

Student name: Hoang Long Tran

Student ID: s223128143

SIT225: Data Capture Technologies

Activity 6.1: Plotly data dashboard

Plotly Dash apps give a point-&-click interface to models written in Python, vastly expanding the notion of what's possible in a traditional "dashboard". With Dash apps, data scientists and engineers put complex Python analytics in the hands of business decision-makers and operators. In this activity, you will learn basic building blocks of Plotly to create Dash apps.

Hardware Required

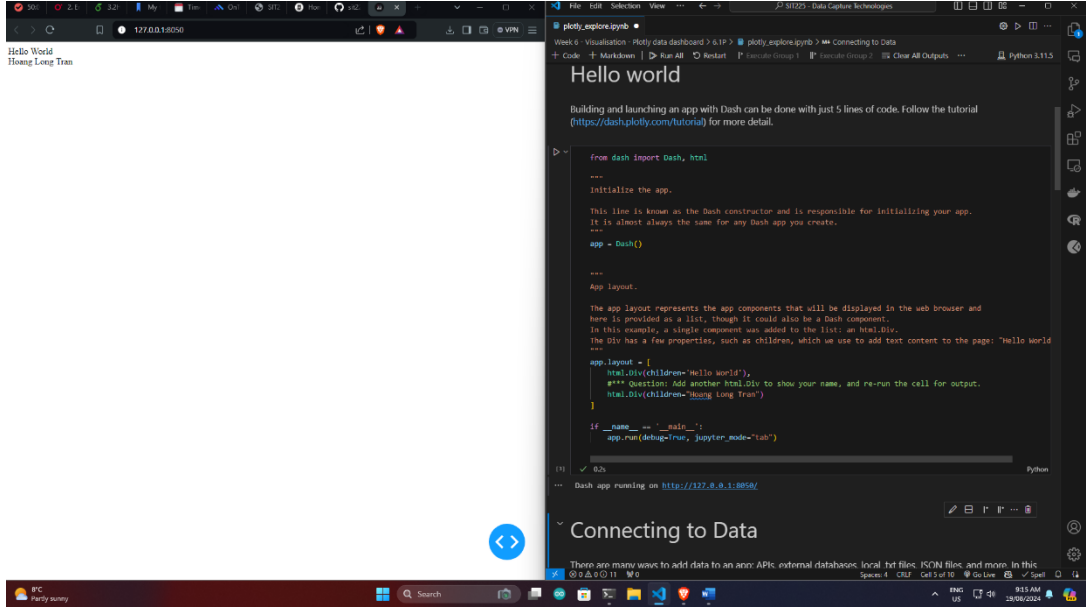
No hardware is required.

Software Required

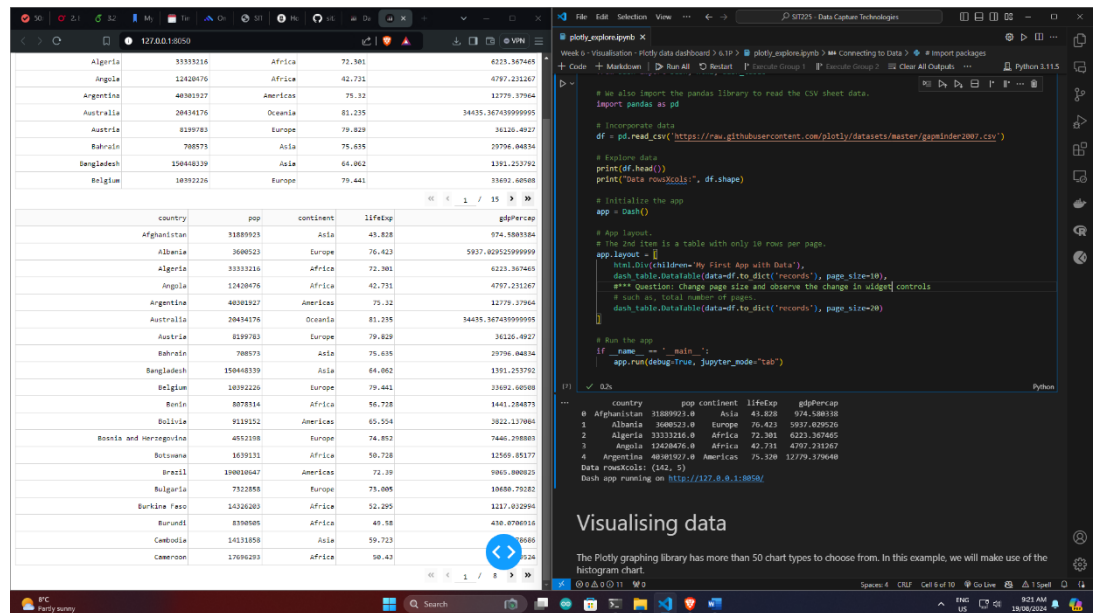
Plotly library and Dash module
Python 3

Steps

Step	Action
1	<p>Install Plotly and dash using the command below in the command line.</p> <pre>\$ pip install plotly dash</pre> <p>You can download Jupyter Notebook from here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_6/plotly_explore.ipynb) and run all the cells. The Notebook contains multiple sections such as Hello World which follows a sample code in a following cell. If you run the Hello world cell it will show Plotly Dash web page. The cell also includes a Question (**** Question) which you will need to carry out to get a modified output. You will need to capture the output and share the screenshot in the following steps.</p>

2	<p>Question: Hello World cell has a question - add another html.Div to show your name, and re-run the cell for output. You will need to update the code, run the cell, capture the screenshot of the output and paste it here.</p> <p>Answer:</p> 
3	<p>Question: Connecting to Data cell has a question - change page size and observe the change in widget controls such as, total number of pages. You will need to update the code, run the cell, capture the screenshot of the output and paste it here.</p>

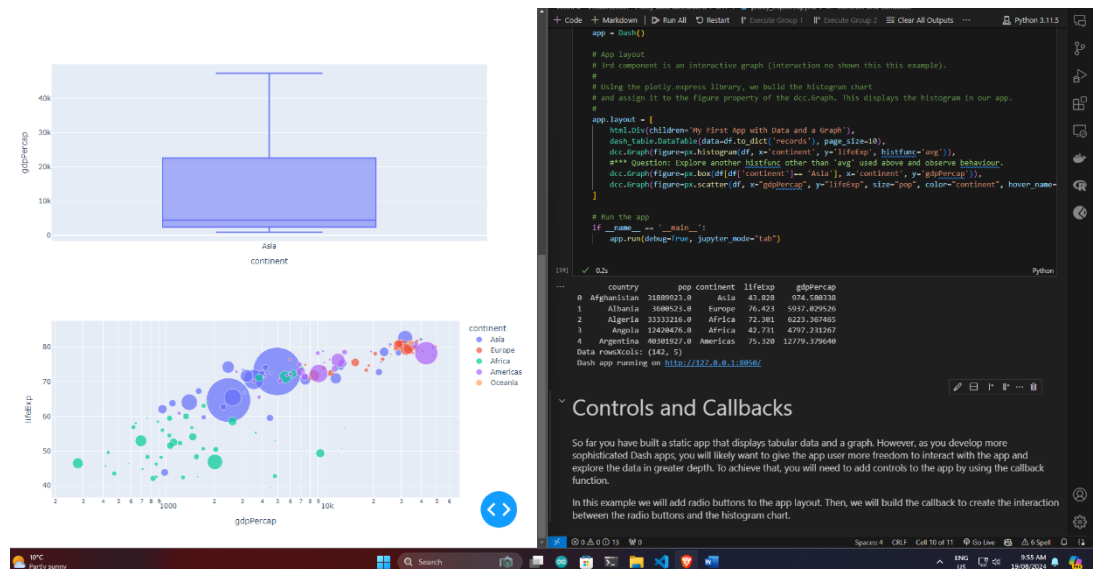
Answer:



4

Question: Visualising data cell has a question - explore another histfunc other than 'avg' used above and observe behaviour. You will need to update the code, run the cell, capture the screenshot of the output and paste it here.

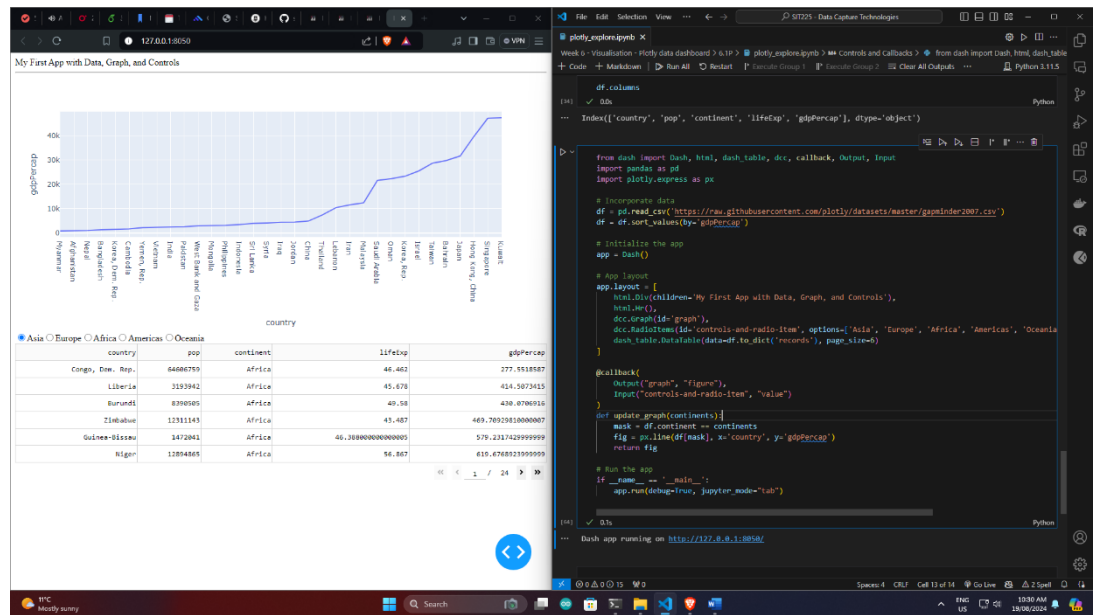
Answer:



5

Question: Controls and Callbacks cell has a question - use line graphs instead of histogram. You will need to update the code, run the cell, capture the screenshot of the output and paste it here.

Answer:



6

Question: Now you have learned how to use Plotly Dash for visualising your data, describe how you will be using this tool for your desired sensor monitoring dashboard with a number of sensors including DHT22 or accelerometer data.

Answer: If I have enough data for DHT22, I could use a scatterplot and machine learning techniques to describe the relationship between temperature and humidity based on the season.

7

Question: Convert the Notebook to PDF and merge with this activity sheet PDF. You will need this merged PDF to combine with this week's OnTrack task for submission.

Answer:

Hello world

```
# Fill in student ID and name
#
student_id = "Hoang Long Tran"
student_first_last_name = "s223128143"
print(student_id, student_first_last_name)
```

Hoang Long Tran s223128143

```
# install plotly and dash, if not yet already
# ! pip install plotly dash

import plotly, dash
print(plotly.__version__)
print(dash.__version__)
```

5.22.0
2.17.1

Building and launching an app with Dash can be done with just 5 lines of code. Follow the tutorial (<https://dash.plotly.com/tutorial>) for more detail.

```
from dash import Dash, html

"""
Initialize the app.

This line is known as the Dash constructor and is responsible for initializing your app.
It is almost always the same for any Dash app you create.
"""
app = Dash()
```

6.1P. Plotly data dashboard

Q2.

```
from dash import Dash, html, dash_table, dcc, callback, Output, Input, State
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Initialize the app
```

```

app = Dash()

# App layout
app.layout = html.Div([
    html.H1(children='Gyroscope data'),
    html.Hr(),

    # Drop down for graphs
    html.Label("Select Graph Type:"),
    dcc.Dropdown(
        id='graph-type',
        options=[
            {'label': 'Scatter Plot', 'value': 'scatter'},
            {'label': 'Line Chart', 'value': 'line'},
            {'label': 'Distribution Plot', 'value': 'distribution'}],
        value = 'scatter'
    ),

    # Data Table
    dash_table.DataTable(data=df1.to_dict('records'), page_size=6),

    # Drop down for rotation selection
    dcc.Dropdown(id='rotation-type',
        options=[ {'label': 'x', 'value': 'x'},
                   {'label': 'y', 'value': 'y'},
                   {'label': 'z', 'value': 'z'} ],
        value=['x', 'y', 'z'], multi=True),

    # Input for the number of samples
    html.Label("Number of Samples: "),
    dcc.Input(id='num-samples', type='number', value=1020, min=100, step=20),

    # Container for the graph and summary table
    html.Div([
        dcc.Graph(figure={}, id='graph', style={'display': 'inline-block',
'width': '49%'}),
        dcc.Graph(id='data-summary', style={'display': 'inline-block', 'width':
'49%', 'vertical-align': 'top'}),
    ], style={'display': 'flex', 'flex-wrap': 'wrap'})
])

@callback(
    Output("graph", "figure"),
    Output('data-summary', 'figure'),

```

```

    Input("rotation-type", "value"),
    Input('graph-type', 'value'),
    Input('num-samples', 'value'),
)
def update(rotation, graph_type, num_sample):
    df2 = df1.head(num_sample)

    # Create plot based on selected graph
    fig = {}
    if graph_type == 'scatter':
        fig = px.scatter(df2, x='Timestamp', y=rotation)
    elif graph_type == 'line':
        fig = px.line(df2, x='Timestamp', y=rotation)
    elif graph_type == 'distribution':
        fig = px.histogram(df2, x=rotation)

    # Data summary table
    try:
        df3 = df2[rotation]
        summary = df3.describe().reset_index()
        summary_table = go.Figure(data=[go.Table(
            header=dict(values=list(summary.columns),
                        fill_color='paleturquoise',
                        align='left'),
            cells=dict(values=[summary[col] for col in summary.columns],
                      fill_color='lavender',
                      align='left')

        )])
    except Exception as e:
        summary_table = go.Figure(data=[go.Table(
            header=dict(values=["No data selected"],
                        fill_color='paleturquoise',
                        align='left'),
            cells=dict(values=[["Please select at least one rotation axis"]],
                      fill_color='lavender',
                      align='left')

        )])

    return fig, summary_table

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")

```

I have added some comments but here is a brief explanation. First, I initialize the app layout, then add a heading. Below the main heading is a drop-down menu of plots, with the default one being scatter plot. Below is the data table containing all the data in the data frame. Then it's the drop-down menu to select rotations, the default is all rotation is selected. There are options to choose the number of samples, default value is 1020 (all of the sample available), the step is 20 samples, if we go over 1020 then nothing changes because 1020 is the maximum of data, if we go below 1020, then the graph and summary statistics table will change because there is less data. If no rotation is selected, then the summary table notifies the user to select at least one rotation.

Q3.

<https://www.youtube.com/watch?v=v0ceW-48gig>

Q4.

https://github.com/tomadonna1/SIT225_2024T2/tree/main/Pass%20Task%20Plotly%20data%20dashboard

Hello world

```
# Fill in student ID and name
#
student_id = "Hoang Long Tran"
student_first_last_name = "s223128143"
print(student_id, student_first_last_name)
```

Hoang Long Tran s223128143

```
# install plotly and dash, if not yet already
# ! pip install plotly dash

import plotly, dash
print(plotly.__version__)
print(dash.__version__)
```

5.22.0

2.17.1

Building and launching an app with Dash can be done with just 5 lines of code. Follow the tutorial (<https://dash.plotly.com/tutorial>) for more detail.

```
from dash import Dash, html

"""
Initialize the app.

This line is known as the Dash constructor and is responsible for initializing your app.
It is almost always the same for any Dash app you create.
"""
app = Dash()
```

```
"""
```

```
App layout.
```

The app layout represents the app components that will be displayed in the web browser and here is provided as a list, though it could also be a Dash component.

In this example, a single component was added to the list: an `html.Div`.

The Div has a few properties, such as `children`, which we use to add text content to the page

```
"""
```

```
app.layout = [
```

```
    html.Div(children='Hello World'),
```

```
    *** Question: Add another html.Div to show your name, and re-run the cell for output.
```

```
    html.Div(children="Hoang Long Tran")
```

```
]
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True, jupyter_mode="tab")
```

Dash app running on <http://127.0.0.1:8050/>

<IPython.core.display.Javascript object>

Connecting to Data

There are many ways to add data to an app: APIs, external databases, local .txt files, JSON files, and more. In this example, we will highlight one of the most common ways of incorporating data from a CSV sheet.

```
# Import packages
```

```
# We import the dash_table module to display the data inside a Dash DataTable.  
from dash import Dash, html, dash_table
```

```
# We also import the pandas library to read the CSV sheet data.  
import pandas as pd
```

```
# Incorporate data
```

```
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')
```

```

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout.
# The 2nd item is a table with only 10 rows per page.
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    *** Question: Change page size and observe the change in widget controls
    # such as, total number of pages.
    dash_table.DataTable(data=df.to_dict('records'), page_size=20)
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")

```

	country	pop	continent	lifeExp	gdpPercap
0	Afghanistan	31889923.0	Asia	43.828	974.580338
1	Albania	3600523.0	Europe	76.423	5937.029526
2	Algeria	33333216.0	Africa	72.301	6223.367465
3	Angola	12420476.0	Africa	42.731	4797.231267
4	Argentina	40301927.0	Americas	75.320	12779.379640

Data rowsXcols: (142, 5)

Dash app running on http://127.0.0.1:8050/

<IPython.core.display.Javascript object>

Visualising data

The Plotly graphing library has more than 50 chart types to choose from. In this example, we will make use of the histogram chart.

```

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')
df.head()

```

	country	pop	continent	lifeExp	gdpPercap
0	Afghanistan	31889923.0	Asia	43.828	974.580338
1	Albania	3600523.0	Europe	76.423	5937.029526
2	Algeria	33333216.0	Africa	72.301	6223.367465
3	Angola	12420476.0	Africa	42.731	4797.231267
4	Argentina	40301927.0	Americas	75.320	12779.379640

```
# Import packages

# We import the dcc module (DCC stands for Dash Core Components).
# This module includes a Graph component called dcc.Graph, which is used to render interactive plots.
from dash import Dash, html, dash_table, dcc

# We also import the plotly.express library to build the interactive graphs.
import plotly.express as px

import pandas as pd

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout
# 3rd component is an interactive graph (interaction not shown in this example).
#
# Using the plotly.express library, we build the histogram chart
# and assign it to the figure property of the dcc.Graph. This displays the histogram in our app.
#
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg')),
    """*** Question: Explore another histfunc other than 'avg' used above and observe behaviour ***"""
    dcc.Graph(figure=px.box(df[df['continent']=='Asia'], x='continent', y='gdpPercap')),
    dcc.Graph(figure=px.scatter(df, x="gdpPercap", y="lifeExp", size="pop", color="continent"))
]
```

```
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")
```

```

      country      pop continent  lifeExp    gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1    Albania    3600523.0     Europe   76.423   5937.029526
2    Algeria   33333216.0     Africa   72.301   6223.367465
3    Angola    12420476.0     Africa   42.731   4797.231267
4  Argentina   40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8050/
```

```
<IPython.core.display.Javascript object>
```

Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

```
# Import packages

# We import dcc like we did in the previous section to use dcc.Graph.
# In this example, we need dcc for dcc.Graph as well as the radio buttons component, dcc.Radi
#
# To work with the callback in a Dash app, we import the callback module and the two arguments
# commonly used within the callback: Output and Input.
#
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
```

```

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp', id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
    *** Question: Use line graphs instead of histogram.
]

#
# Notice that we add that RadioItems component to the layout, directly above the DataTable.
# There are three options, one for every radio button.
# The lifeExp option is assigned to the value property, making it the currently selected value.
#
# Both the RadioItems and the Graph components were given id names: these will be used by
# the callback to identify the components.
#
# The inputs and outputs of our app are the properties of a particular component.
# In this example, our input is the value property of the component that has the ID "controls-and-radio-item".
# If you look back at the layout, you will see that this is currently lifeExp.
# Our output is the figure property of the component with the ID "controls-and-graph", which
# is currently an empty dictionary (empty graph).
#
# The callback function's argument col_chosen refers to the component property of the input component.
# We build the histogram chart inside the callback function, assigning the chosen radio item to the
# y-axis attribute of the histogram. This means that every time the user selects a new radio item,
# the figure is rebuilt and the y-axis of the figure is updated.
#
# Finally, we return the histogram at the end of the function. This assigns the histogram to the
# figure property of the dcc.Graph, thus displaying the figure in the app.
#

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
)

```

```

)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")

```

Dash app running on <http://127.0.0.1:8050/>

<IPython.core.display.Javascript object>

```
df.columns
```

```
Index(['country', 'pop', 'continent', 'lifeExp', 'gdpPercap'], dtype='object')
```

```

from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')
df = df.sort_values(by='gdpPercap')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.Graph(id='graph'),
    dcc.RadioItems(id='controls-and-radio-item', options=['Asia', 'Europe', 'Africa', 'America'], value='Asia'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6)
]

@callback(
    Output("graph", "figure"),
    Input("controls-and-radio-item", "value")
)

```

```
)  
def update_graph(continents):  
    mask = df.continent == continents  
    fig = px.line(df[mask], x='country', y='gdpPercap')  
    return fig  
  
# Run the app  
if __name__ == '__main__':  
    app.run(debug=True, jupyter_mode="tab")
```

Dash app running on <http://127.0.0.1:8050/>

<IPython.core.display.Javascript object>