Plotly Express in Python

- Plotly Express is a terse, consistent, high-level API for creating figures.
- 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. Every Plotly Express function uses **graph objects** (The figures created, manipulated and rendered by the plotly Python library are **represented by tree-like data structures** which are automatically serialized to **JSON** for rendering by the Plotly.js JavaScript library.) internally and returns a **plotly.graph_objects.Figure** instance.

Plotly Express currently includes the following functions:

- **Basics**: scatter, line, area, bar, funnel, timeline
- **Part-of-Whole**: pie, sunburst, treemap, icicle, funnel_area
- **1D Distributions**: histogram, box, violin, strip, ecdf
- **2D Distributions**: density_heatmap, density_contour
- **Matrix or Image Input**: imshow
- **3-Dimensional**: scatter_3d, line_3d
- **Multidimensional**: scatter_matrix, parallel_coordinates, parallel_categories
- **Tile Maps**: scatter_mapbox, line_mapbox, choropleth_mapbox, density_mapbox
- **Outline Maps**: scatter_geo, line_geo, choropleth
- **Polar Charts**: scatter_polar, line_polar, bar_polar
- **Ternary Charts**: scatter_ternary, line_ternary

High-Level Features

- **A single entry point into plotly**: just import plotly.express as px and get access to all the plotting functions, plus built-in demo datasets under px.data and built-in color scales and sequences under px.color. Every PX function returns a plotly.graph_objects.Figure object, so you can edit it using all the same methods like update_layout and add_trace.
- **Sensible, Overridable Defaults**: PX functions will infer sensible defaults wherever possible, and will always let you override them.
- **Flexible Input Formats**: PX functions accept input in a variety of formats, from lists and dicts to long-form or wide-form DataFrames to numpy arrays and xarrays to GeoPandas GeoDataFrames.
- **Automatic Trace and Layout configuration**: PX functions will create one trace per animation frame for each unique combination of data values mapped to discrete color, symbol, line-dash, facet-row and/or facet-column. Traces' legendgroup and showlegend attributes are set such that only one legend item appears per unique combination of discrete color, symbol and/or line-dash. Traces are automatically linked to a correctly-configured subplot of the appropriate type.
- **Automatic Figure Labelling**: PX functions label axes, legends and colorbars based in the input DataFrame or xarray, and provide extra control with the labels argument.
- **Automatic Hover Labels**: PX functions populate the hover-label using the labels mentioned above, and provide extra control with the hover_name and hover_data arguments.
- **Styling Control**: PX functions read styling information from the default figure template, and support commonly-needed cosmetic controls like category_orders and color_discrete_map to precisely control categorical variables.

- **Uniform Color Handling**: PX functions automatically switch between continuous and categorical color based on the input type.
- **Faceting**: the 2D-cartesian plotting functions support row, column and wrapped facetting with `facet_row`, `facet_col` and `facet_col_wrap` arguments.
- **Marginal Plots**: the 2D-cartesian plotting functions support marginal distribution plots with the `marginal`, `marginal_x` and `marginal_y` arguments.
- **A Pandas backend**: the 2D-cartesian plotting functions are available as a Pandas plotting backend so you can call them via `df.plot()`.
- **Trendlines**: `px.scatter` supports built-in trendlines with accessible model output.
- **Animations**: many PX functions support simple animation support via the `animation_frame` and `animation_group` arguments.
- **Automatic WebGL switching**: for sufficiently large scatter plots, PX will automatically use WebGL for hardware-accelerated rendering.

For detailed information, follow the web link:

https://plotly.com/python/plotly-express/

***

Program (10a): Write a Python program to draw *Time Series* using *Plotty Libraries*.

Aim

To write a Python program to draw ***time series using Plotly Libraries***.

Procedure

The provided Python script showcases the use of the Plotly Express library to create an interactive line plot depicting the exchange rate between the US Dollar and the Indian Rupee over time.

1. **Data Import:**
   - The script uses the Pandas library to read currency conversion data from a CSV file (‘**set1.csv**’). You can download the csv file given above.
2. **Plotly Express:**
   - Plotly Express (`px`) is employed to create an interactive line plot with the exchange rate data.
3. **Line Plot:**
   - The `line` function from Plotly Express is used to generate a line plot.
   - The x-axis represents dates (‘DATE’), and the y-axis represents exchange rates (‘RATE’).
4. **Title:**
   - The plot is given a title, ‘Dollar vs Rupee,’ for context.
5. **Interactive Display:**
   - The `show` method is called on the figure (`fig`) to display the interactive plot.

Program 10.1

```
import pandas as pd
import plotly.express as px
dollar_conv = pd.read_csv('set1.csv')
fig = px.line(dollar_conv, x='DATE', y='RATE', title='Dollar vs Rupee')
fig.show()
```

Output

Program 10.2

```
import pandas as pd
import plotly.express as px

runs_scored = pd.read_csv('AustraliaVsIndia.csv')
fig = px.line(runs_scored, x='Overs', y=['AUS', 'IND'], markers=True)
fig.update_layout(title='Australia vs India ODI Match', xaxis_title='OVERS', yaxis_title='RUNS',
legend_title='Country')

fig.show()
```

Output

Program 10.3

```
#Bar Graph of Runs scored every Over
import pandas as pd
import plotly.express as px

runs_scored = pd.read_csv('AustraliaVsIndia.csv')

fig = px.bar(runs_scored, x='Overs', y=['AUS_RPO', 'IND_RPO'], barmode='group')
fig.update_layout(title='Australia vs India ODI Match', xaxis_title='OVERS', yaxis_title='RUNS',
legend_title='Country')

fig.show()
```

Output

Result: Python program was successfully executed and drawn ***time series using Plotly Libraries.***

Introducing JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

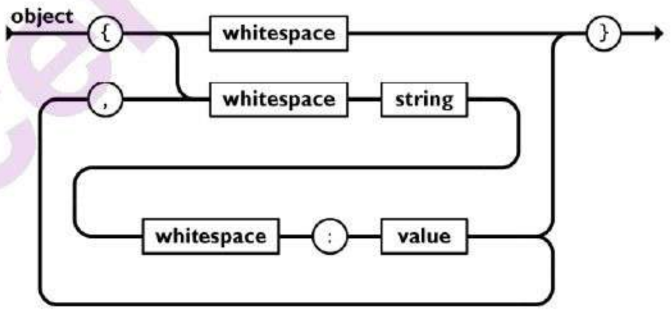
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

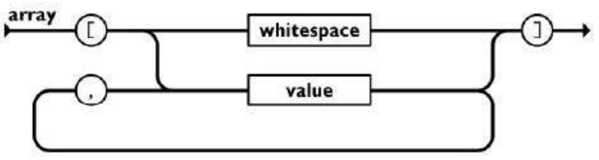
These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

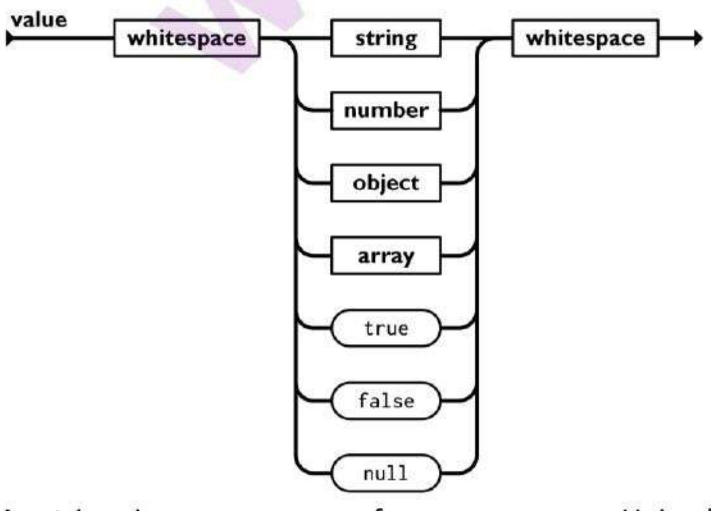
An *object* is an unordered set of name/value pairs. An object begins with `{`left brace and ends with `}`right brace. Each name is followed by `:`colon and the name/value pairs are separated by `,` comma.



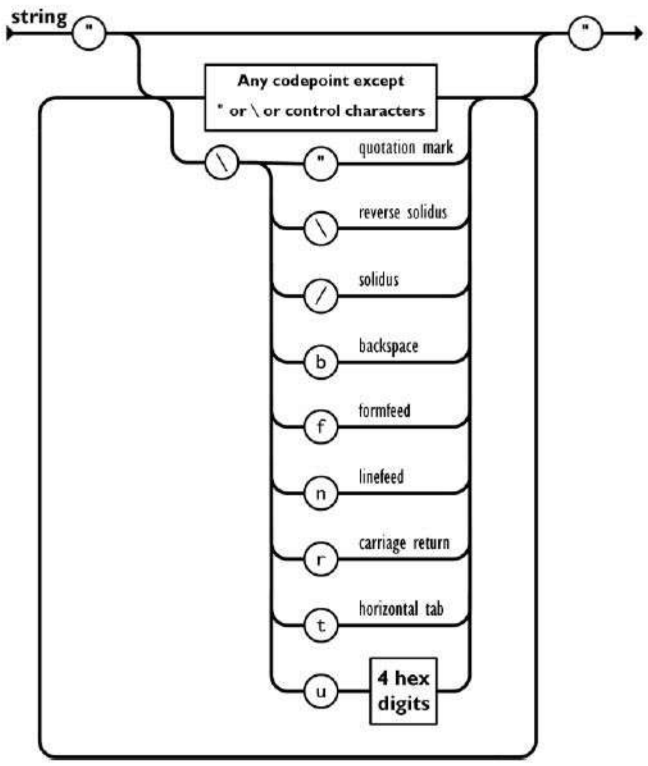
An *array* is an ordered collection of values. An array begins with `[`left bracket and ends with `]`right bracket. Values are separated by `,` comma.



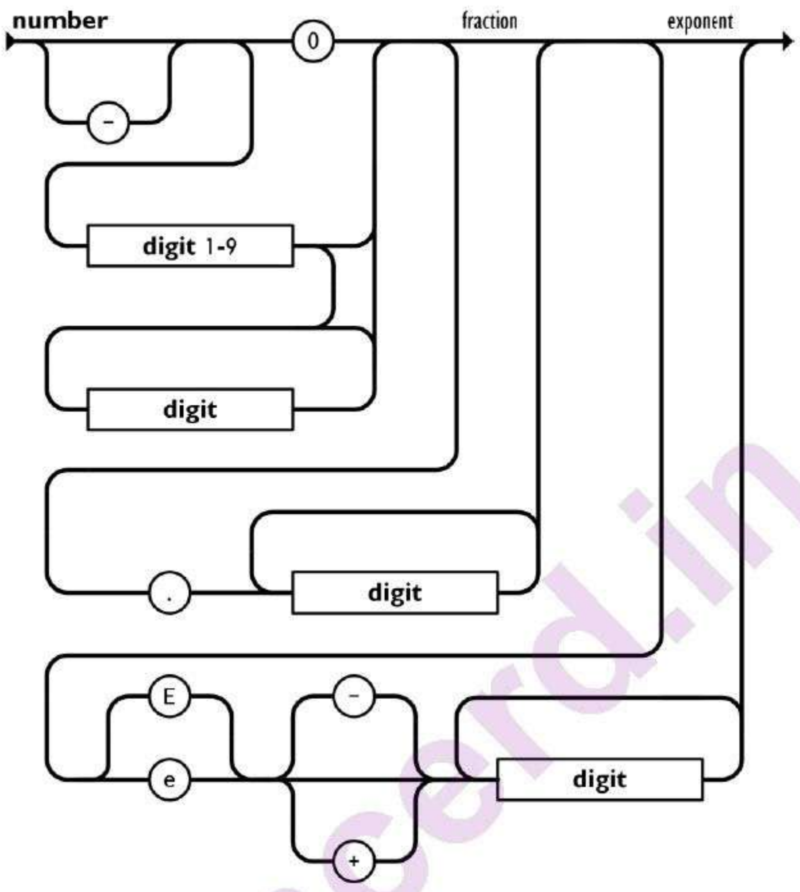
A *value* can be a *string* in double quotes, or a *number*, or `true` or `false` or `null`, or an *object* or an *array*. These structures can be nested.



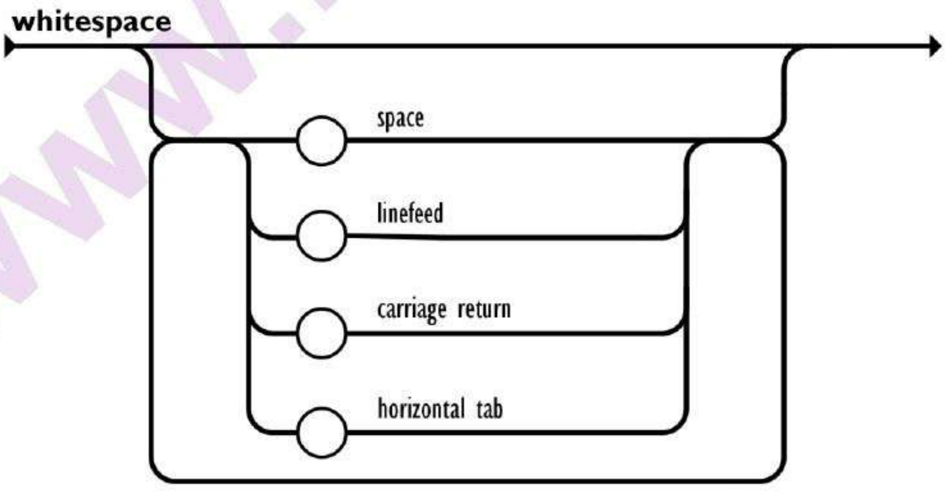
A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, which completely describes the language.



***

Choropleth Maps in Python

- A Choropleth Map is a map composed of colored polygons. It is used to represent spatial variations of a quantity. This page documents how to build **outline** choropleth maps, but you can also build choropleth **tile maps** using our Mapbox trace types.
- Below we show how to create Choropleth Maps using either Plotly Express' `px.choropleth` function or the lower-level `go.Choropleth` graph object.

Introduction: main parameters for choropleth outline maps

Making choropleth maps requires two main types of input:

1. Geometry information:
    1. This can either be a supplied GeoJSON file where each feature has either an `id` field or some identifying value in `properties`; or
    2. one of the built-in geometries within `plotly`: US states and world countries (see below)
2. A list of values indexed by feature identifier.

The GeoJSON data is passed to the `geojson` argument, and the data is passed into the `color` argument of `px.choropleth` (`z` if using `graph_objects`), in the same order as the IDs are passed into the `location` argument.

Note the `geojson` attribute can also be the URL to a GeoJSON file, which can speed up map rendering in certain cases.

Choropleth Map with plotly.express

- Plotly Express is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data and produces easy-to-style figures.

GeoJSON with feature.id

- Here we load a GeoJSON file containing the geometry information for US counties, where `feature.id` is a FIPS code (The **Federal Information Processing Standard Publication 6-4** (FIPS 6-4) is a five-digit Federal Information Processing Standards code which uniquely identified counties and county equivalents in the United States, certain U.S. possessions, and certain freely associated states.).

Example 10.1

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
    counties = json.load(response)

counties["features"][0]
```

Output:

Example 10.2

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
    counties = json.load(response)

import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv",
                 dtype={"fips": str})

import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                    color_continuous_scale="Viridis",
                    range_color=(0, 12),
                    scope="usa",
                    labels={'unemp':'unemployment rate'}
                   )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Output:

Program (10b): Write a Python program for *Creating maps* using *Plotty Libraries*.

Aim

To write a Python program for ***creating maps using plotty libraries.***

Procedure

In this Python program, we utilize Plotly Express to create an interactive choropleth map visualizing GDP per Capita by country. The dataset used is sourced from Gapminder, providing a comprehensive view of economic indicators globally.

1. **Import Libraries:**
   - We start by importing the necessary libraries, including `plotly.express` for easy and interactive visualizations.
2. **Data Loading:**
   - The program fetches data from a CSV file hosted on GitHub using `pd.read_csv`. The dataset includes information about countries, their ISO codes, and GDP per Capita.
3. **Choropleth Map:**
   - The choropleth map is created using `px.choropleth`.
   - Key parameters include:
     - `locations`: ISO codes of countries.
     - `color`: GDP per Capita, determining the color intensity on the map.
     - `hover_name`: Country names appearing on hover.
     - `projection`: 'natural earth' projection for a global view.
     - `title`: The title of the map.
4. **Interactive Exploration:**
   - The resulting choropleth map is interactive, enabling users to hover over countries to see GDP per Capita values.

(upload this file for output)

CSV File: **gapminder_with_codes.csv**

Program 10(b1)

```
import plotly.express as px
import pandas as pd

# Import data from GitHub
data = pd.read_csv('gapminder_with_codes.csv')

# Create basic choropleth map
fig = px.choropleth(data, locations='iso_alpha', color='gdpPercap', hover_name='country',
            projection='natural earth', title='GDP per Capita by Country')
fig.show()
```

Output

Procedure

In this Python program, we leverage Plotly Express to create an insightful choropleth map that visualizes key demographic indicators across states and union territories in India. The data is sourced from India's census, providing a comprehensive overview of population distribution, density, and sex ratio.

Program Overview:

1. **Import Libraries:**
   - We begin by importing necessary libraries, including `json` for handling GeoJSON data, `numpy` for numerical operations, `pandas` for data manipulation, and `plotly.express` for creating interactive visualizations.
2. **Load GeoJSON and Census Data:**
   - The GeoJSON file representing Indian states is loaded, and census data is read from a CSV file containing information about population, density, and sex ratio.
3. **Data Preparation:**
   - We create a mapping between state names and their corresponding IDs for seamless integration with GeoJSON features.
   - Additional data preprocessing includes converting density values to integers and creating a unique identifier (`id`) for each state.
4. **Choropleth Map:**
   - The choropleth map is generated using `px.choropleth`. Key parameters include:
     - `locations`: State IDs for mapping.
     - `geojson`: GeoJSON data for Indian states.
     - `color`: Population, determining color intensity on the map.
     - `hover_name`: State names for hover information.
     - `hover_data`: Additional information displayed on hover, including density, sex ratio, and population.
     - `title`: Title of the map.
5. **Interactive Exploration:**
   - The resulting choropleth map is interactive, allowing users to hover over states to explore population demographics.

(upload these 2 files for output)

CSV File: **india_census.csv**

JSON File: **states_india.geojson**

Program 10(b2)

```
import json
import numpy as np
import pandas as pd
import plotly.express as px

#Uncomment below lines to render map on your browser
#import plotly.io as pio
#pio.renderers.default = 'browser'

india_states = json.load(open("states_india.geojson", "r"))

df = pd.read_csv("india_census.csv")
```

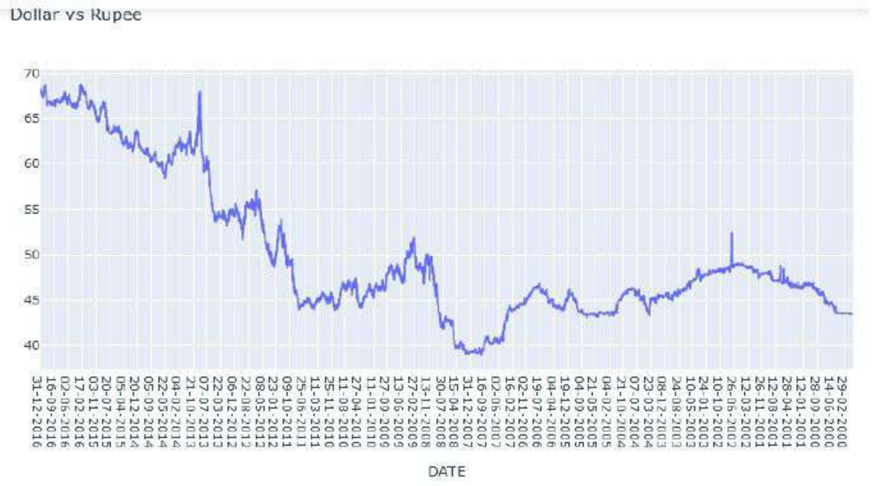
```
state_id_map = {}
for feature in india_states["features"]:
   feature["id"] = feature["properties"]["state_code"]
   state_id_map[feature["properties"]["st_nm"]] = feature["id"]

df = pd.read_csv("india_census.csv")
df["Density"] = df["Density[a]"].apply(lambda x: int(x.split("/")[0].replace(",", "")))
df["id"] = df["State or union territory"].apply(lambda x: state_id_map[x])

#print(df.head())
fig = px.choropleth(
   df,
   locations="id",
   geojson=india_states,
   color="Population",
   hover_name="State or union territory",
   hover_data=["Density", "Sex ratio", "Population"],
   title="India Population Statewise",
)
fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

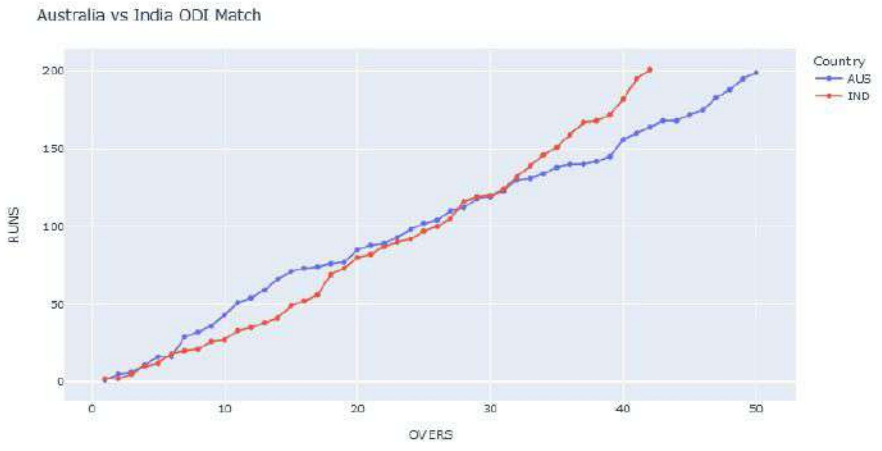
Output

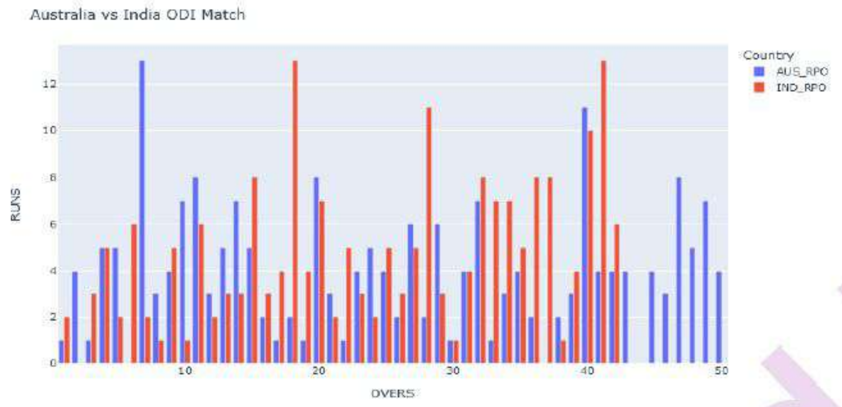
Result: Python program was successfully executed and ***created maps*** using ***Plotly Libraries.***

Do not write the following (X):

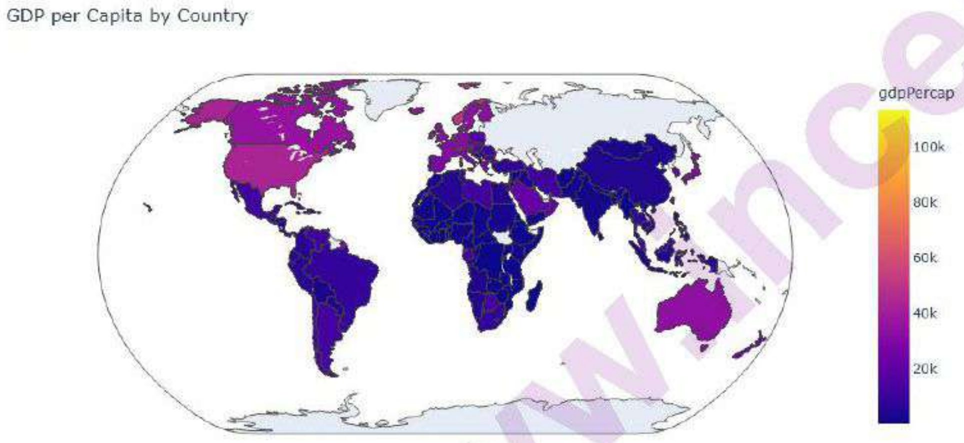
Program 10(a) Output

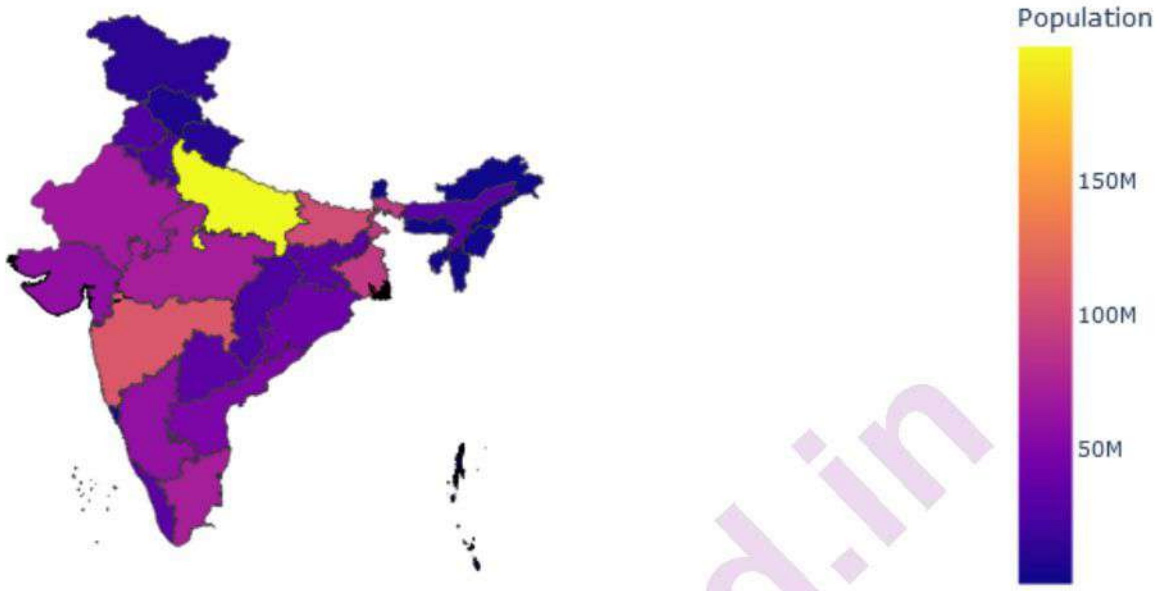






Program 10(b) Output

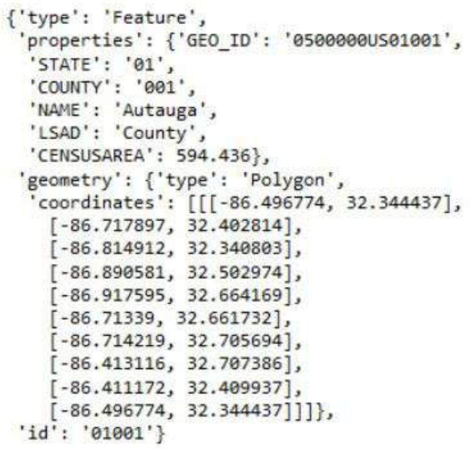




Example 10.1

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
   counties = json.load(response)

counties["features"][0]
```

Output



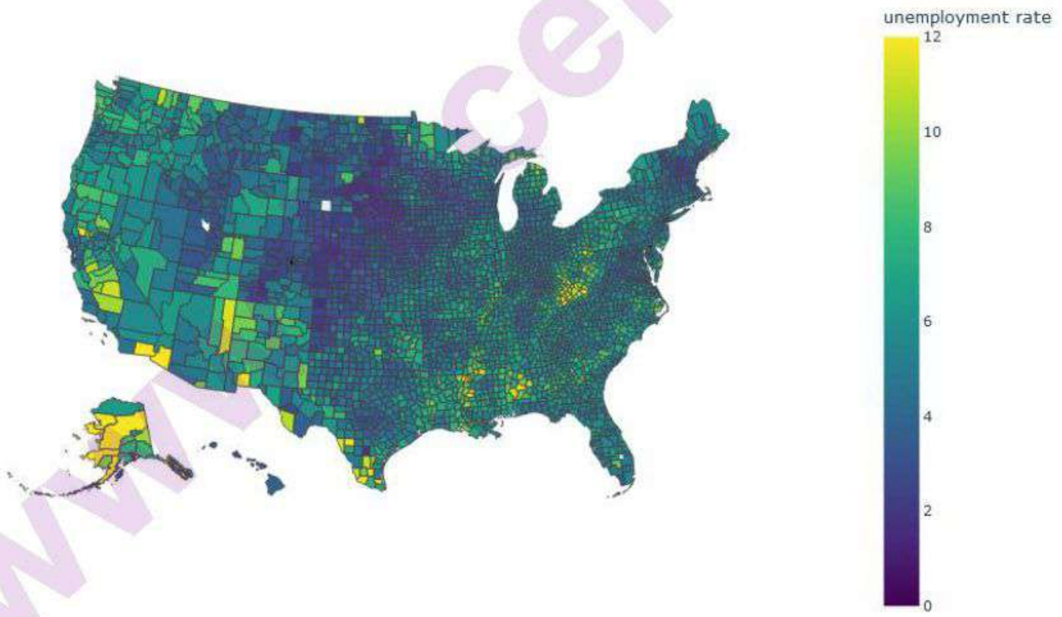
```
{'type': 'Feature',
 'properties': {'GEO_ID': '0500000US01001',
  'STATE': '01',
  'COUNTY': '001',
  'NAME': 'Autauga',
  'LSAD': 'County',
  'CENSUSAREA': 594.436},
 'geometry': {'type': 'Polygon',
  'coordinates': [[[-86.496774, 32.344437],
    [-86.717897, 32.402814],
    [-86.814912, 32.340803],
    [-86.890581, 32.502974],
    [-86.917595, 32.664169],
    [-86.71339, 32.661732],
    [-86.714219, 32.705694],
    [-86.413116, 32.707386],
    [-86.411172, 32.409937],
    [-86.496774, 32.344437]]]},
 'id': '01001'}
```

Example 10.2

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
   counties = json.load(response)

import pandas as pd
```

```
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-
16.csv",
            dtype={"fips": str})

import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                    color_continuous_scale="Viridis",
                    range_color=(0, 12),
                    scope="usa",
                    labels={'unemp':'unemployment rate'}
                   )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Output



***