch your Dash applications.

Important Notice about this lab environment

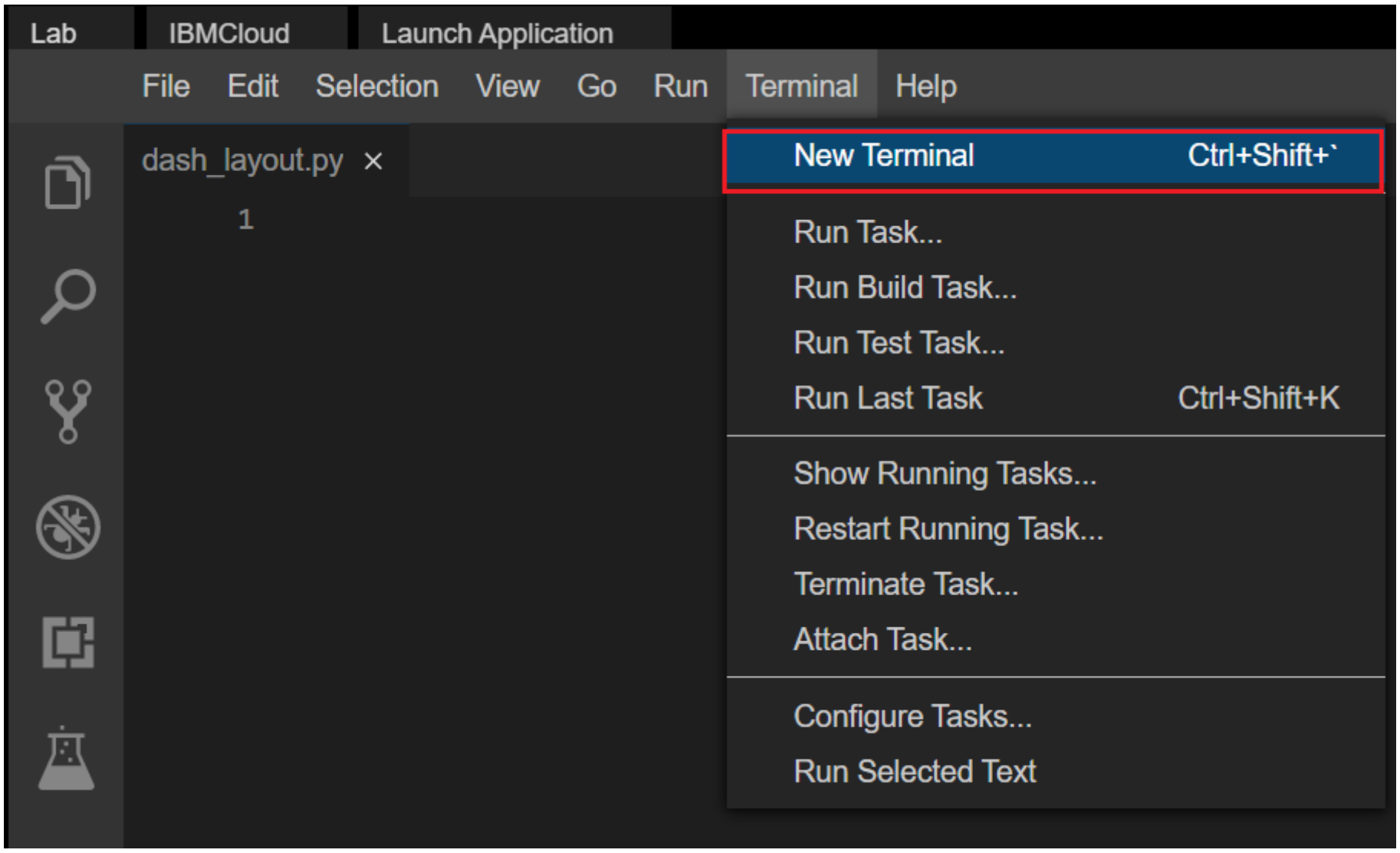
Please be aware that sessions for this lab environment are not persisted. When you launch the Cloud IDE, you are presented with a 'dedicated computer on the cloud' exclusively for you. This is available to you as long as you are actively working on the labs.

Once you close your session or it is timed out due to inactivity, you are logged off, and this ‘dedicated computer on the cloud’ is deleted along with any files you may have created, dowloaded or installed. The next time you launch this lab, a new environment is created for you.

If you finish only part of the lab and return later, you may have to start from the beginning. So, it is a good idea to plan to your time accordingly and finish your labs in a single session.

Get the tool ready

1. Open a new terminal, by clicking on the menu bar and selecting **Terminal**->**New Terminal**, as in the image below.



2. Install python packages required to run the application. Copy and paste the below command to the terminal.

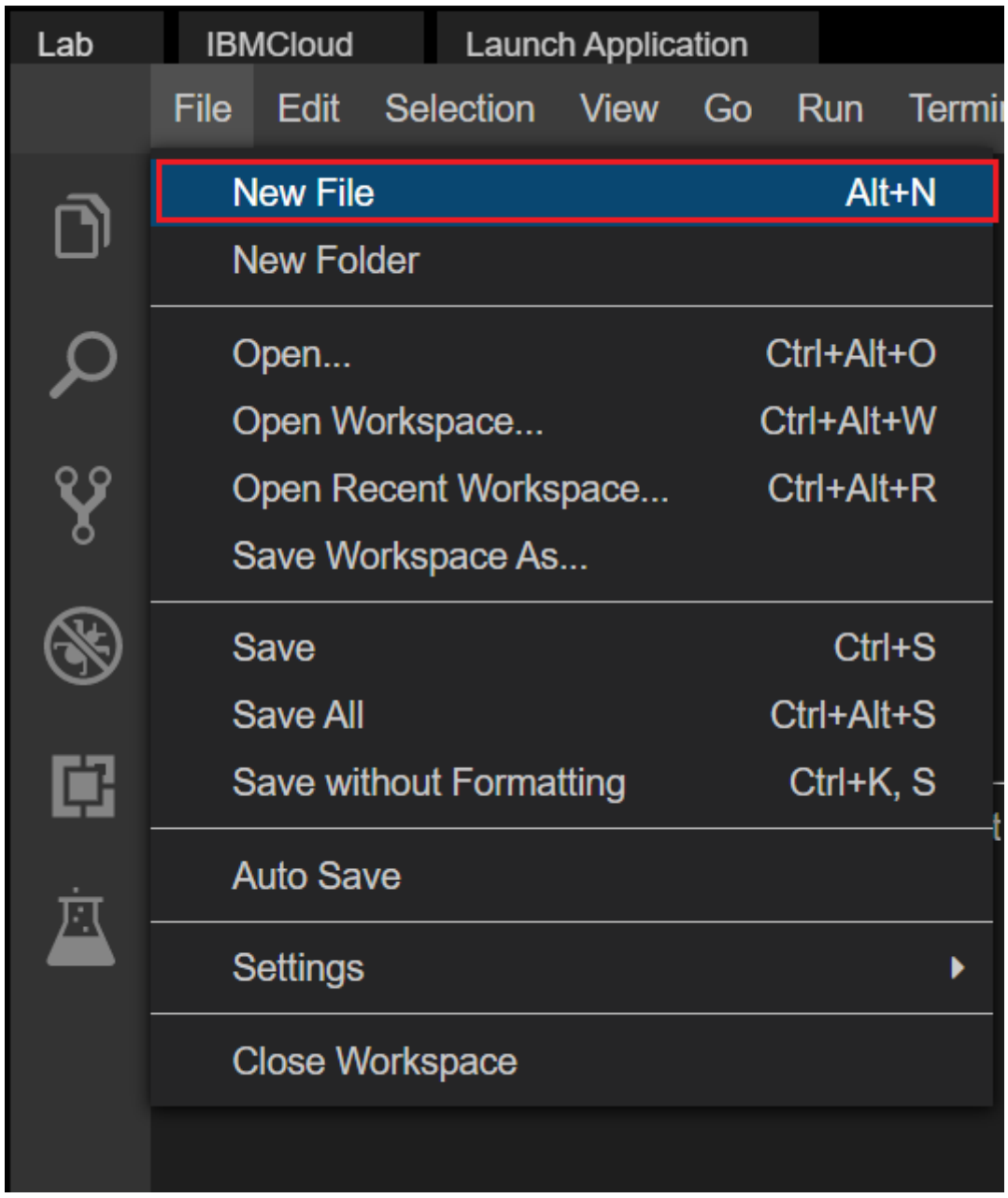
```
pip3 install pandas dash
```

TASK 1 - Dash Application layout

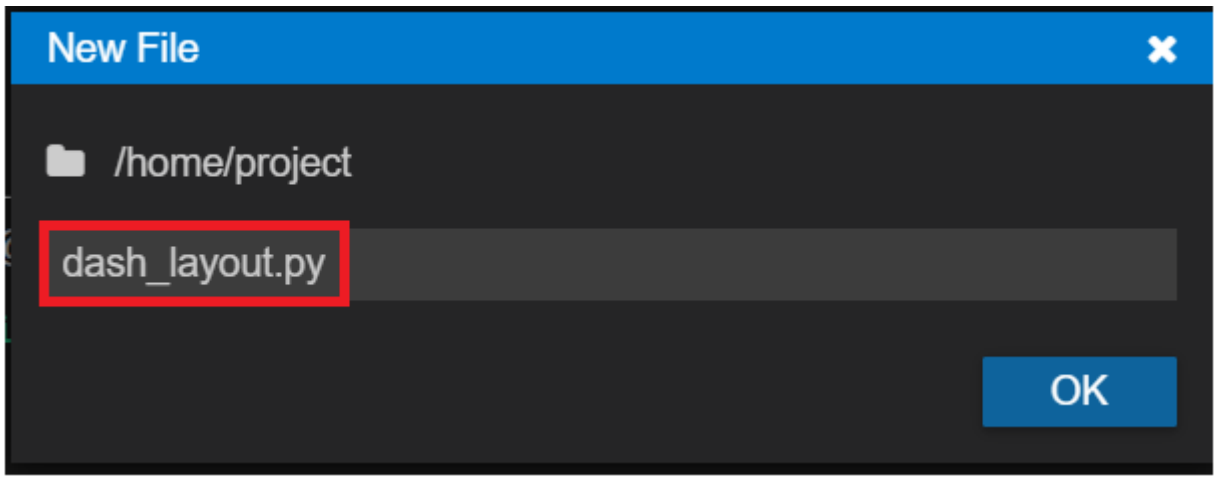
Let's start with

- Importing necessary libraries
- Title added using `html.H1()` tag

1. Create a new python script, by clicking on the menu bar and selecting **File**->**New File**, as in the image below.



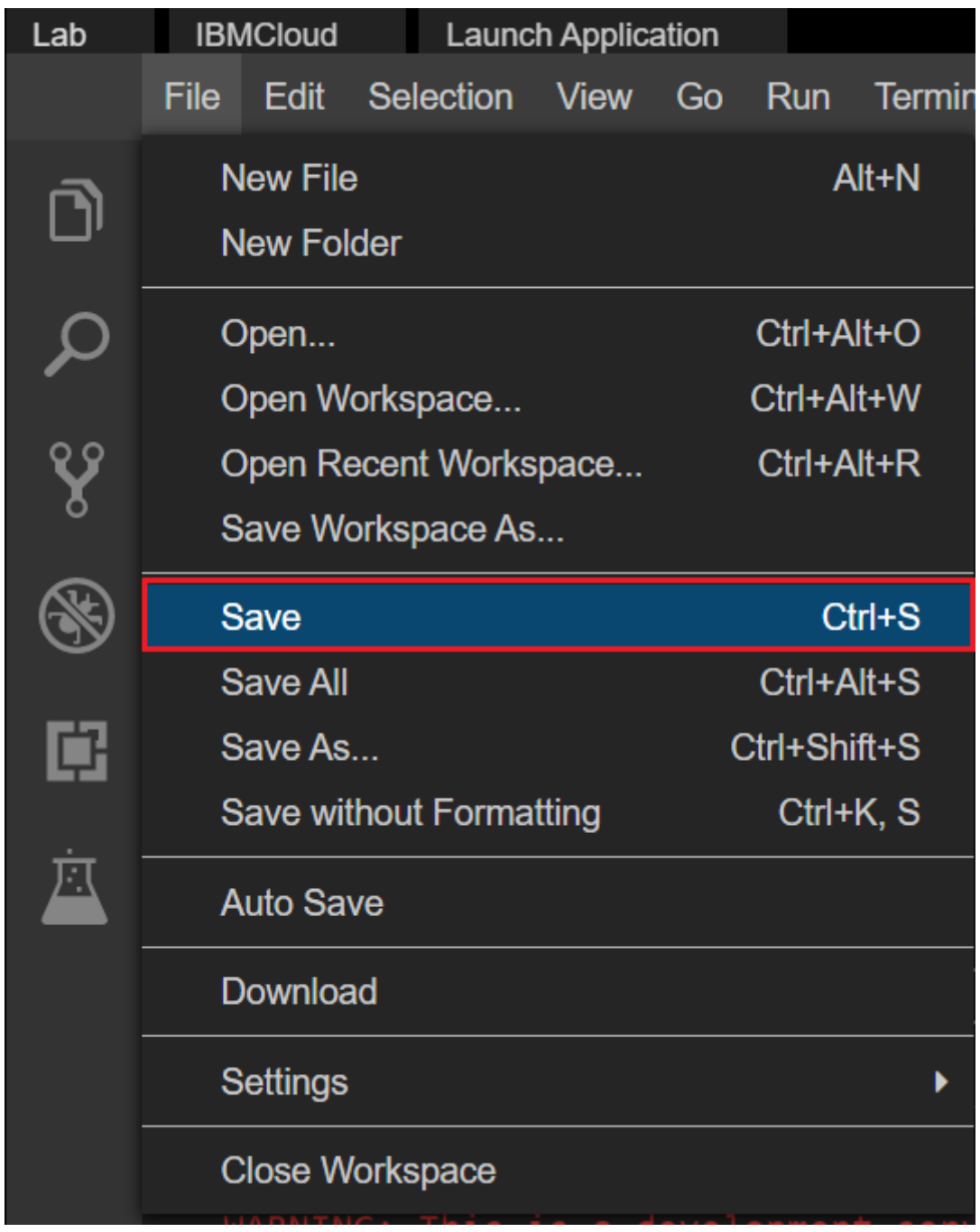
2. Provide the file name as `dash_layout.py`



3. Copy the below code to the `dash_layout.py` script and review the code.

```
# Import required packages
import pandas as pd
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
from dash.dependencies import Input, Output

# Add Dataframe

# Add a bar graph figure

app = dash.Dash()
app.layout = html.Div(children=[
    html.H1(
        children='Dashboard',
        style={
            'textAlign': 'center'
        }
    )

    # Create dropdown

    # Bar graph
])

# Run Application
if __name__ == '__main__':
    app.run_server()
```

4. Save the application using `Save` option from `File` menu.



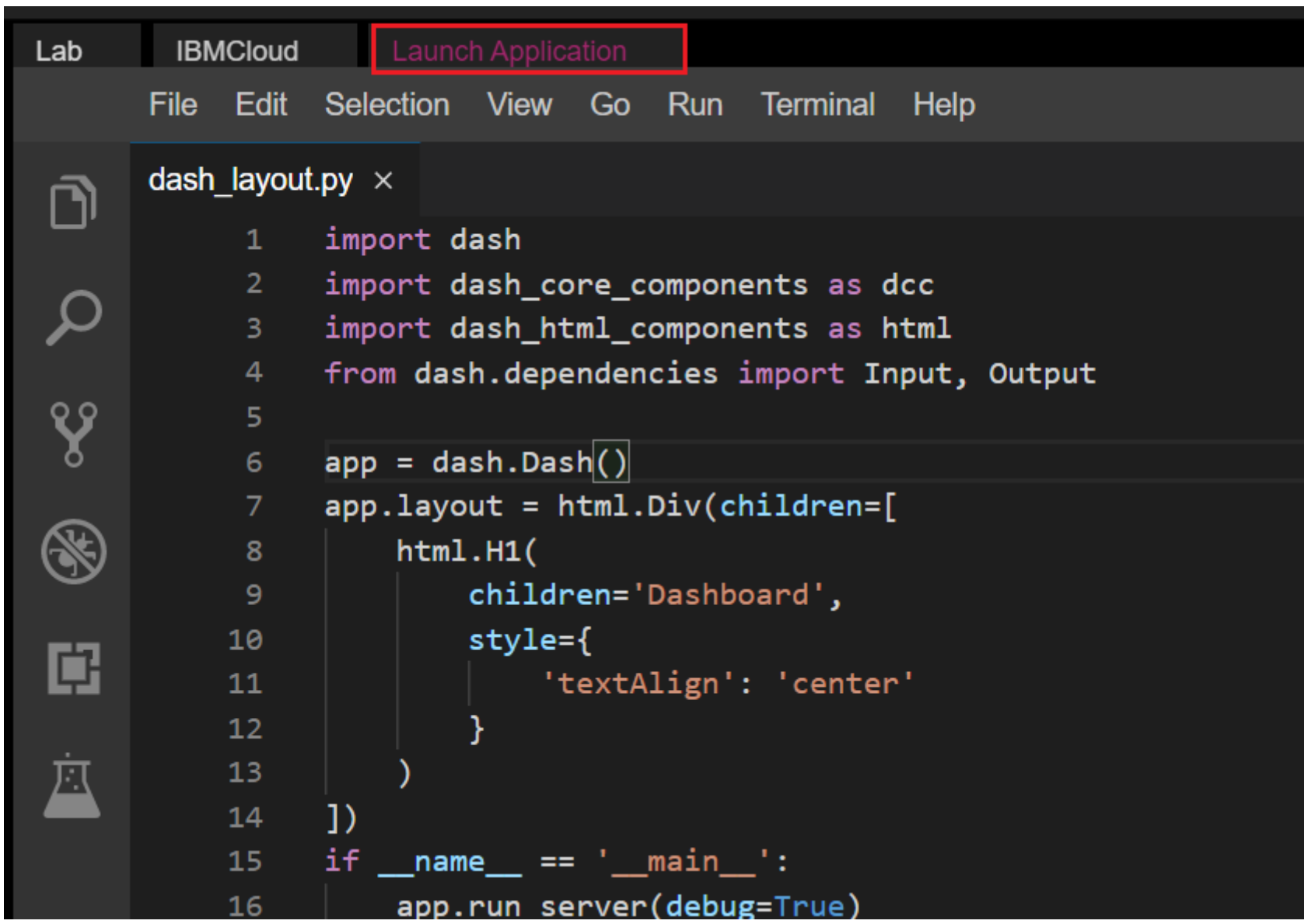
5. Run the python file using the following command in the terminal

```
python3 dash_layout.py
```

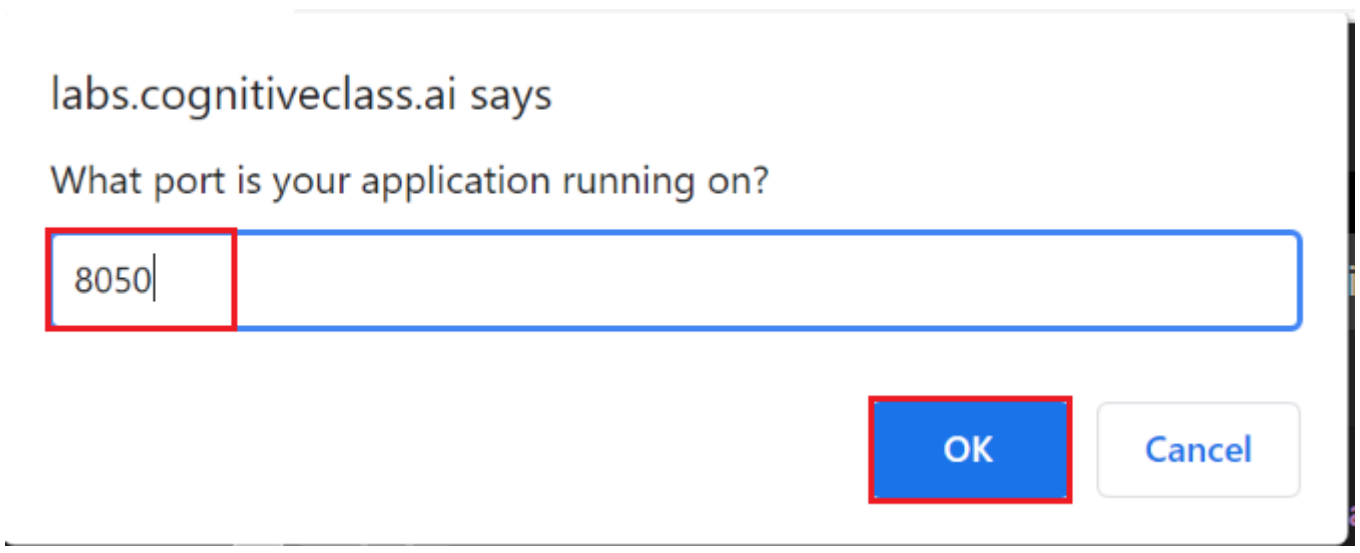
6. Observe the port number shown in the terminal.

```
theia@theia-malikas:/home/project$ python3 dash_layout.py
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'dash_layout' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

7. Click on the `Launch Application` option from the menu bar.



8. Provide the port number and click `OK`



> Note: If you are not able to see the application after launching just check the pop up window for your browser is enabled.

9. The app will open in a new browser tab like below:

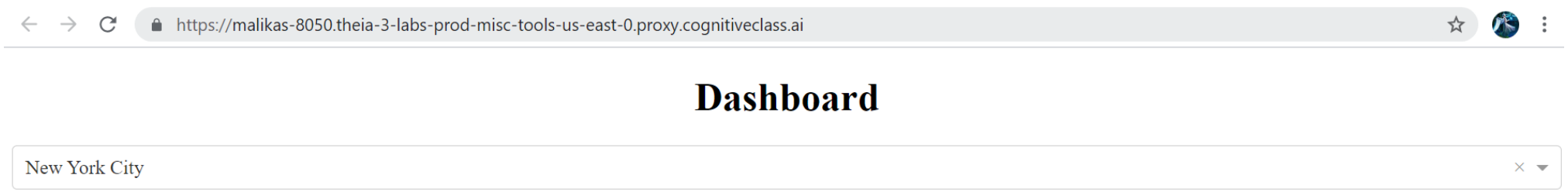
Add dropdown

1. You can generate a drop down as shown below. You do by calling `Dropdown` off `dash_core_components` and passing the options as a list of dictionaries. You can set the default value using the `value` attribute and passing in the default option.

> Note:
>
> - Add a comma (,) before the placeholder in the skeleton file and then place the code.
> - The placeholder here is "# Create dropdown " in the skeleton file.

```
# Create dropdown
dcc.Dropdown(options=[
        {'label': 'New York City', 'value': 'NYC'},
        {'label': 'Montréal', 'value': 'MTL'},
        {'label': 'San Francisco', 'value': 'SF'}
    ],
    value='NYC' # Providing a vallue to dropdown
)
```

2. After adding the dropdown the dashboard is displayed as below.



Adding a dataframe

Assume you have a dataframe as:

> Note: Place the code under the placeholder # Add Dataframe in the skeleton file copied before.

```
# Add Dataframe
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "NYC", "MTL", "NYC"]
})
```

Task 2: Create Bar graph

The `plotly.express` module (usually imported as px) contains functions that can create entire figures at once, and is referred to as `Plotly Express or PX`. Plotly Express is a built-in part of the plotly library, and is the recommended starting point for creating most common figures

In order to create a graph on our layout, we use the Graph class from dash_core_components.

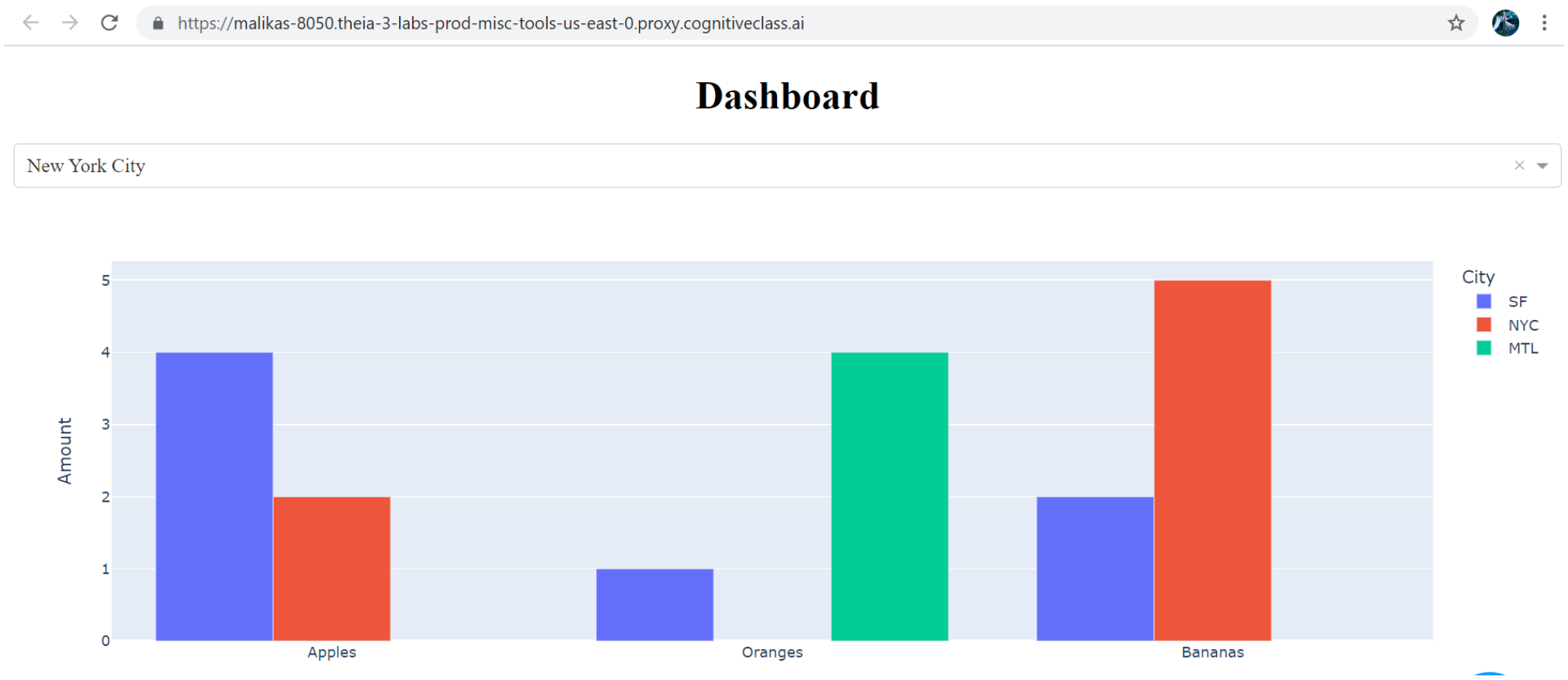
> Note: Place the code under the placeholder # Add a bar graph figure in the skeleton file copied before.

```
# Add a bar graph figure

fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")
```

> Note: Place the code under the placeholder # Bar graph figure in the skeleton file copied before and also add a comma , before the placeholder.

```
# Bar graph
dcc.Graph(id='example-graph-2',figure=fig)
```

The dashboard with the dropdown and the bar graph is displayed as below.



> Note: Here we are just creating the dropdown and bar chart without any functionality. Let's start with the real dataset to get the dropdown functionality with the graph. When you finish running the application press thekey `Ctrl+C` near the terminal window

to stop the running application and begin with the new application.

For complete code click HERE.

Task 3: Practice Exercise

Story:

Here we are looking into an **automobile dataset** which has various attributes like **drive-wheels,body-style and price**.

Lets view the snapshot of our selected dataset.

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	13495.0
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	16500.0
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154.0	5000.0	19	26	16500.0
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102.0	5500.0	24	30	13950.0
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115.0	5500.0	18	22	17450.0

Here let's say we are selecting 3 important features **drive-wheels, body-style and Price**.

- The possible values of drive-wheels are **4 wheel Drive(4wd),Front WheelDrive(fwd) and Rear wheel Drive(rwd)**.
- The different body styles of the cars are **hardtop,sedan,convertible** and so on.

- There are 2 types of people here:
    - A customer who wants to purchase the cars with less price , different body styles and wants to look for the drive wheel with this arrangement.
    - A dealer who wants to showcase the prices for the cars with different body styles and drive wheels.

- As a data analyst, you have been given a task to visually show the **body-style and prices** with respect to each **drive wheel** selected.
- So ideally you want to showcase this in the form of 2 interactive charts such as **pie chart** and **bar chart** on selection of drive wheel.

Below is the key item,

- Drive wheels

Components of the item

1. Drive Wheel Type

   For the chosen Drive wheel,

    - Pie Chart showing body style and price.
    - Bar Chart showing body style and price.

Expected Layout

Requirements to create the expected result

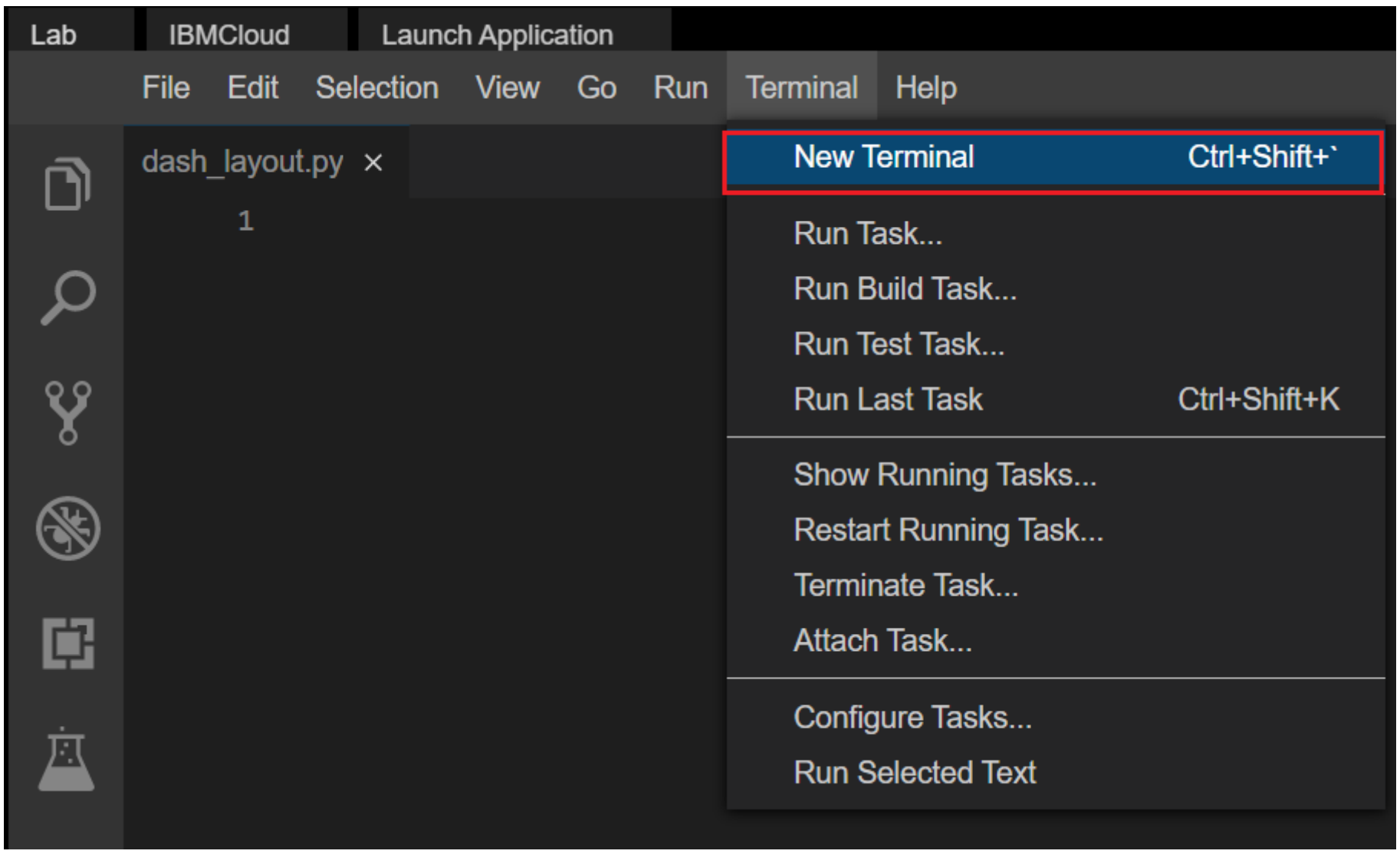
- A dropdown menu: For choosing Drive wheel type
- The layout will be designed as follows:
- An outer division with two inner divisions (as shown in the expected layout)
- One of the inner divisions will have information about the dropdown(which is the input) and the other one is for adding graphs(the 2 output graphs).
- Callback function to compute data, create graph and return to the layout.

To do:

1. Import required libraries and read the dataset
2. Create an application layout
3. Add title to the dashboard using HTML H1 component
4. Add a dropdown using dcc.dropdown
5. Add the pie chart and bar chart core graph components.
6. Run the app

Get the tool ready

- Open a new terminal, by clicking on the menu bar and selecting **Terminal**->**New Terminal**, as in the image below.



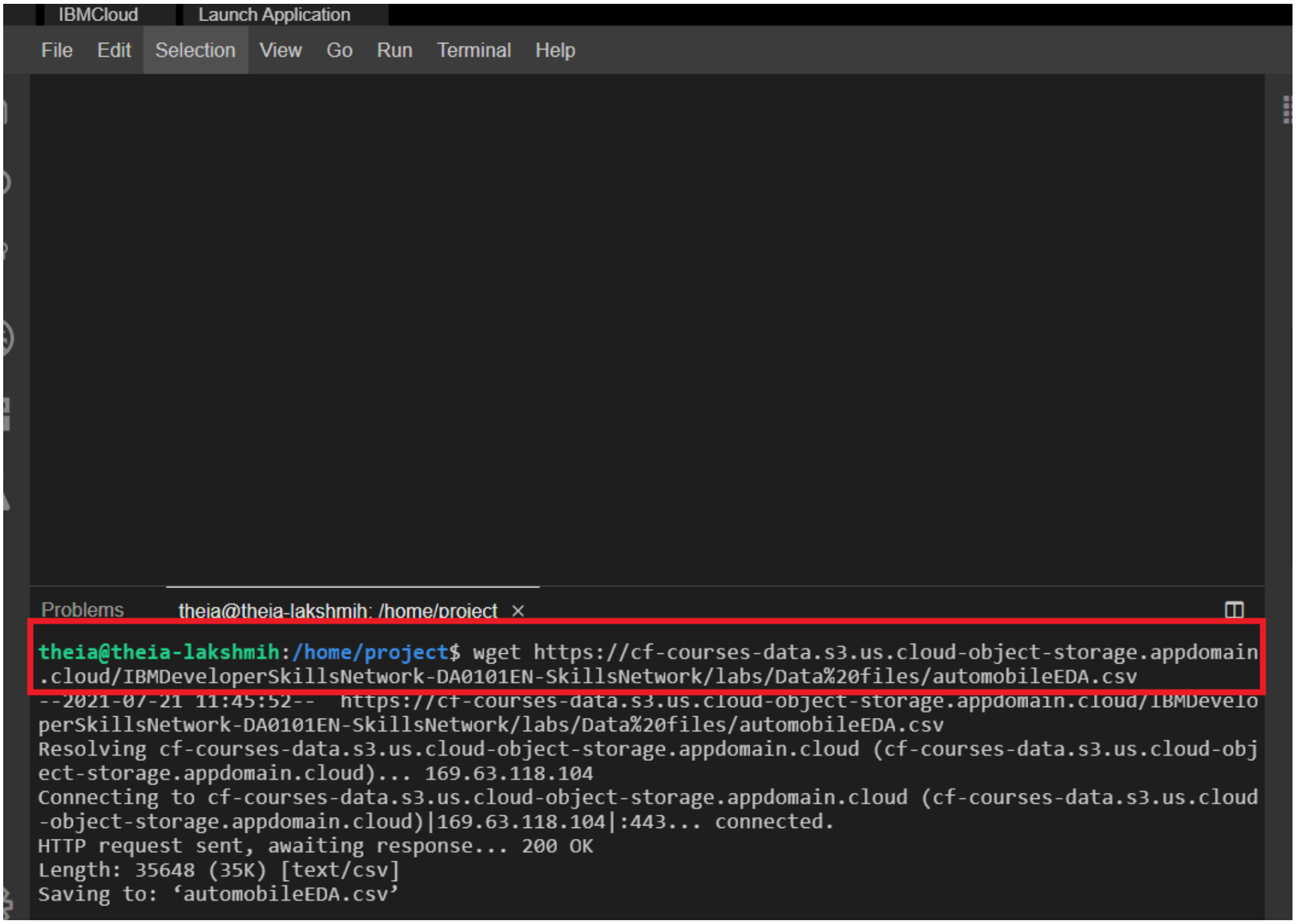
- Now, you have a terminal ready to start the lab.

Get the application skeleton

- Copy and paste the command in the terminal to download the csv.

```
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/automobileEDA.csv
```

The csv gets downloaded.



You can use this as a base code to complete the task below.

Let's create the application

- Create a new file called `Dash_Auto.py`
- Copy the code mentioned in the skeleton file and save it.

Structure of the skeleton file

```
import pandas as pd
import dash
import dash_html_components as html
import dash_core_components as dcc
from dash.dependencies import Input, Output, State
import plotly.graph_objects as go
import plotly.express as px
from dash import no_update

app = dash.Dash(__name__)

# REVIEW1: Clear the layout and do not display exception till callback gets executed
app.config.suppress_callback_exceptions = True

# Read the automobiles data into pandas dataframe
auto_data =  pd.read_csv('automobileEDA.csv',
                         encoding = "ISO-8859-1",
                         )

#Layout Section of Dash

app.layout = html.Div(children=[#TASK 3A

     #outer division starts
     html.Div([
                  # First inner divsion for  adding dropdown helper text for Selected Drive wheels
                   html.Div(
                           #TASK 3B
                    ),


                   #TASK 3C

                   #Second Inner division for adding 2 inner divisions for 2 output graphs
                   html.Div([

                       #TASK 3D

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


    ])
    #outer division ends

])
#layout ends

#Place to add @app.callback Decorator
#TASK 3E


#Place to define the callback function .
#TASK 3F



if __name__ == '__main__':
    app.run_server()
```

Hints to complete TASKS

Search/Look for `TASK` word in the script to identify places where you need to complete the code.

TASK 3A: Add title to the dashboard

Update the `html.H1()` tag to hold the application title.

- Application title is `Car Automobile Components`
- Use style parameter provided below to make the title `center` aligned, with color code `#503D36`, and font-size as `24`

```
html.H1('Car Automobile Components',
                              style={'textAlign': 'center', 'color': '#503D36',
                              'font-size': 24}),
```

After updating the `html.H1()` with the application title, the `app.layout` will look like:

```
html.H1('Car Automobile Components',
                              style={'textAlign': 'center', 'color': '#503D36',
                              'font-size': 24}),
```

Reference Links: H1 component

Dash HTML Components

TASK 3B: Add a Label to the dropdown

- Use the `html.H2()` tag to hold the label for the dropdown inside the first inner division
  - Label is `Drive Wheels Type:`
  - Use style parameter provided below to allign the label `margin-right` with value `2em` which means 2 times the size of the current font.

```
html.H2('Drive Wheels Type:', style={'margin-right': '2em'}),
```

After updating the label the `app.layout` will now look like this

```
html.Div(
        [
            html.H2('Drive Wheels Type:', style={'margin-right': '2em'}),
           ]
 ),
```

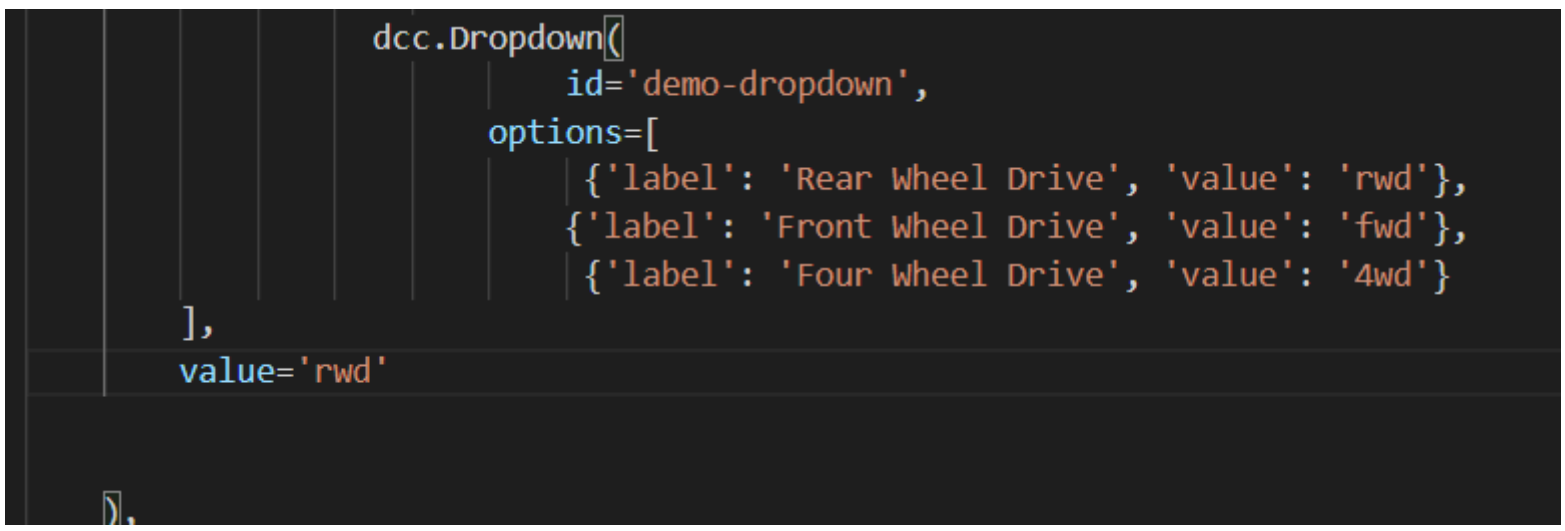
TASK 3C: Next lets add the dropdown right below the first inner division.

- The dropdown has an `id` as `demo-dropdown`.
- These options have the labels as `Rear Wheel Drive` ,`Front Wheel Drive` and `Four Wheel Drive`
- The values allowed in the dropdown are `rwd`,`fwd`,`4wd`
- The default value when the dropdown is displayed is `rwd`.

```
dcc.Dropdown(
                        id='demo-dropdown',
                    options=[
                         {'label': 'Rear Wheel Drive', 'value': 'rwd'},
                        {'label': 'Front Wheel Drive', 'value': 'fwd'},
                         {'label': 'Four Wheel Drive', 'value': '4wd'}
      ],
      value='rwd'
      ),
```

Reference link

Once you add the dropdown the '`app.layout` will appear as follows

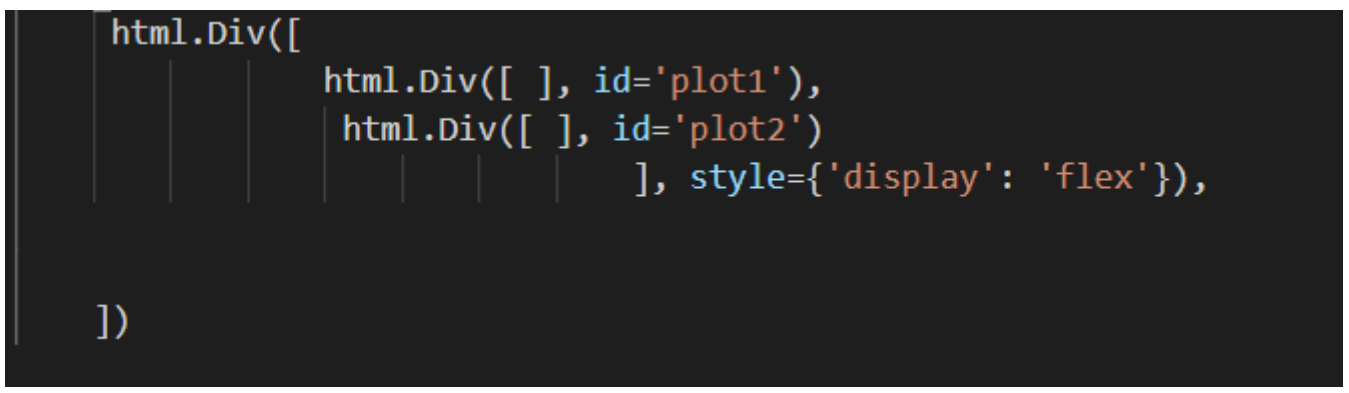


TASK 3D: Add two empty divisions for output inside the next inner division .

- Use 2 `html.Div()` tags .
- Provide division ids as `plot1` and `plot2`.

```
html.Div([ ], id='plot1'),
html.Div([ ], id='plot2')
```

Once you add the divisions the '`app.layout` will appear as follows



TASK 3E: Add the Ouput and input components inside the app.callback decorator.

- The `inputs` and `outputs` of our application's interface are described declaratively as the arguments of `@app.callback` decorator.

-In Dash, the `inputs` and `outputs` of our application are simply the properties of a particular component.

- In this example, our input is the `value` property of the component that has the ID `demo-dropdown`
- Our layout has 2 outputs so we need to create 2 output components.

It is a list with 2 output parameters with component id and property. Here, the component property will be `children` as we have created empty division and passing in `dcc.Graph` (figure) after computation.

Component ids will be `plot1` , `plot2`.

```
@app.callback([Output(component_id='plot1', component_property='children'),
               Output(component_id='plot2', component_property='children')],
               Input(component_id='demo-dropdown', component_property='value'))
```

Once you add the callback decorator the '`app.layout` will appear as follows

```
53
54  @app.callback([Output(component_id='plot1', component_property='children'),
55                 Output(component_id='plot2', component_property='children')
56  ],
57                Input(component_id='demo-dropdown', component_property='value'))
```

TASK 3F: Add the callback function.

- Whenever an input property changes, the function that the callback decorator wraps will get called automatically.
- In this case let us define a function `display_selected_drive_charts()` which will be wrapped by our decorator.
- The function first filters our dataframe `auto_data` by the selected value of the drive-wheels from the dropdown as follows
- `auto_data[auto_data['drive-wheels']==value]` .
- Next we will group by the `drive-wheels` and `body-style` and calculate the mean `price` of the dataframe.
- Use the `px.pie()` and `px.bar()` function we will plot the pie chart and bar chart

```
def display_selected_drive_charts(value):



   filtered_df = auto_data[auto_data['drive-wheels']==value].groupby(['drive-wheels','body-style'],as_index=False). \
            mean()

   filtered_df = filtered_df

   fig1 = px.pie(filtered_df, values='price', names='body-style', title="Pie Chart")
   fig2 = px.bar(filtered_df, x='body-style', y='price', title='Bar Chart')

   return [dcc.Graph(figure=fig1),
            dcc.Graph(figure=fig2) ]
```

- Here for the pie chart we pass the `filtered dataframe` where `values` correspond to `price` and names will be `body-style`
- For the bar chart also we will pass the `filtered dataframe` where x-axis corresponds to `body-style` and y-axis as `price`.
- Finally we return the 2 figure objects `fig1` and `fig2` in `dcc.Graph` method and finally the plots are displayed as follows

```
def display_selected_drive(value):



   filtered_df = auto_data[auto_data['drive-wheels']==value].groupby(['drive-wheels'
            mean()

   filtered_df = filtered_df

   fig1 = px.pie(filtered_df, values='price', names='body-style', title="Pie Chart")
   fig2 = px.bar(filtered_df, x='body-style', y='price', title='Bar Chart')

   return [dcc.Graph(figure=fig1),
            dcc.Graph(figure=fig2) ]
```

- Once you have finished coding save your code.

Run the Application

- Firstly, install pandas and dash using the following command

```
pip3 install pandas dash
```

```
theia@theia-lakshmih:/home/project$ pip3 install pandas dash
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15: CryptographyDeprecationWarning: int_from
_bytes is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19: CryptographyDeprecationWarning: int_from_byt
es is deprecated, use int.from_bytes instead
  from cryptography.utils import int_from_bytes
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /home/theia/.local/lib/python3.6/site-packages (1.1.5)
Collecting dash
  Downloading dash-1.21.0.tar.gz (1.1 MB)
```

- Next Run the python file using the command

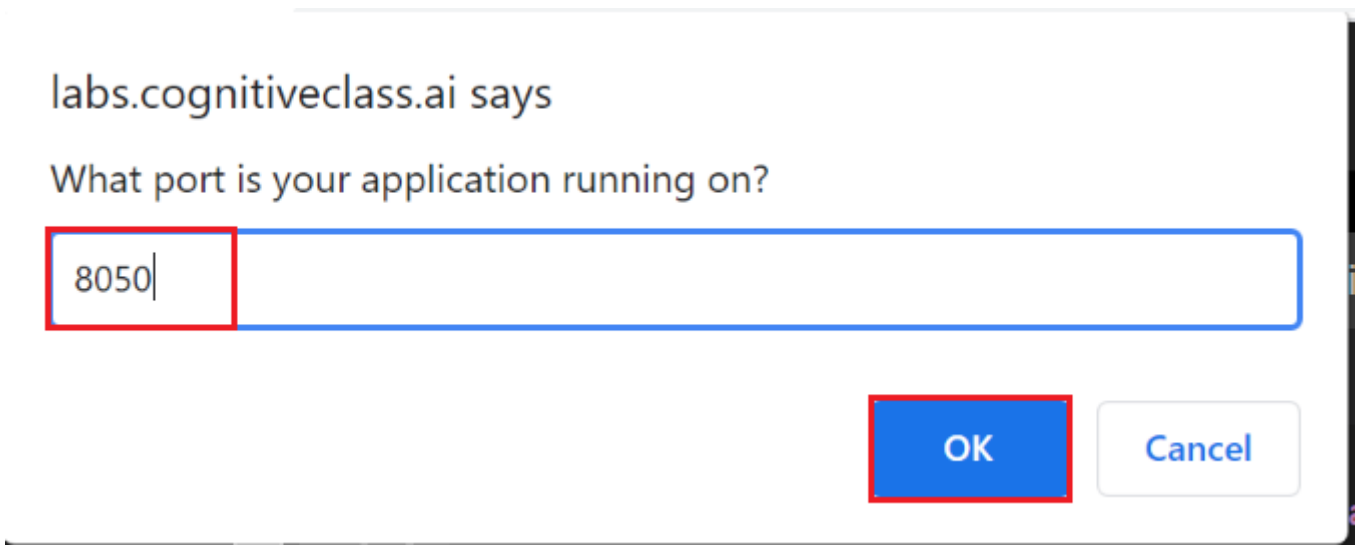
```
python3 Dash_Auto.py
```

- Observe the port number shown in the terminal.

```
theia@theia-lakshmih:/home/project$ python3 Dash_Auto.py
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "Dash_Auto" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

- Click on the `Launch Application` option from the menu bar.

- Provide the port number and click `OK`



- The graphs appear on selection of drive wheels.
- For complete code click HERE.

Congratulations, you have successfully created dash application!

Author

Malika Singla

Lakshmi Holla

Changelog

Date	Version	Changed by	Change Description
2021-07-21	0.1	Lakshmi Holla, Malika Singla	Initial Version

© IBM Corporation 2021. All rights reserved.