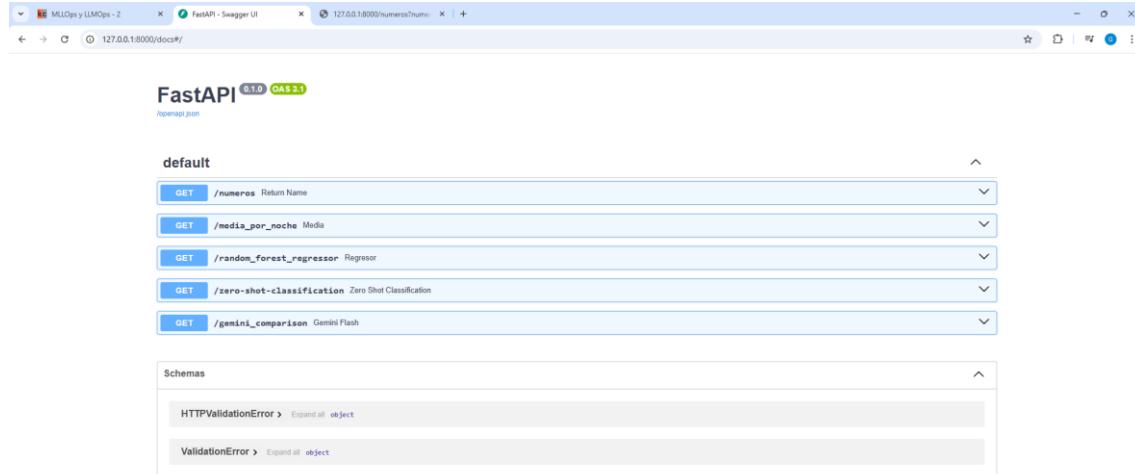


## Capturas de pantallas de FastAPI

Comenzamos con los 5 módulos, que van creciendo en dificultad. Ahora mostraremos las capturas de la API, y después las de las llamadas http.



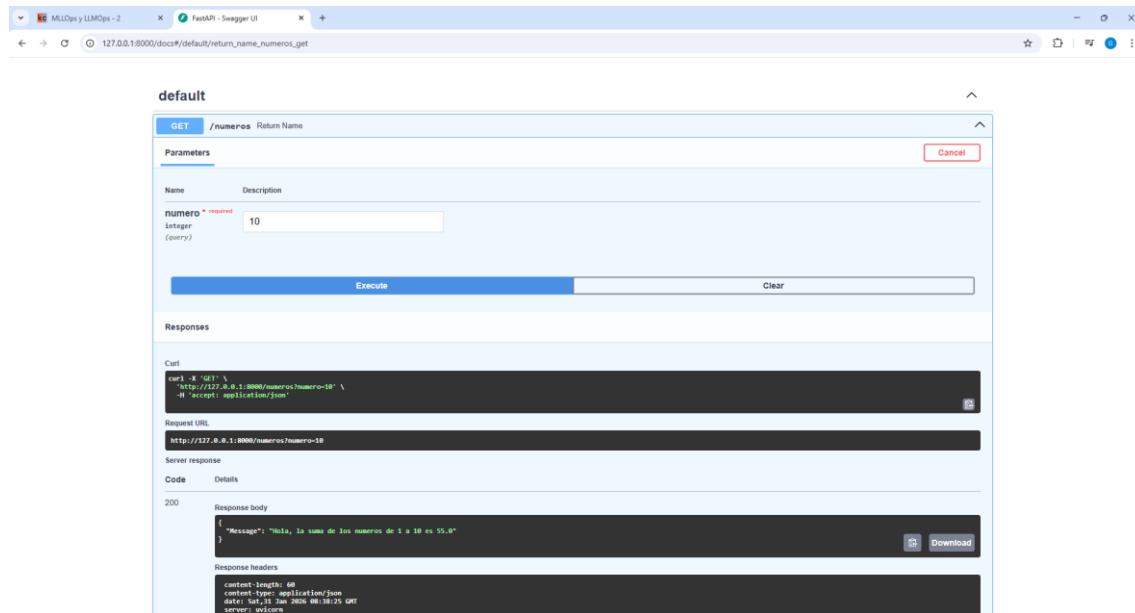
The screenshot shows the FastAPI Swagger UI interface. At the top, it displays "FastAPI 0.1.0 GAS 3.1" and "openapi.json". Below this, under the "default" section, there is a list of five GET endpoints:

- /numeros Return Name
- /media\_por\_noche Media
- /random\_forest\_regressor Regresor
- /zero-shot-classification Zero Shot Classification
- /gemini\_comparison Gemini Flash

Below the endpoints, there is a "Schemas" section containing two definitions:

- HTTPValidationError > Expand all object
- ValidationError > Expand all object

El primer módulo es simplemente que pide un número entero y devuelve la suma de los enteros desde 1 a ese número.



The screenshot shows the FastAPI Swagger UI interface for the "/numeros" endpoint. Under the "Parameters" section, there is one parameter named "numero" with a value of "10". Below this, there are sections for "Responses" and "Code".

In the "Responses" section, there is a "curl" example:

```
curl -X 'GET' \
  'http://127.0.0.1:8000/numeros?numero=10' \
  -H 'accept: application/json'
```

Under "Code", there is a "200" section showing the "Response body" and "Response headers".

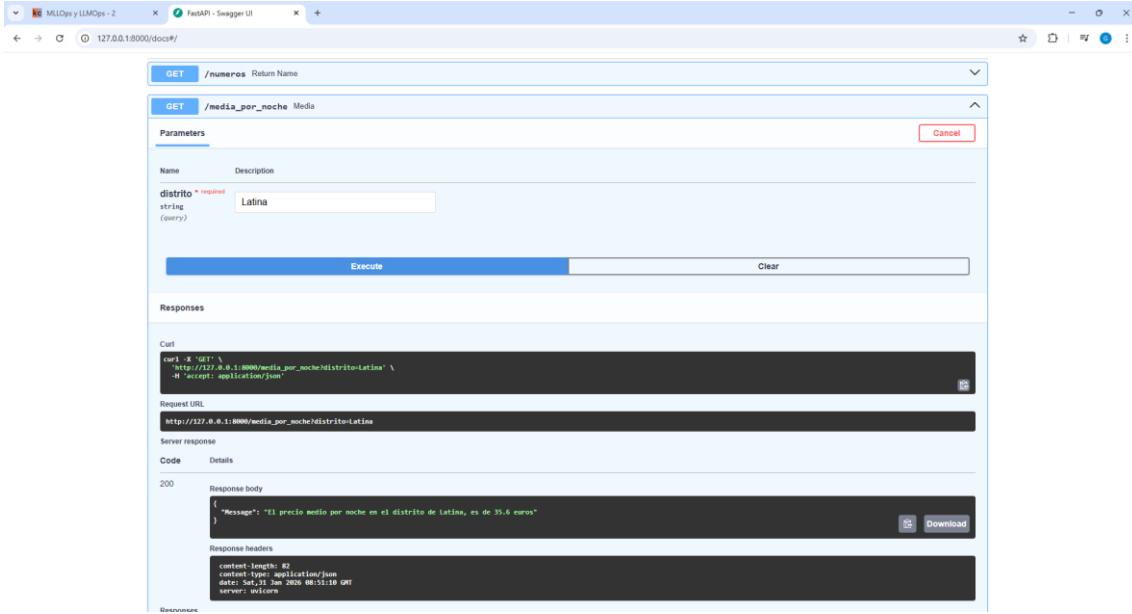
The "Response body" contains the JSON output:

```
[{"Message": "Hola, la suma de los numeros de 1 a 10 es 55.0"}]
```

The "Response headers" show the following information:

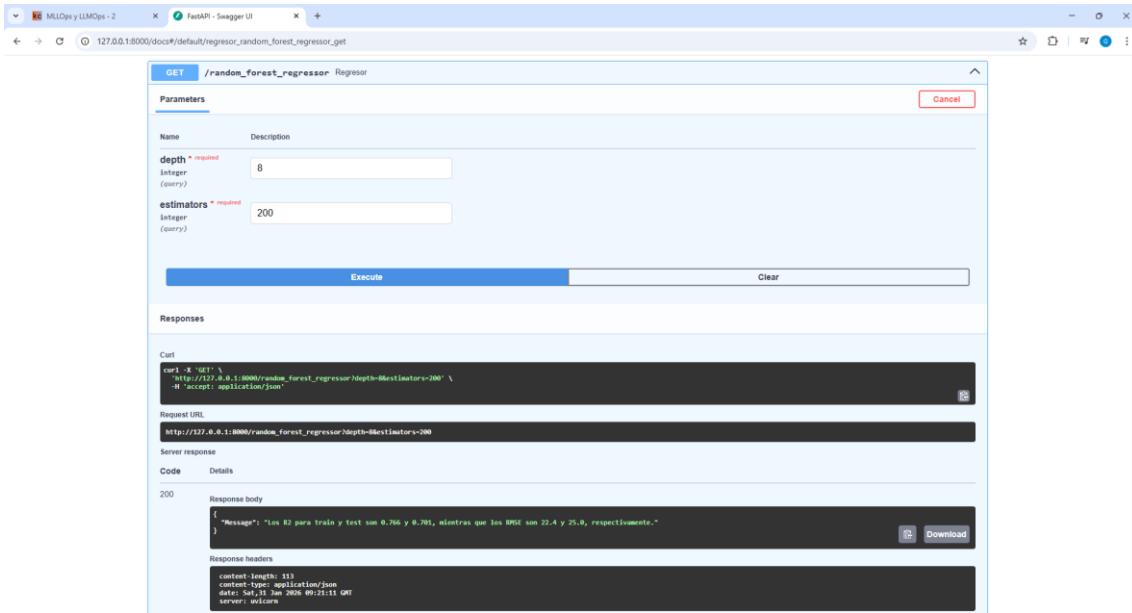
```
content-length: 60
content-type: application/json
date: Sat, 11 Mar 2023 08:49:47 GMT
server: uvicorn
```

El siguiente módulo hace uso de los dataset que hemos utilizado en MFlow, así como de las funciones definidas extraídas del notebook de esa parte. El módulo pide el nombre de un distrito y devuelve el precio medio por noche en él de la oferta de Airbnb.



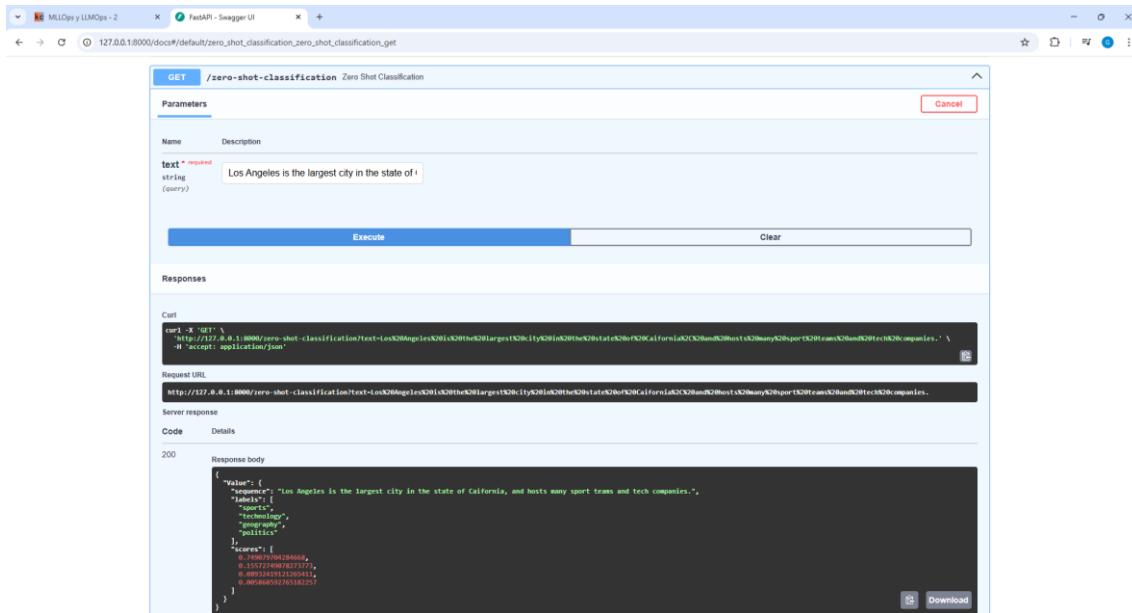
The screenshot shows the FastAPI - Swagger UI interface. A POST request is selected for the endpoint `/media_por_noche`. The parameter `distrito` is set to `Latina`. The response body shows a JSON message: `{ "Message": "El precio medio por noche en el distrito de Latina, es de 35.6 euros" }`. Response headers include `Content-Type: application/json`, `Date: Sat, 31 Jan 2026 09:21:16 GMT`, and `Server: uvicorn`.

El tercer módulo profundiza algo más, y calcula el modelo de Random Forest y métricas de R2 y RMSE, tras pedir la profundidad y el número de estimadores, tanto para train como para test.

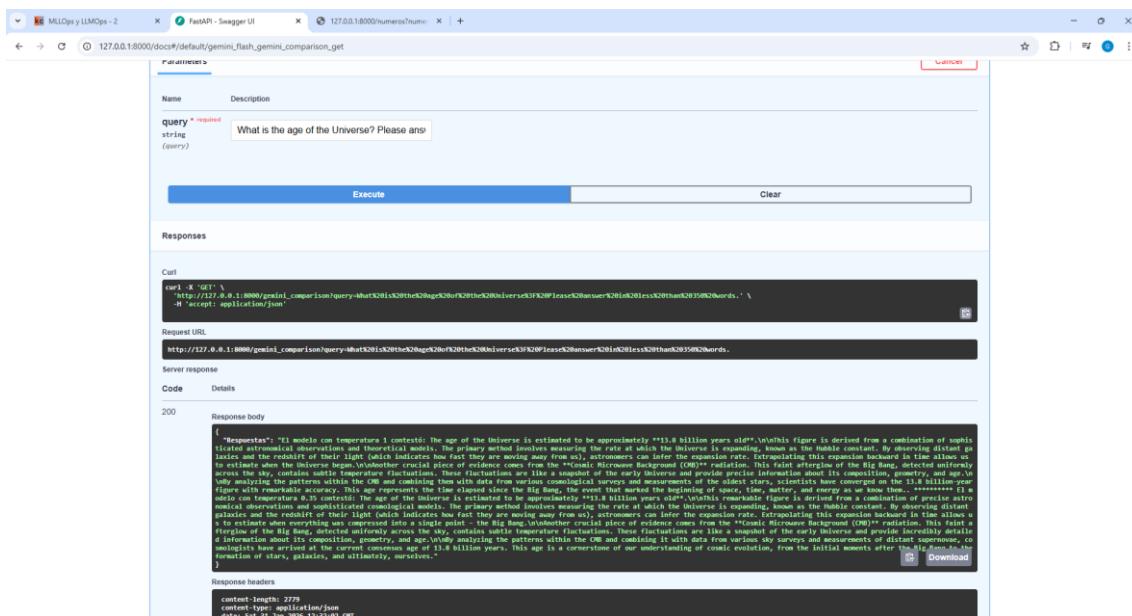


The screenshot shows the FastAPI - Swagger UI interface. A POST request is selected for the endpoint `/random_forest_regressor`. The parameters `depth` is set to `8` and `estimators` is set to `200`. The response body shows a JSON message: `{ "Message": "Los R2 para train y test son 0.766 y 0.701, mientras que los RMSE son 22.4 y 25.0, respectivamente." }`. Response headers include `Content-Type: application/json`, `Date: Sat, 31 Jan 2026 09:21:16 GMT`, and `Server: uvicorn`.

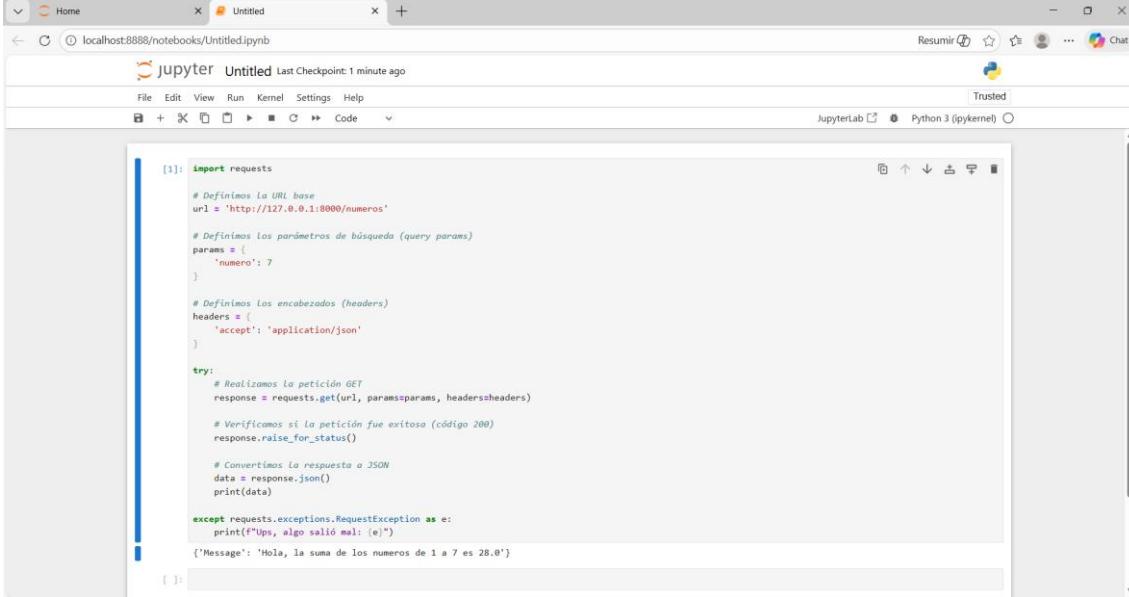
El cuarto módulo hace uso de la pipeline de Hugging Face Zero Shot Classification, a la que hay que introducir un texto. Intenté añadir otras pipelines como NER o traducción, pero me dio errores.



Por último, el quinto módulo ejecuta el modelo Gemini2.5-Flash, para un mismo texto dos veces, pero le variaremos la temperatura en cada caso. Hubo que variar el código bastante, porque ahora la API funciona a base de Client().



A continuación, se muestran las capturas de pantalla de las peticiones http, en el mismo orden.



```
[1]: import requests

# Definimos la URL base
url = 'http://127.0.0.1:8000/numeros'

# Definimos los parámetros de búsqueda (query params)
params = {
    'numero': 7
}

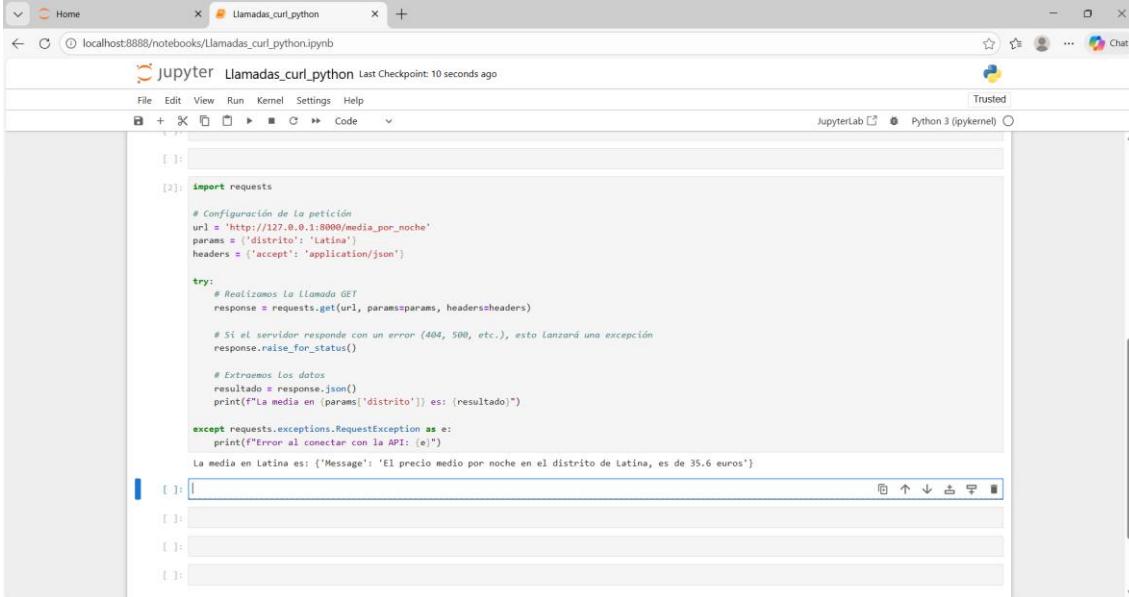
# Definimos los encabezados (headers)
headers = [
    'accept': 'application/json'
]

try:
    # Realizamos la petición GET
    response = requests.get(url, params=params, headers=headers)

    # Verificamos si la petición fue exitosa (código 200)
    response.raise_for_status()

    # Convertimos la respuesta a JSON
    data = response.json()
    print(data)

except requests.exceptions.RequestException as e:
    print(f"Ups, algo salió mal: ({e})")
    print(f"{'Message': 'Hola, la suma de los numeros de 1 a 7 es 28.0'}")
```



```
[2]: import requests

# Configuración de la petición
url = 'http://127.0.0.1:8000/media_por_noche'
params = {'distrito': 'Latina'}
headers = ['accept': 'application/json']

try:
    # Realizamos la llamada GET
    response = requests.get(url, params=params, headers=headers)

    # Si el servidor responde con un error (404, 500, etc.), esto lanzará una excepción
    response.raise_for_status()

    # Extraemos los datos
    resultado = response.json()
    print(f"La media en {params['distrito']} es: ({resultado})")

except requests.exceptions.RequestException as e:
    print(f"Error al conectar con la API: ({e})")
    print(f"{'Message': 'El precio medio por noche en el distrito de Latina, es de 35.6 euros'}")
```

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code uses the requests library to make a GET request to a local endpoint at `http://127.0.0.1:8000/random_forest_regressor`. It defines parameters for depth (8), estimators (150), and accept application/json. A try block handles the response, validating the status and printing the JSON data if successful. An except block catches RequestException and prints an error message. The output cell shows the response message: 'Los R2 para train y test son 0.766 y 0.7, mientras que los RMSE son 22.4 y 25.1, respectivamente.'

```
[3]: # URL del endpoint del modelo
url = 'http://127.0.0.1:8000/random_forest_regressor'

# En este caso tenemos dos parámetros en la query
params = {
    'depth': 8,
    'estimators': 150
}

headers = {
    'accept': 'application/json'
}

try:
    # Realizamos la petición GET con múltiples parámetros
    response = requests.get(url, params=params, headers=headers)

    # Validamos que la respuesta sea correcta
    response.raise_for_status()

    # Obtenemos el resultado (probablemente una predicción o métricas)
    data = response.json()
    print("Respuesta del modelo:", data)

except requests.exceptions.RequestException as e:
    print("Error al consultar el modelo: (e)")

Respueta del modelo: {'Message': 'Los R2 para train y test son 0.766 y 0.7, mientras que los RMSE son 22.4 y 25.1, respectivamente.'}
```

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code. The code uses the requests library to make a GET request to a local endpoint at `http://127.0.0.1:8000/zero-shot-classification`. It defines a parameter `text` with the value 'Los Angeles is the largest city in California, has host many tech companies and professional sport teams.'. The headers are set to accept application/json. A try block handles the response, validating the status and printing the JSON data if successful. An except block catches RequestException and prints an error message. The output cell shows the classification result: 'Value': 'Los Angeles is the largest city in California, has host many tech companies and professional sport teams.', 'labels': ['sports', 'technology', 'geography', 'politics'], 'scores': [0.5985597968101501, 0.3326777517795563, 0.06503484398126602, 0.003727599047124386]}

```
[4]: # URL del endpoint para clasificación zero-shot
url = 'http://127.0.0.1:8000/zero-shot-classification'

# Definimos el texto que queremos clasificar
# Python se encarga de formatear los espacios y comas para la URL
params = {
    'text': 'Los Angeles is the largest city in California, has host many tech companies and professional sport teams.'
}

headers = {
    'accept': 'application/json'
}

try:
    # Realizamos la petición GET
    response = requests.get(url, params=params, headers=headers)

    # Comprobamos si hubo errores
    response.raise_for_status()

    # Procesamos el JSON de respuesta
    resultado = response.json()
    print("Resultado de la clasificación:")
    print(resultado)

except requests.exceptions.RequestException as e:
    print("Hubo un fallo en la conexión: (e)")

Resultado de la clasificación:
{'Value': 'Los Angeles is the largest city in California, has host many tech companies and professional sport teams.', 'labels': ['sports', 'technology', 'geography', 'politics'], 'scores': [0.5985597968101501, 0.3326777517795563, 0.06503484398126602, 0.003727599047124386]}
```

The screenshot shows a Jupyter Notebook interface with the title "Llamadas\_curl\_python". The code cell contains the following Python script:

```
url = "http://127.0.0.1:8990/gemini_comparison"
# Definimos la consulta de forma natural
# requests' convierte automáticamente los espacios y signos en formato URL
params = {
    "query": "What is the age of the Universe? Please answer in less than 350 words."
}

headers = [
    "accept": "application/json"
]

try:
    # Realizamos la petición
    response = requests.get(url, params=params, headers=headers)

    # Validamos si la respuesta es correcta
    response.raise_for_status()

    # Mostramos el resultado de la comparación
    resultado = response.json()
    print("Respuesta de la IA:")
    print(resultado["text"])
except requests.exceptions.RequestException as e:
    print(f"Error al conectar con el servicio: {e}")

Respuesta de la IA:
[{"text": "The primary method involves measuring the rate at which the Universe is expanding, a phenomenon known as the Hubble constant. By observing distant galaxies and how fast they are moving away from us, astronomers can effectively \"readin\" the expansion to estimate when everything was in a much denser state. Another crucial piece of evidence come s from the Cosmic Microwave Background (CMB) radiation. This faint afterglow of the Big Bang, a uniform bath of microwave radiation filling the entire sky, contains subtle temperature fluctuations. These fluctuations are like a snapshot of the early Universe and provide incredibly precise information about its composition, geometry, and age. Only analyzing the patterns within the CMB and combining it with data from various telescopes and surveys, scientists have been able to determine the Universe's age to within 1% error. This remarkable figure is estimated to be approximately 13.8 billion years old. El modelo con temperatura 0.35 contestó: The age of the Universe is estimated to be approximately 13.8 billion years old."}]
```