**Assignment 3 and 4**

Mallikarjun Challa

16342908

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).
   1. 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:
      1. [Amazon Price](https://rapidapi.com/ajmorenodelarosa/api/amazon-price1)
      2. [Fixer Currency](https://rapidapi.com/fixer/api/fixer-currency)
      3. [TheRunDown](https://rapidapi.com/therundown/api/therundown)
      4. [OpenAPI 1.2](https://rapidapi.com/transloc/api/openapi-1-2)
      5. [Zillow](https://rapidapi.com/dimashirokov/api/Zillow)
      6. [Sportspage Feeds](https://rapidapi.com/SportspageFeeds/api/sportspage-feeds)
      7. [Nexmo Number Insight](https://rapidapi.com/nexmo/api/nexmo-number-insight)
      8. [Google Shopping](https://rapidapi.com/ajmorenodelarosa/api/google-shopping)

A screenshot of a computer

Description automatically generated

I have installed Dash Library.

Getting data from public Open API, I have used Rapid API to the data, its free and provides 100 calls per day

import requests

# My URL and headers (used rapid api website and movies based data)

url = "https://streaming-availability.p.rapidapi.com/countries"

headers = {

    "X-RapidAPI-Key": "8061d144dfmsh774995b2e2872b6p1bf18ejsnba58c382ff31",

    "X-RapidAPI-Host": "streaming-availability.p.rapidapi.com"

}

# Sending a GET request to the API

response = requests.get(url, headers=headers)

# Checking if the request was successful

if response.status\_code == 200:

    # Parsing my JSON response

    data = response.json()

Used the above key and header to get movie data in JSON format

   def display\_sample\_dashboard(data, sample\_countries=3, sample\_services=2):

        print("Streaming Availability Sample Dashboard")

        print("----------------------------------------")

        total\_countries = len(data.get("result", {}))

        print("Total Countries:", total\_countries)

        displayed\_countries = 0

        displayed\_services = 0

        for country\_code, country\_data in data.get("result", {}).items():

            print("Country Code:", country\_code)

            print("Country Name:", country\_data.get("name", ""))

            print("Services:")

            for service\_id, service\_data in country\_data.get("services", {}).items():

                print("\tService ID:", service\_id)

                print("\tService Name:", service\_data.get("name", ""))

                print("\tHomepage:", service\_data.get("homePage", ""))

                print("\tTheme Color Code:", service\_data.get("themeColorCode", ""))

                print("\tLight Theme Image:", service\_data.get("images", {}).get("lightThemeImage", ""))

                print("\tDark Theme Image:", service\_data.get("images", {}).get("darkThemeImage", ""))

                print("\tWhite Image:", service\_data.get("images", {}).get("whiteImage", ""))

                print("\tSupported Streaming Types:")

                streaming\_types = service\_data.get("supportedStreamingTypes", {})

                print("\t\tAddon:", streaming\_types.get("addon", False))

                print("\t\tBuy:", streaming\_types.get("buy", False))

                print("\t\tRent:", streaming\_types.get("rent", False))

                print("\t\tFree:", streaming\_types.get("free", False))

                print("\t\tSubscription:", streaming\_types.get("subscription", False))

                print("\tAddons:")

                for addon\_id, addon\_data in service\_data.get("addons", {}).items():

                    print("\t\tAddon ID:", addon\_id)

                    print("\t\tAddon Name:", addon\_data.get("displayName", ""))

                    print("\t\tAddon Homepage:", addon\_data.get("homePage", ""))

                    print("\t\tAddon Theme Color Code:", addon\_data.get("themeColorCode", ""))

                    print("\t\tAddon Image:", addon\_data.get("image", ""))

                print("---------------------------------")

                displayed\_services += 1

                if displayed\_services >= sample\_services:

                    break

            displayed\_countries += 1

            if displayed\_countries >= sample\_countries:

                break

    # Calling the display\_sample\_dashboard function

    display\_sample\_dashboard(data)

else:

    print("Failed to retrieve data. Status code:", response.status\_code)

A screenshot of a computer

Description automatically generated

Sample data from the API

* 1. To display the data you may use the Python library Dash or some other library.

1. import matplotlib.pyplot as plt
2. # Function to generate a bar chart for streaming services by country
3. def bar\_chart\_services\_by\_country(data):
4. countries = []
5. num\_services = []
6. for country\_code, country\_data in data.get("result", {}).items():
7. countries.append(country\_data.get("name", ""))
8. num\_services.append(len(country\_data.get("services", {})))
9. plt.figure(figsize=(10, 6))
10. plt.bar(countries, num\_services, color='skyblue')
11. plt.xlabel('Countries')
12. plt.ylabel('Number of Services')
13. plt.title('Streaming Services by Country')
14. plt.xticks(rotation=45, ha='right')
15. plt.tight\_layout()
16. plt.show()
17. # Calling the function to generate the bar chart
18. bar\_chart\_services\_by\_country(data)

plotting a visual dashboard using matplot

A graph of blue lines

Description automatically generated

Code for plotting a visual dashboard using Dash Library

import dash

from dash import dcc, html

import plotly.graph\_objs as go

# Initializing the Dash app

app = dash.Dash(\_\_name\_\_)

def bar\_chart\_services\_by\_country(data):

    countries = []

    num\_services = []

    for country\_code, country\_data in data.get("result", {}).items():

        countries.append(country\_data.get("name", ""))

        num\_services.append(len(country\_data.get("services", {})))

    # Creating a bar chart figure

    figure = {

        'data': [

            go.Bar(x=countries, y=num\_services, marker=dict(color='skyblue'))

        ],

        'layout': {

            'title': 'Streaming Services by Country',

            'xaxis': {'title': 'Countries'},

            'yaxis': {'title': 'Number of Services'},

            'margin': {'l': 40, 'b': 40, 't': 40, 'r': 40},

            'plot\_bgcolor': '#f7f7f7',

            'paper\_bgcolor': '#f7f7f7'

        }

    }

    return figure

#  generating a pie chart for streaming types distribution

def pie\_chart\_streaming\_types(data):

    streaming\_types\_count = {

        'Addon': 0,

        'Buy': 0,

        'Rent': 0,

        'Free': 0,

        'Subscription': 0

    }

    # Iterating through the data to count streaming types

    for country\_code, country\_data in data.get("result", {}).items():

        for service\_id, service\_data in country\_data.get("services", {}).items():

            streaming\_types = service\_data.get("supportedStreamingTypes", {})

            for streaming\_type, supported in streaming\_types.items():

                if supported:

                    # Incrementing the count for the respective streaming type

                    streaming\_types\_count[streaming\_type.capitalize()] += 1

    labels = list(streaming\_types\_count.keys())

    sizes = list(streaming\_types\_count.values())

    # Creating the pie chart figure

    figure = {

        'data': [

            go.Pie(labels=labels, values=sizes, hole=0.3)

        ],

        'layout': {

            'title': 'Streaming Types Distribution'

        }

    }

    return figure

app.layout = html.Div([

    html.H1('Streaming Dashboard', style={'textAlign': 'center'}),

    html.Div([

        html.Div([

            dcc.Graph(id='bar-chart', figure=bar\_chart\_services\_by\_country(data))

        ], className='six columns'),

        html.Div([

            dcc.Graph(id='pie-chart', figure=pie\_chart\_streaming\_types(data))

        ], className='six columns')

    ], className='row')

])

if \_\_name\_\_ == '\_\_main\_\_':

    app.run\_server(debug=True)

Plotted the visual using the Dash library

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Data link for question 2:  <https://app.box.com/s/7qv44umhw0vnzgmoe9krfkfkv5kf2atv>

2) The data file diabetes.csv contains data of 768 patients. In this data there are 8 attributes (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, 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:

1. 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)

# Set seed for reproducibility

np.random.seed(42)

# Take a random sample of 25 observations

sample = data.sample(n=25)

# Calculating mean and highest Glucose values for sample and population

sample\_mean\_glucose = sample['Glucose'].mean()

sample\_max\_glucose = sample['Glucose'].max()

population\_mean\_glucose = data['Glucose'].mean()

population\_max\_glucose = data['Glucose'].max()

# Creating a bar chart for comparison

labels = ['Sample Mean Glucose', 'Sample Max Glucose', 'Population Mean Glucose', 'Population Max Glucose']

values = [sample\_mean\_glucose, sample\_max\_glucose, population\_mean\_glucose, population\_max\_glucose]

plt.figure(figsize=(10, 6))

plt.bar(labels, values, color=['blue', 'green', 'orange', 'red'])

plt.title('Comparison of Glucose Statistics (Sample vs Population)')

plt.ylabel('Glucose Value')

plt.show()

A graph of different colored squares

Description automatically generated

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

# Finding the 98th percentile of BMI for sample and population

sample\_98th\_percentile\_bmi = np.percentile(sample['BMI'], 98)

population\_98th\_percentile\_bmi = np.percentile(data['BMI'], 98)

# Creating a pie chart for comparison

labels = ['Sample 98th Percentile BMI', 'Population 98th Percentile BMI']

sizes = [sample\_98th\_percentile\_bmi, population\_98th\_percentile\_bmi]

plt.figure(figsize=(8, 8))

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)

plt.title('Comparison of 98th Percentile BMI (Sample vs Population)')

plt.axis('equal')

plt.show()

A blue and orange pie chart

Description automatically generated

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 BloodPressure 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)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Loading the dataset from Excel file

file\_path = "diabetes.csv"

data = pd.read\_csv(file\_path)

def bootstrap\_sampling(data, variable, num\_samples, sample\_size):

    population\_data = data[variable].values

    mean\_values = []

    std\_dev\_values = []

    percentile\_values = []

    # Performing the bootstrap sampling

    for \_ in range(num\_samples):

        sample = np.random.choice(population\_data, size=sample\_size, replace=True)

        mean\_values.append(np.mean(sample))

        std\_dev\_values.append(np.std(sample))

        percentile\_values.append(np.percentile(sample, 98))

    return mean\_values, std\_dev\_values, percentile\_values

# Setting seed for reproducibility

np.random.seed(42)

# Parameters for bootstrap sampling

num\_samples = 500

sample\_size = 150

mean\_values, std\_dev\_values, percentile\_values = bootstrap\_sampling(data, 'BloodPressure', num\_samples, sample\_size)

population\_mean = data['BloodPressure'].mean()

population\_std\_dev = data['BloodPressure'].std()

population\_percentile = np.percentile(data['BloodPressure'], 98)

# Printing the  population statistics

print("Population Mean BloodPressure:", population\_mean)

print("Population Standard Deviation BloodPressure:", population\_std\_dev)

print("Population 98th Percentile BloodPressure:", population\_percentile)

# Comparing the  bootstrap statistics with population statistics

# Plotting histograms for comparison

plt.figure(figsize=(12, 6))

# Histogram of population data

plt.hist(data['BloodPressure'], bins=30, alpha=0.5, label='Population', color='blue', density=True)

# Histogram of bootstrap sample means

plt.hist(mean\_values, bins=30, alpha=0.5, label='Bootstrap Samples', color='orange', density=True)

plt.axvline(population\_mean, color='blue', linestyle='dashed', linewidth=1)

plt.axvline(np.mean(mean\_values), color='orange', linestyle='dashed', linewidth=1)

plt.xlabel('BloodPressure')

plt.ylabel('Density')

plt.title('Comparison of BloodPressure Distribution')

plt.legend()

plt.grid(True)

plt.show()

# Plotting boxplots for comparison

plt.figure(figsize=(10, 6))

plt.boxplot([data['BloodPressure'], mean\_values], labels=['Population', 'Bootstrap Samples'])

plt.ylabel('BloodPressure')

plt.title('Comparison of BloodPressure Distribution')

plt.grid(True)

plt.show()

A graph of a normal distribution

Description automatically generated

Population Mean BloodPressure: 69.10546875

Population Standard Deviation BloodPressure: 19.355807170644777

Population 98th Percentile BloodPressure: 99.31999999999994

A diagram of a blood pressure distribution

Description automatically generated

It is determined that the population's mean blood pressure is 69.11 mmHg, with a standard deviation of roughly 19.36 mmHg. Furthermore, the population's blood pressure is roughly 99.32 mmHg at the 98th percentile. We found that the bootstrap sample means tend to cluster around the population mean, suggesting a reasonably accurate estimation, when we compared these population statistics with those derived from 500 bootstrap samples, each comprising 150 observations. The bootstrap sample standard deviation distribution, however, shows some variability around the population standard deviation, indicating possible sampling variability. The robustness of bootstrapping in capturing high percentile values is demonstrated by the fact that, despite this variability, the 98th percentile blood pressure values in the bootstrap samples are relatively close to the population's 98th percentile.

Submission:

Create a public GitHub repo and upload the folders for the assignment on the GitHub and submit the link to Canvas.