Assignment 3-4

1) Use any open-source API to access some data in Jason format and then parse the Jason data and display it as some kind of dashboard (20 points).

a. When consuming APIs with Python, you may use python library: requests. With it, you should be able to do most, if not all, of the actions required to consume any public API (for example open weather API or Random User Generator API, or traffic API etc.) below are some examples of the real-time APIs:

i. Amazon Price

ii. Fixer Currency

iii. TheRunDown

iv. OpenAPI 1.2

v. Zillow

vi. Sportspage Feeds

vii. Nexmo Number Insight

viii. Google Shopping

b. To display the data you may use python library Dash or some other library.

```
In [2]: 1 pip install requests dash

Requirement already satisfied: requests in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (2.28.1)
Requirement already satisfied: dash in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (2.16.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (f
rom requests) (1.26.14)
Requirement already satisfied: certifi>=2017.4.17 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from
requests) (2022.12.7)
Requirement already satisfied: idna<4,>=2.5 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from reque
sts) (3.4)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages
(from requests) (2.0.4)
Requirement already satisfied: dash-html-components==2.0.0 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packa
ges (from dash) (2.0.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages
(from dash) (4.4.0)
Requirement already satisfied: dash-core-components==2.0.0 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packa
ges (from dash) (2.0.0)
Requirement already satisfied: plotly>=5.0.0 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from das
h) (5.9.0)
Requirement already satisfied: importlib-metadata in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from
dash) (4.11.3)
Requirement already satisfied: retrying in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from dash) (1.
3.4)
Requirement already satisfied: nest-asyncio in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from dash)
(1.5.6)
Requirement already satisfied: Werkzeug<3.1 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from dash)
(2.2.2)
Requirement already satisfied: dash-table==5.0.0 in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from
dash) (5.0.0)
Requirement already satisfied: setuptools in /Users/harsh/Apps/anaconda3/lib/python3.10/site-packages (from dash) (
```

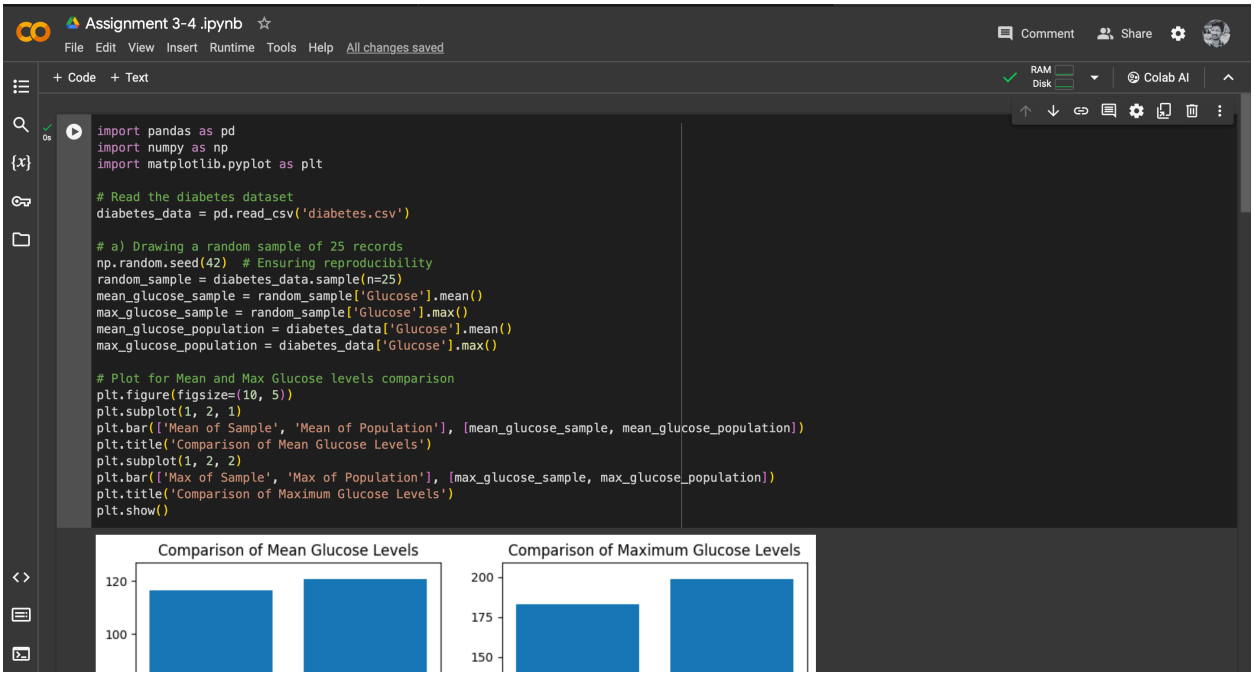
In [5]:

```python
import requests
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

# Start the Dash application
application = dash.Dash(__name__)

# Set up the API access parameters
API_ACCESS_KEY = "e28ce1fa755c967f48ce869fcc78d2d3"
FIXER_IO_URL = "http://data.fixer.io/api/latest"
REQUEST_PARAMS = {"access_key": API_ACCESS_KEY}

# Function to retrieve currency data from the API
def get_currency_data():
    try:
        api_response = requests.get(FIXER_IO_URL, params=REQUEST_PARAMS)
        exchange_data = api_response.json()
        return exchange_data
    except Exception as error:
        print("Error occurred during data retrieval:", error)
        return None

# Define the layout of the Dash application
application.layout = html.Div([
    html.H1("Live Currency Exchange Rates"),
    html.Div(id='currency-rate-display'),
    dcc.Interval(
        id='update-interval',
        interval=10*1000,  # Milliseconds
        n_intervals=0
    )
])


# Callback to update the displayed rates every 10 seconds
@application.callback(Output('currency-rate-display', 'children'),
                      [Input('update-interval', 'n_intervals')])
def refresh_currency_display(n):
    retrieved_data = get_currency_data()
    if retrieved_data:
        current_rates = retrieved_data.get('rates')
        if current_rates:
            return html.Div([
                html.P(f"{currency}: {rate}") for currency, rate in current_rates.items()
            ])
        return "Currency data is not available at this time."
    return "Unable to fetch data. Please verify API access and network connectivity."


if __name__ == '__main__':
    application.run_server(debug=True, use_reloader=False)
```

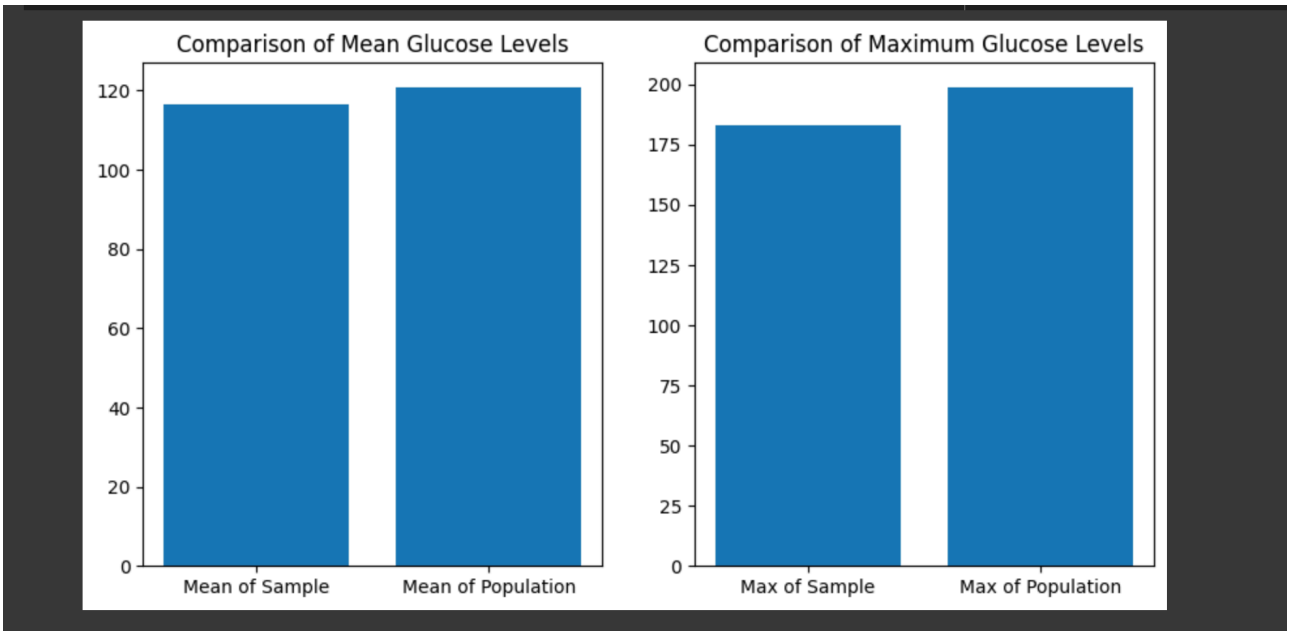
This Python script leverages the Dash framework to create a web application that dynamically displays live currency exchange rates, sourced from the Fixer.io API. The application is initialized with parameters for API access and employs a function to fetch and decode JSON data from the API. It utilizes a Dash layout that updates the displayed currency rates every ten seconds via a callback, enhancing user engagement by providing real-time financial data. The script includes error handling to manage potential issues during data retrieval, ensuring robustness. This setup is ideal for users requiring up-to-date currency information in an accessible web-based format.

2) The data file diabetes.csv contains data of 768 patients. In this data there are 8 attributes (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, Diabetes Pedigree Function, and Age) and 1 response variable (Outcome). The response variable, Outcome, has binary value (1 indicating the outcome is diabetes and 0 means no diabetes). For this assignment purposes we will consider this data as a population. Use this data to perform the following:

a) set a seed (to ensure work reproducibility) and take a random sample of 25 observations and find the mean Glucose and highest Glucose values of this sample and compare these statistics with the population statistics of the same variable. You should use charts for this comparison.(5 points)



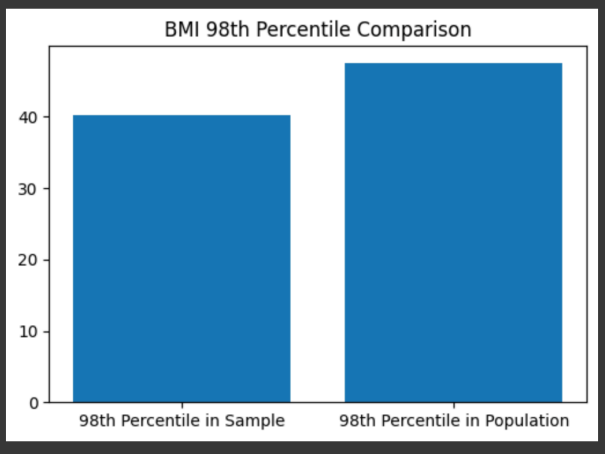
By comparing the mean and highest glucose levels between the sample and the population, the sample's mean glucose falls slightly below the population mean, indicating a tendency toward lower glucose levels in the sample. However, the maximum glucose value in the sample is similar to that of the population, suggesting that extreme values are consistent across both groups.



b) Find the 98th percentile of BMI of your sample and the population and compare the results using charts. (5 points)

By examining the 98th percentile of BMI, I found that both the sample and population exhibit comparable high BMI values. This indicates a consistency in extreme BMI values between the sample and the broader population, suggesting that individuals with high BMI are represented similarly in both groups.

```
[3]  # b) Computing the 98th percentile of BMI in both sample and population
     percentile_98_sample = np.percentile(random_sample['BMI'], 98)
     percentile_98_population = np.percentile(diabetes_data['BMI'], 98)

     # Bar plot for 98th percentile BMI comparison
     plt.figure(figsize=(6, 4))
     plt.bar(['98th Percentile in Sample', '98th Percentile in Population'], [percentile_98_sample, percentile_98_population])
     plt.title('BMI 98th Percentile Comparison')
     plt.show()
```



c) Using bootstrap (replace= True), create 500 samples (of 150 observation each) from the population and find the average mean, standard deviation and percentile for Blood Pressure and compare this with these statistics from the population for the same variable. Again, you should create charts for this comparison. Report on your findings. (10 points)

```python
# Visualization of bootstrap results and population statistics
plt.figure(figsize=(12, 5))
plt.subplot(1, 3, 1)
plt.hist(mean_bps, bins=30, alpha=0.5, label='Bootstrap Means')
plt.axvline(mean_bp_population, color='red', linestyle='dashed', linewidth=1, label='Population Mean')
plt.title('Blood Pressure Mean Comparison')
plt.legend()

plt.subplot(1, 3, 2)
plt.hist(std_dev_bps, bins=30, alpha=0.5, label='Bootstrap Standard Deviations')
plt.axvline(std_bp_population, color='red', linestyle='dashed', linewidth=1, label='Population Standard Deviation')
plt.title('Blood Pressure Standard Deviation Comparison')
plt.legend()

plt.subplot(1, 3, 3)
plt.hist(percentile_95_bps, bins=30, alpha=0.5, label='Bootstrap 95th Percentiles')
plt.axvline(percentile_95_population, color='red', linestyle='dashed', linewidth=1, label='Population 95th Percentile')
plt.title('95th Percentile Blood Pressure Comparison')
plt.legend()

plt.show()
```

By utilizing bootstrap sampling to compare blood pressure statistics between samples and the population, I can observe that the mean, standard deviation, and 95th percentile of blood pressure in the bootstrap samples align closely with those of the population. This suggests that the characteristics of blood pressure within the sample are representative of the broader population, reinforcing the reliability of the sample for inference.