Analytics Vidhya

Accessing APIs using Python

Instructor

Prashant Sahu
Manager Data Science, Analytics Vidhya

Understanding HTTP Protocol

HTTP

Hyper Text Transfer Protocol

Foundations of Data Communications on the Web

HTTP Methods

GET: Retrieve data

POST: Submit data to the server

PUT: Update existing data

DELETE: Remove data

Status Codes

200: Success

404: Not Found

500: Server Error

RESTful APIs

REST

Representational State Transfer

Architectural Style for Designing Networked Applications

Principles of REST

Statelessness: No client is stored on the serv

Client-server Architecture: Separation of concerns

Uniform Interface: Standard methods and endpoints

Cacheability: Responses can be matched for better performance

Uses HTTP Methods for CRUD Operations

Analytics Vidhya

Components of a RESTful API

Endpoint/Resources URL:

The URL where the API can be accessed

Example: https://api.example.com/users

HTTP Method

Determines the operation (GET, POST, PUT, DELETE)

Headers

Additional information sent with the request

Content-Type, Authorization tokens

Body/Payoad

Data sent with the request (for POST and PUT)

often in JSON format

Responses

Content-Type, Authorization tokens

Analytics Vidhya

```json
{
  "name": "Alice",
  "age": 30,
  "skills": ["Python", "Machine Learning"]
}
```

JSON Data Format

- JSON: Javascript Object Notation
- Lightweight Data Interchange Format
- Easy to Read and Write for Humans and Machines
- Data is structured in key-value pairs

Analytics Vidhya

Accessing APIs Using Python

Python Libraries for API Access

requests: Simplifies HTTP requests

urllib: Standard library module for URL handling

Why use requests

User friendly and concise

Supports all HTTP methods

Handles sessions and cookies

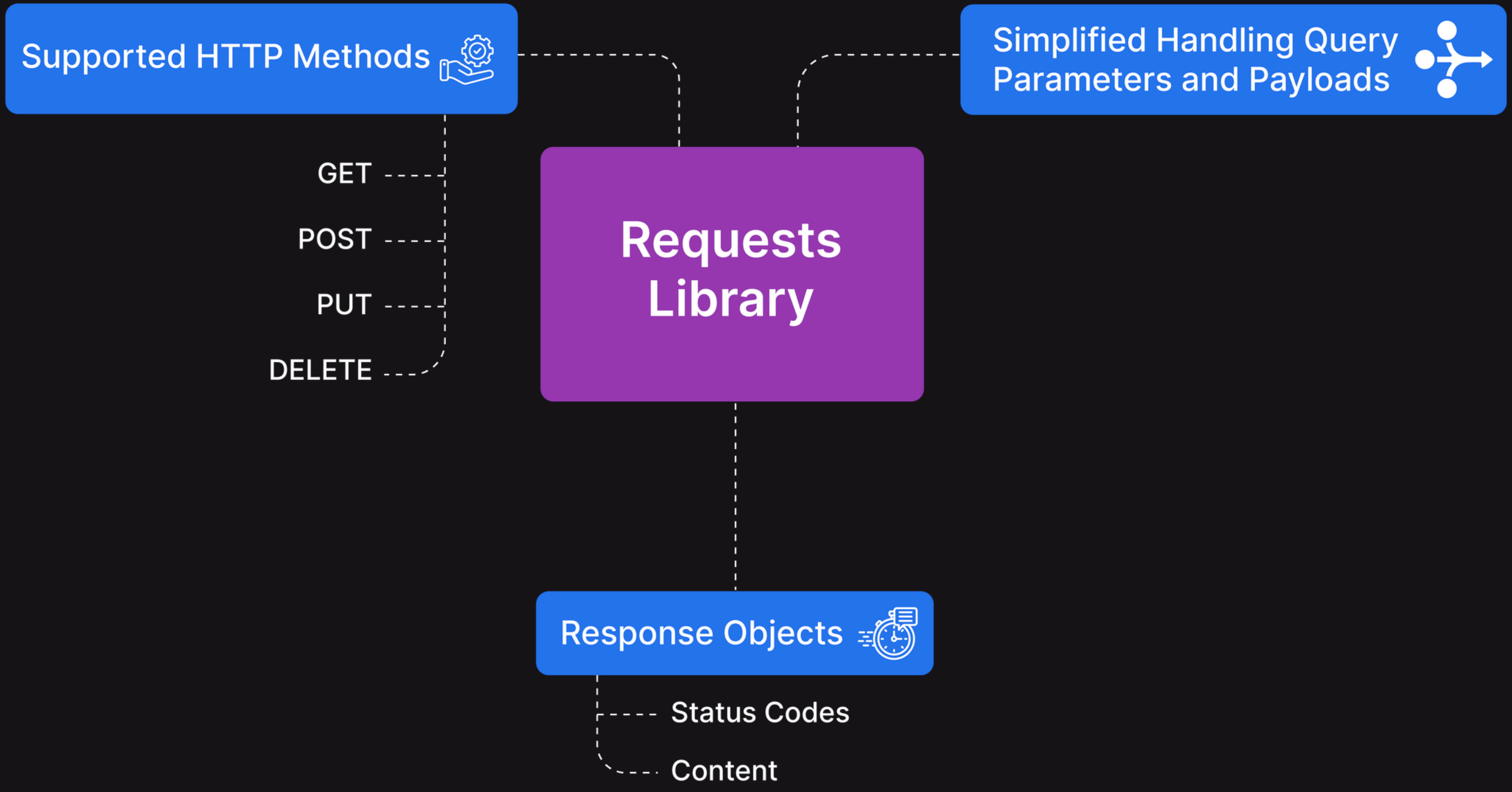
Basic Steps

Import the library

Use functions like get(),post()

Handles the response

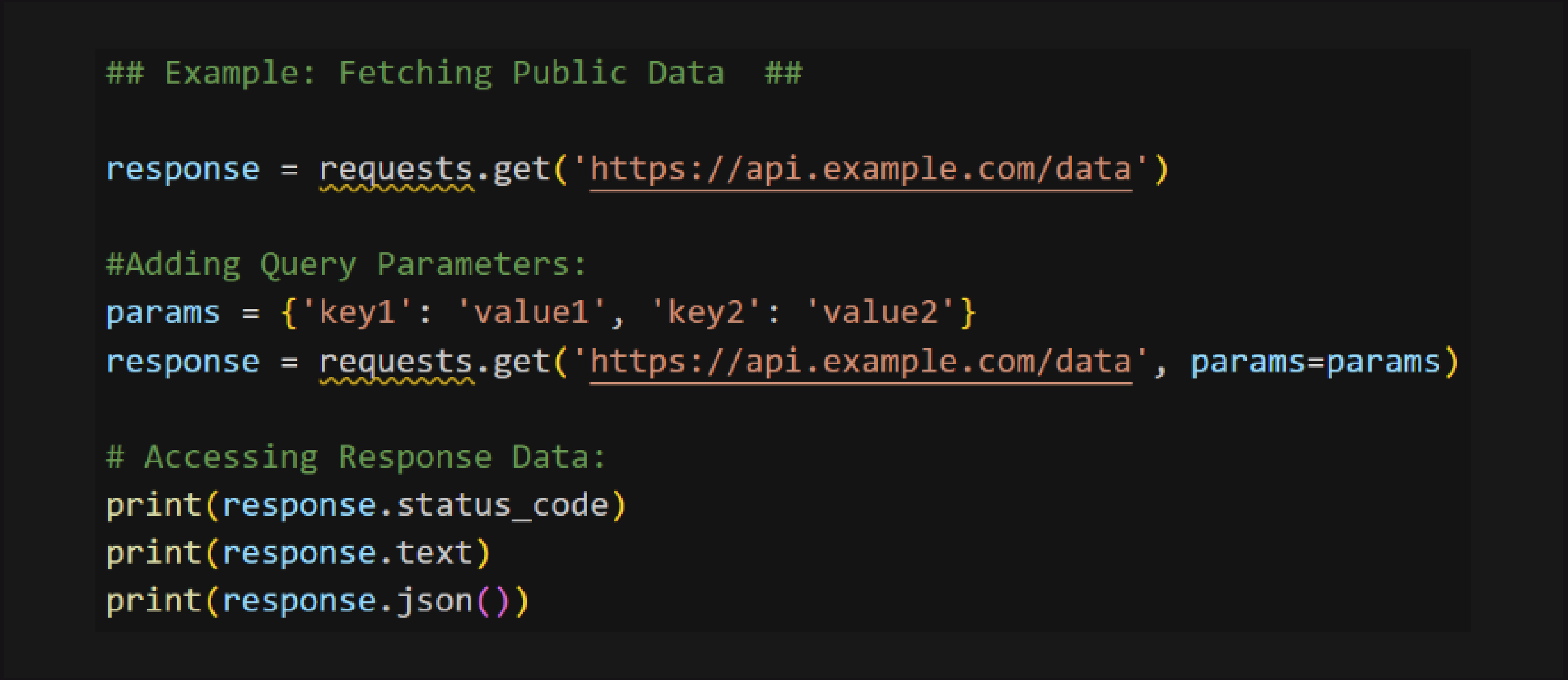
Introduction to the requests Library



pip install request

Analytics Vidhya

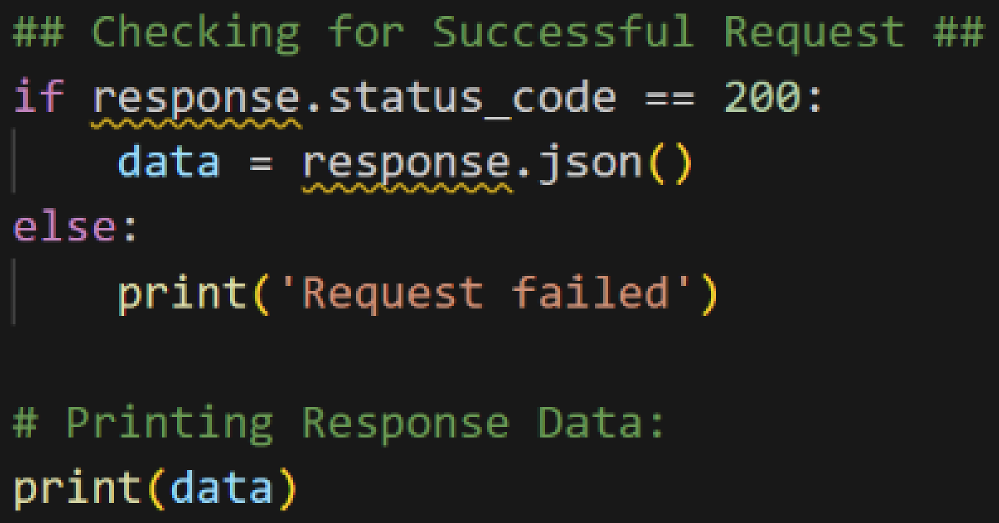
Making GET Requests



```python
## Example: Fetching Public Data  ##

response = requests.get('https://api.example.com/data')

#Adding Query Parameters:
params = {'key1': 'value1', 'key2': 'value2'}
response = requests.get('https://api.example.com/data', params=params)

# Accessing Response Data:
print(response.status_code)
print(response.text)
print(response.json())
```

Handling API Responses

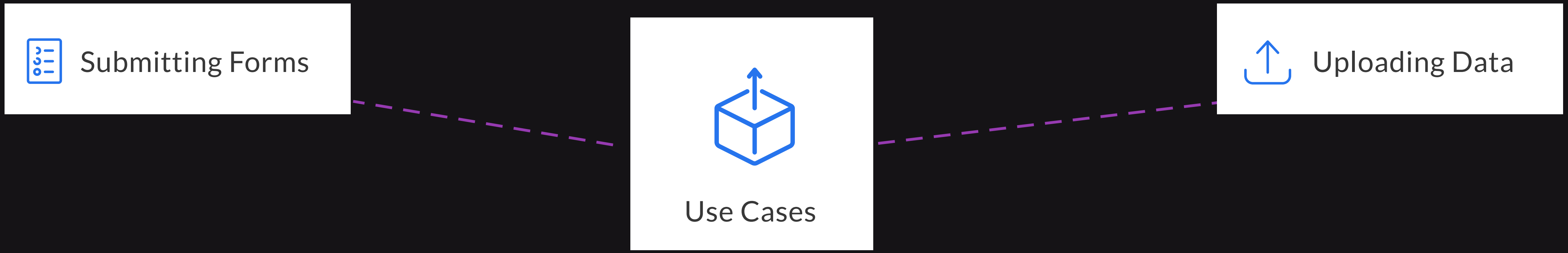
Responses Object Attributes:

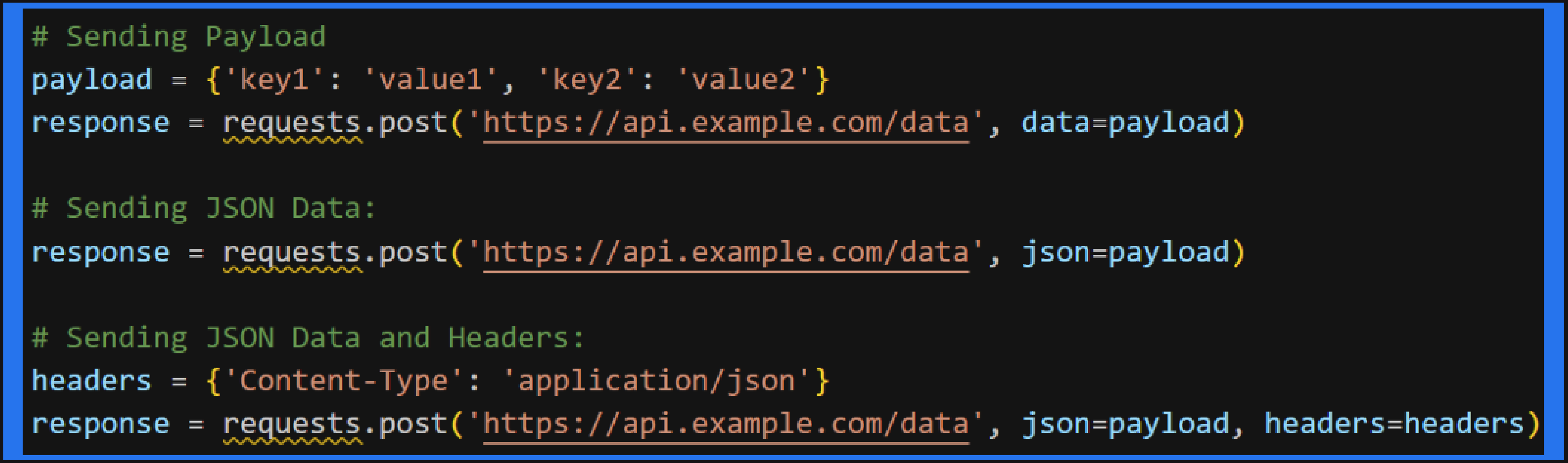


```python
## Checking for Successful Request ##
if response.status_code == 200:
    data = response.json()
else:
    print('Request failed')

# Printing Response Data:
print(data)
```

- status_code: HTTP status code
- headers: Response headers
- text: Response content as a string
- json(): Parse response as JSON

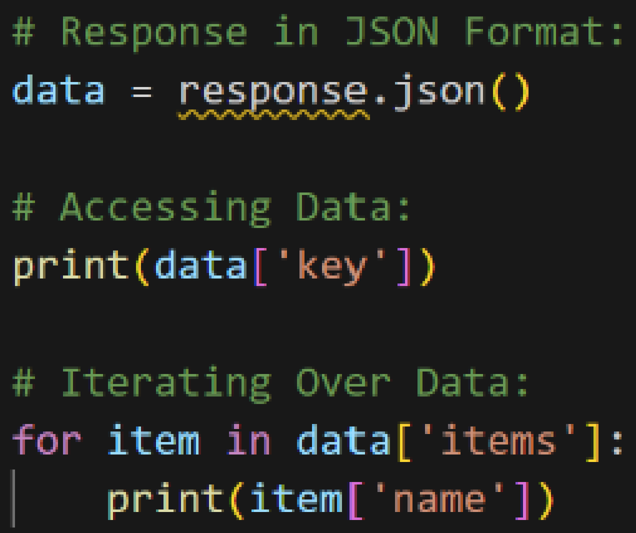
Making POST Requests





```python
# Sending Payload
payload = {'key1': 'value1', 'key2': 'value2'}
response = requests.post('https://api.example.com/data', data=payload)

# Sending JSON Data:
response = requests.post('https://api.example.com/data', json=payload)

# Sending JSON Data and Headers:
headers = {'Content-Type': 'application/json'}
response = requests.post('https://api.example.com/data', json=payload, headers=headers)
```

Parsing JSON Data



```python
# Response in JSON Format:
data = response.json()

# Accessing Data:
print(data['key'])

# Iterating Over Data:
for item in data['items']:
    print(item['name'])
```

Handling Nested Structures

- Understand the structure of the JSON response
- Use nested keys to access data

Analytics Vidhya