Despliegue y Monitoreo de un Modelo de Machine Learning para Clasificación de Clientes

Grupo 4

Integrantes: Tomás Azema Di Pizio, Leonardo Rivadeneira, Priscila Moreno.

Profesor: Diego Mosquera

Materia: Laboratorio de Minería de Datos

Comisión: Martes de 19 a 21 hs



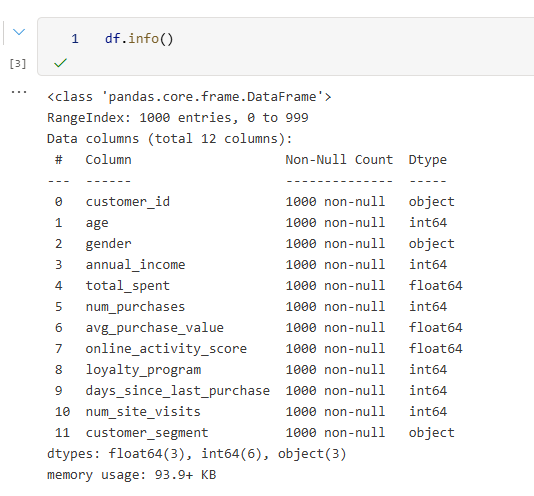
### 

### Contexto

Una empresa minorista quiere implementar un sistema de recomendación de productos personalizado. El primer paso es clasificar a los clientes en diferentes categorías (por ejemplo, "alto valor", "frecuente", "ocasional", etc.) basándose en datos históricos de compras y comportamiento de los usuarios. Una vez que se entrena el modelo de clasificación, debe ser desplegado para que el equipo de marketing pueda usarlo en tiempo real.

### 1. Análisis y preprocesamiento de datos

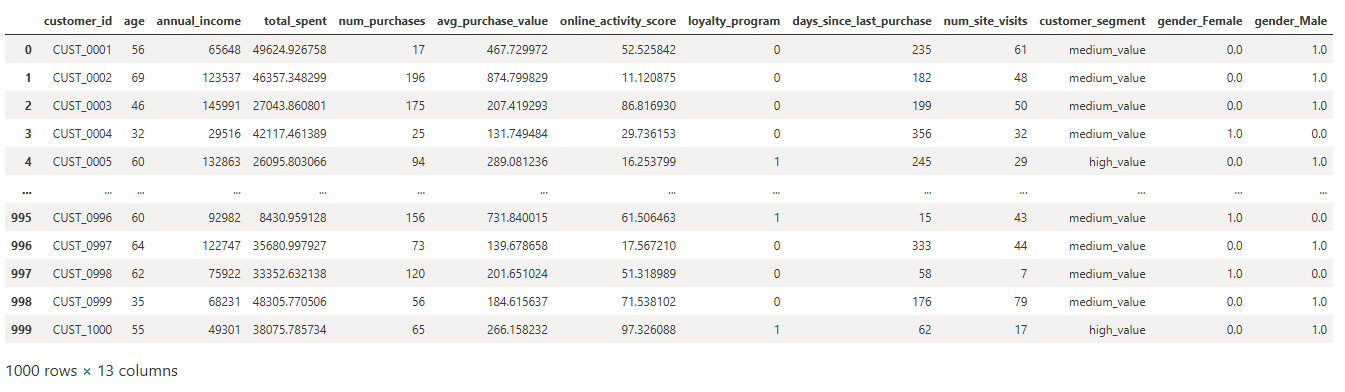
En el marco del contexto explicado en el punto anterior, se analizan los datos existentes en el archivo “*synthetic\_customer\_data.csv*”. Utilizando un cuaderno de jupyter notebook, en primer lugar se describe la estructura del set de datos:



El **último campo**, llamado *customer\_segment*, representa el valor que tiene cada cliente para la empresa, indicado como bajo, medio o alto (*low*, *medium* o *high*). Esta será nuestra variable independiente, es decir, aquella **variable a predecir por el modelo**. Con excepción de *customer\_id*, **el resto de columnas serán las columnas de entrada** que recibirá nuestro modelo de machine learning, nuestras variables dependientes.

**Luego de confirmar que no hay valores nulos o filas duplicadas**, **se procede a realizar una transformación de la columna *gender***. **Se transforma la columna de categórica a numerica con** OneHotEncoder. Como resultado, se obtienen dos columnas nuevas: una llamada *gender\_female* y la otra *gender\_male*. A modo de bandera, cuando uno de estos campos contiene un “1” significa que el cliente del registro pertenece a dicho género, y por ende, se anotará un “0” en la otra columna.

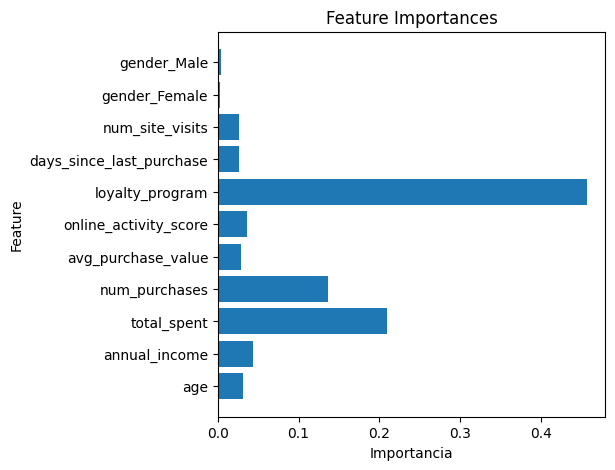
Así nos quedaría el conjunto de datos con el cual el modelo será entrenado.



### 2. Entrenamiento del modelo

Se divide el set de datos, creándose dos subconjuntos: uno para entrenamiento y el otro para testing, con una proporción de 80 % y 20 % respectivamente. **Se desarrolla un modelo de clasificación de bosque aleatorio**, instanciando a RandomForestClassifier() de scikit-learn. A continuación, se entrena, se valida y se evalúa el modelo.

**El modelo obtiene muy buenos resultados, destacando un 98% de precisión**. Al revisar y visualizar la importancia de cada “feature” (cada columna o campo en este caso), podemos ver que campo es más importante para el modelo.



**Se procede a hacer una revisión de los hiper parámetros del modelo con el objetivo de mejorar su rendimiento**. En este caso, el modelo utiliza el índice de gini como criterio; **se procede a utilizar el índice de entropía como nuevo criterio** con el fin de obtener mejores resultados, o confirmar si el criterio por defecto ofrece el mejor rendimiento posible.

**Si bien había poco margen de mejora, el modelo ahora obtiene un 99,5% de precisión**, además de mejorar en otras métricas como el *recall* o el *f1-score*.

### 3. Empaquetado y despliegue del modelo

Utilizando la librería joblib, se serializa y exporta el modelo en un archivo pickle (de extensión .pkl) llamado “*random\_forest\_model.pkl*”. Este archivo es el que después se utilizará para el despliegue del modelo.



Previo a registrar y desplegar el modelo en Azure, hacemos una API en nuestro cuaderno jupyter notebook (que para nosotros es nuestro área de desarrollo). En el mismo, se utilizará la librería flask para crear una API RestFul.

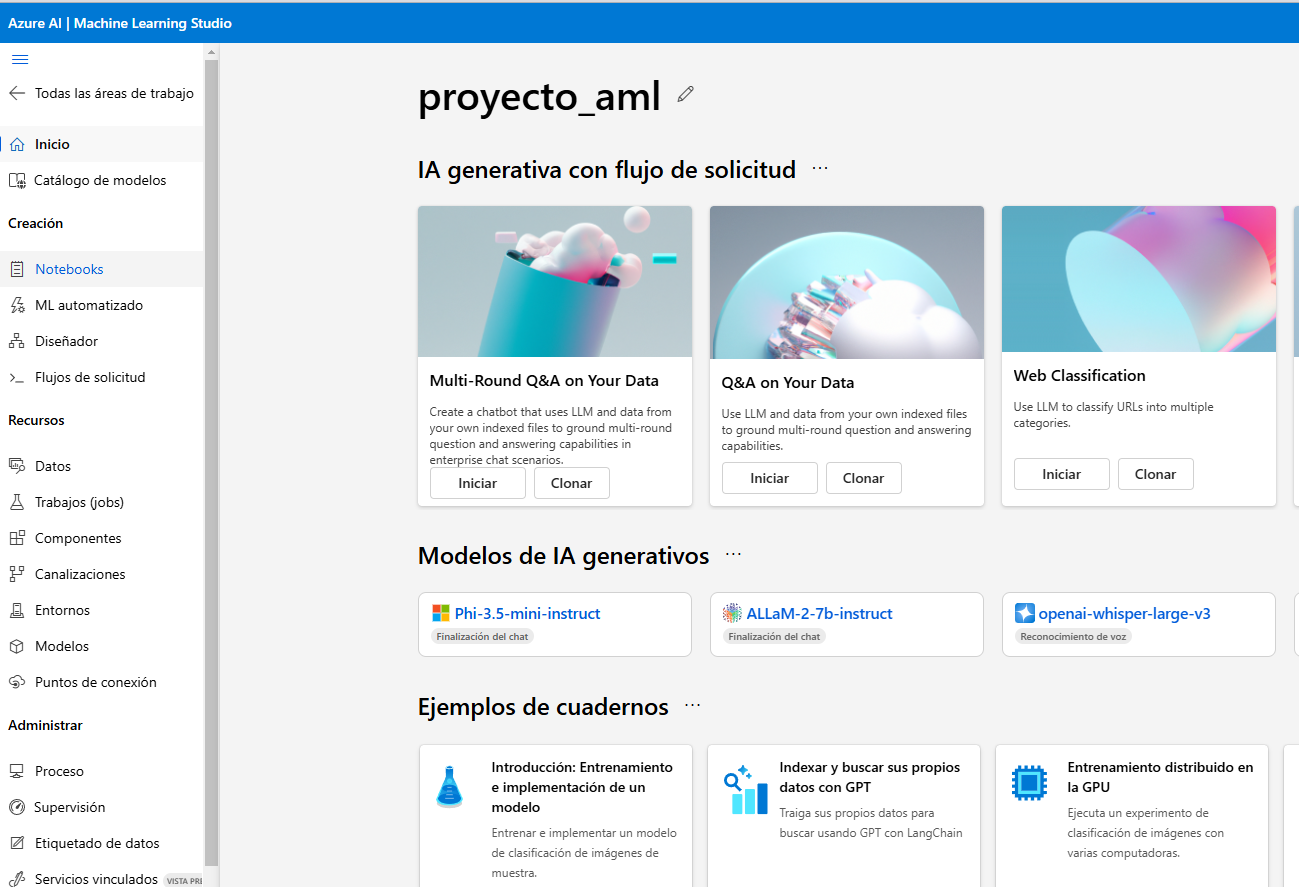
Entre otras cosas, aquí se practicó el hecho de como los datos seran entregados al modelo y devueltos por la API. Por ejemplo, los datos de entrada poseerán al genero como una variable de texto, que deberá ser transformada a dos columnas numéricas por la función de predicción.

Se realizan dos pruebas y en ambos casos se obtiene un código 200: no solo se obtiene una predicción, sino que también esta misma es “razonable” en base al criterio proporcionado.

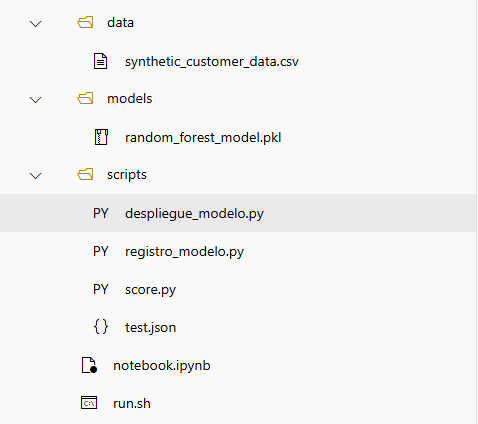




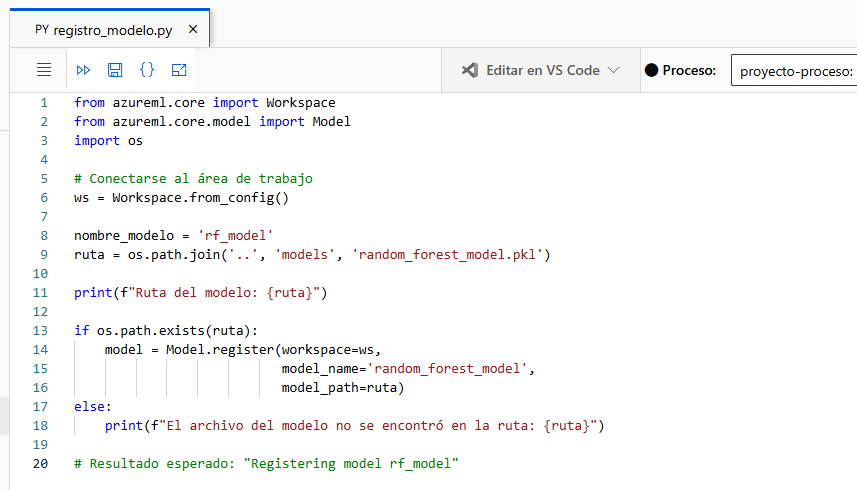
Concluída esta prueba, ahora si estamos en condiciones de registrar y desplegar el modelo. **Hasta el momento veníamos trabajando en jupyter notebook alojado en Google Colab. Ahora, procederemos a utilizar Azure**, creándose un área de trabajo de Machine Learning (ML Studio o AML) y un grupo de recursos. Se subirán o crearán los archivos necesarios y se creará una instancia de proceso.

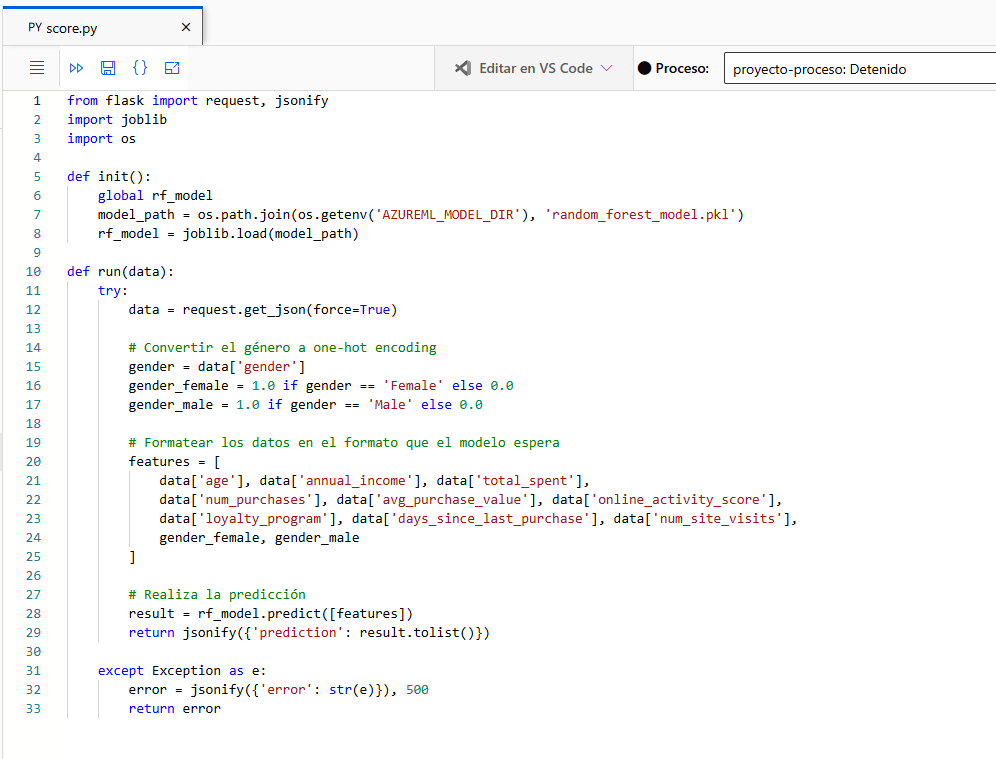


Los archivos de programación se utilizarán por separado (en vez de un solo notebook). En todos los archivos, va a ser necesario estar conectado al area de trabajo utilizando la siguiente línea de python: ws = Workspace.from\_config().

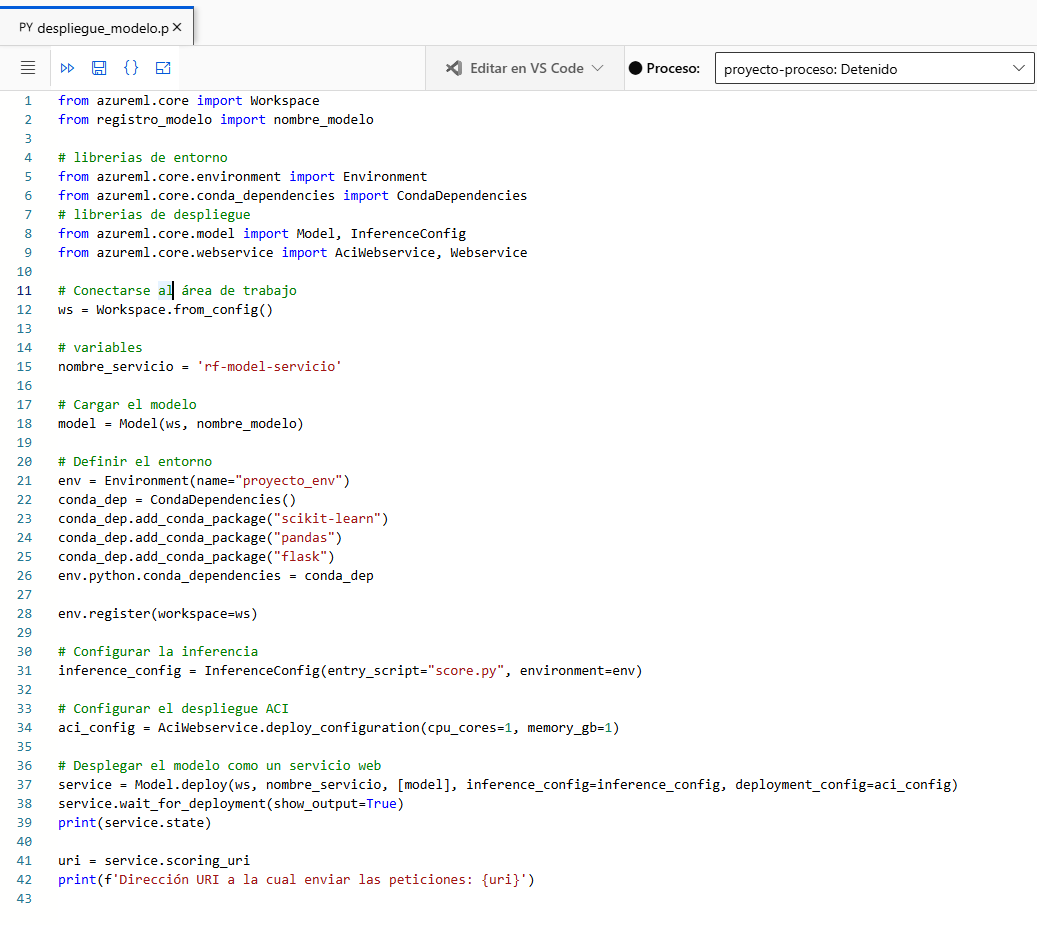


En primer lugar, se ejecutará el archivo “**registro\_modelo.py**”, que, como bien indica el nombre, registrará el modelo en el servicio de Azure ML Studio.



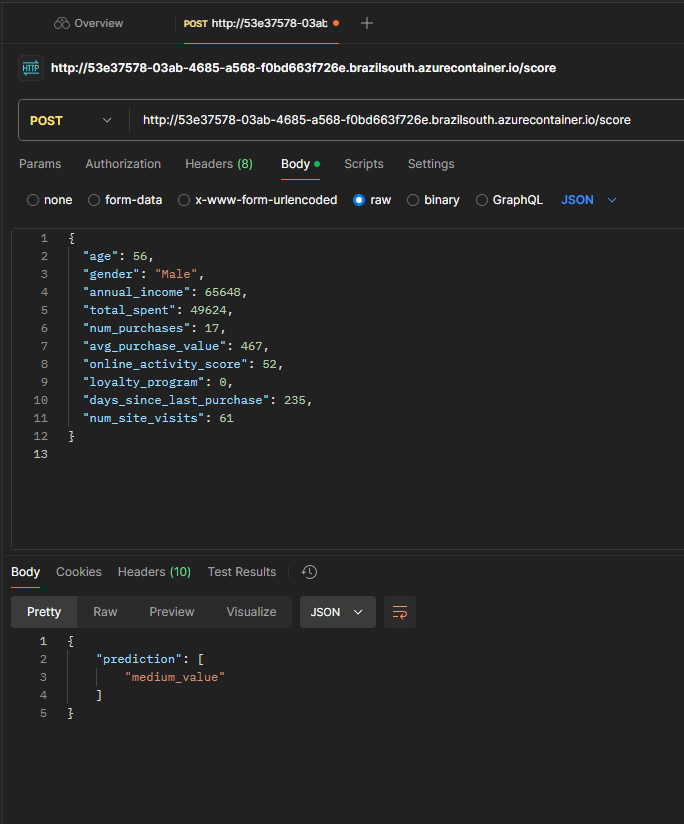
Una vez registrado el modelo, nos aseguraremos de tener el archivo “**score.py**” subido a la plataforma. Aquí dentro es donde desarrollaremos la implementación del modelo y el formato de la entrada de sus parámetros. Es similar al que se encuentra en el notebook, solo que este esta diseñado para su posterior implementación en un punto de conexión de Azure.

En última instancia, se ejecutará el archivo “**despliegue\_modelo.py**”. Este creará nuestro punto de conexión en donde podremos envíar peticiones al modelo para esperar una respuesta. Se define un entorno de azureml.core.environment y se agregarán las dependencias necesarias. Se ajustan dos elementos de configuración: InferenceConfig y AciWebservice.



**Se realizaron pruebas y comprobaciones finales utilizando el servicio de** [**Postman**](https://www.postman.com/). Se le pasó la URL generada por Azure para envíar peticiones, y una petición como prueba.

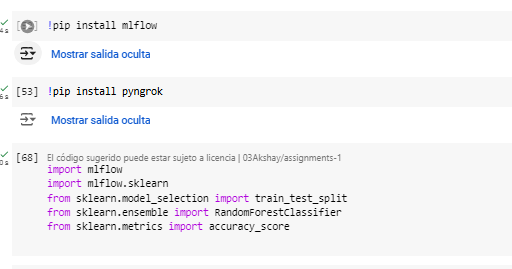
Se obtuvo como resultado un código 200, indicando que no hubo errores en el transcurso. El valor devuelto es una predicción, con el segmento del cliente como resultado.



### 4. Monitoreo y mantenimiento del modelo

En esta etapa del proyecto se implementa el seguimiento y la gestión del modelo utilizando **MLflow**. Para permitir el acceso remoto a la interfaz de usuario de MLflow, se utiliza un túnel seguro configurado con **ngrok**.

**Bibliotecas Utilizadas**



**!pip install mlflow:** Se utiliza para el seguimiento de modelos de aprendizaje automático, el registro de modelos y la implementación.

**!pip install pyngrok:** Permite crear túneles seguros a localhost, lo que es útil para compartir la interfaz de usuario de MLflow que se ejecuta localmente.

**import mlflow, import mlflow.sklearn:** Son necesarias para interactuar con MLflow y registrar modelos de scikit-learn.

**from sklearn.model\_selection import train\_test\_split, from sklearn.ensemble import RandomForestClassifier, from sklearn.metrics import accuracy\_score:**Importa las funciones necesarias de scikit-learn para dividir datos, entrenar un modelo RandomForest y calcular la precisión.

**Configuración de MLflow**

**mlflow.set\_experiment('proyecto'):** Establece el modelo en MLflow. Todos los runs subsiguientes se registrarán bajo este modelo.

**if mlflow.active\_run() is not None: mlflow.end\_run():** Esta línea es importante para asegurar que no haya ejecuciones activas antes de comenzar una nueva. Si existe una ejecución activa, la termina. Esto evita conflictos y asegura que cada ejecución se registre correctamente.

**Entrenamiento y registro del modelo**



**with mlflow.start\_run():** Inicia una nueva ejecución en MLflow. Todo el código dentro de este bloque se registrará como parte de esta ejecución.

**rf\_model = RandomForestClassifier(n\_estimators=10):** Crea un clasificador RandomForest con 10 árboles.

**rf\_model.fit(X\_train, y\_train):** Entrena el modelo con los datos de entrenamiento (X\_train, y\_train).

**mlflow.log\_param("n\_estimators", 10**): Registra el hiperparámetro "n\_estimators" en MLflow.

**y\_pred = rf\_model.predict(X\_test):** Realiza predicciones en el conjunto de prueba.

**accuracy = accuracy\_score(y\_test, y\_pred):** Calcula la precisión del modelo.

**mlflow.sklearn.log\_model(rf\_model, "RandomForestModel"):** Guarda el modelo entrenado en MLflow. Esto permite la gestión de versiones del modelo y la implementación posterior.

**Configuración y ejecución de ngrok**



**get\_ipython().system\_raw("mlflow ui --port 5000 &"):** Inicia la interfaz de usuario de MLflow en el puerto 5000 en segundo plano. El & permite que el script continúe ejecutándose sin bloquearse en la interfaz de usuario.

**from pyngrok import ngrok:** Importa la biblioteca ngrok.

**ngrok.kill():** Mata cualquier túnel ngrok existente para evitar conflictos.

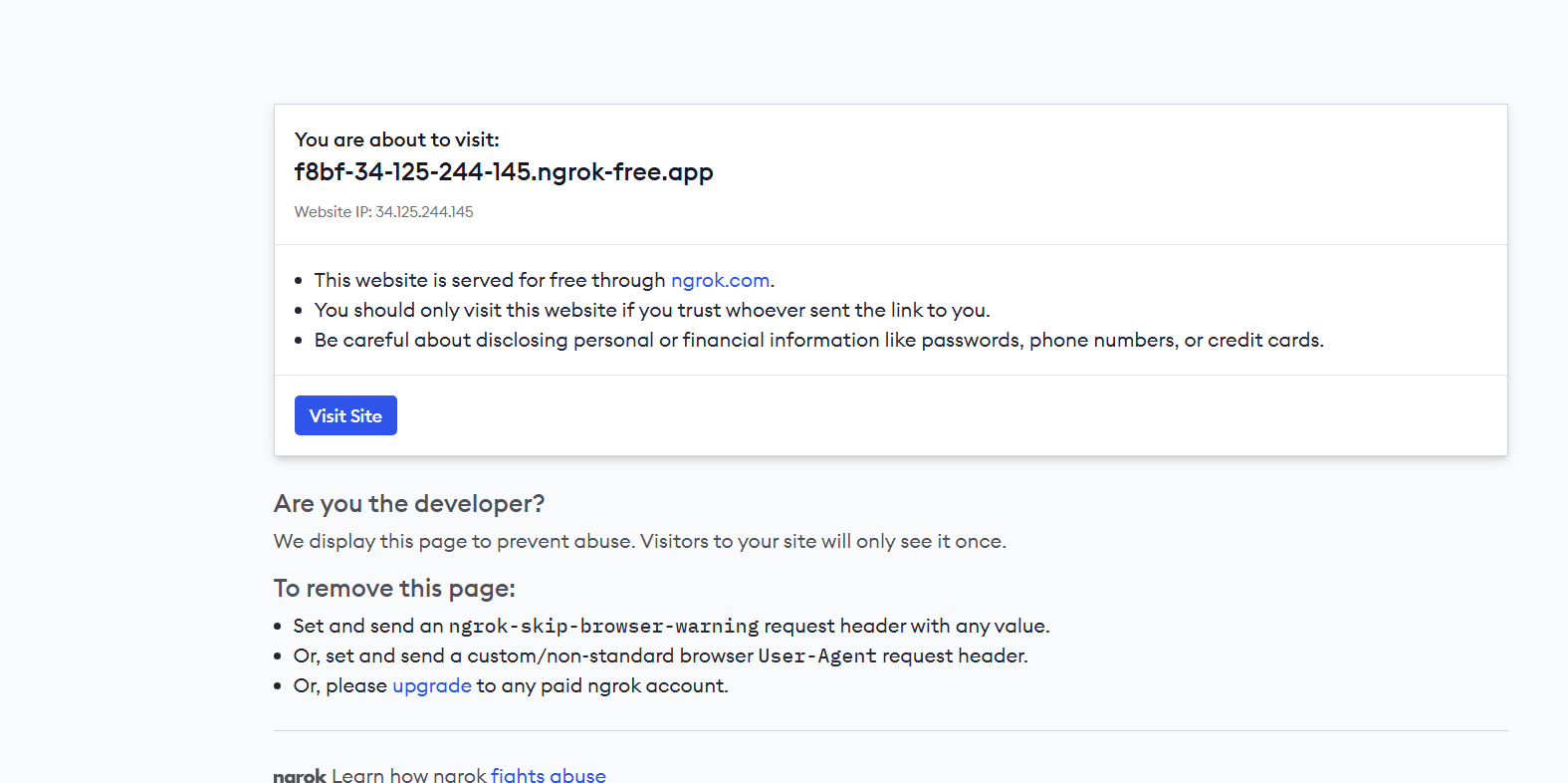
**NGROK\_AUTH\_TOKEN =** "2p2khEow4fR9Jyb9mIIWfDwPFj4\_7hFMxUFE2PXQBCBRNxcL3": Define el token de autenticación de ngrok. Necesitas obtener tu propio token del sitio web de ngrok.

**ngrok.set\_auth\_token(NGROK\_AUTH\_TOKEN):** Configura el token de autenticación.

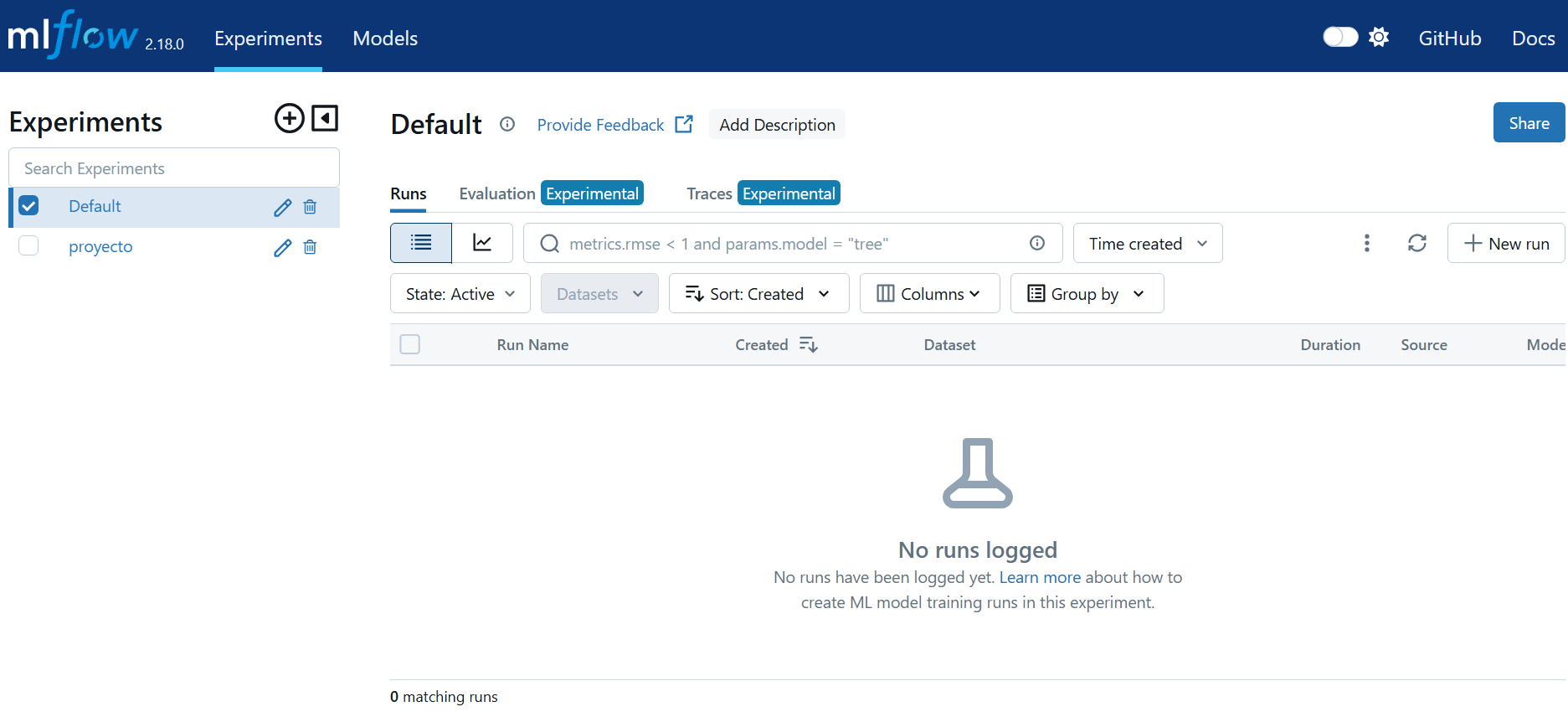
**ngrok\_tunnel = ngrok.connect(addr="5000", proto="http", bind\_tls=True):** Crea un túnel a localhost:5000 (donde se ejecuta la interfaz de usuario de MLflow) usando HTTP con TLS.

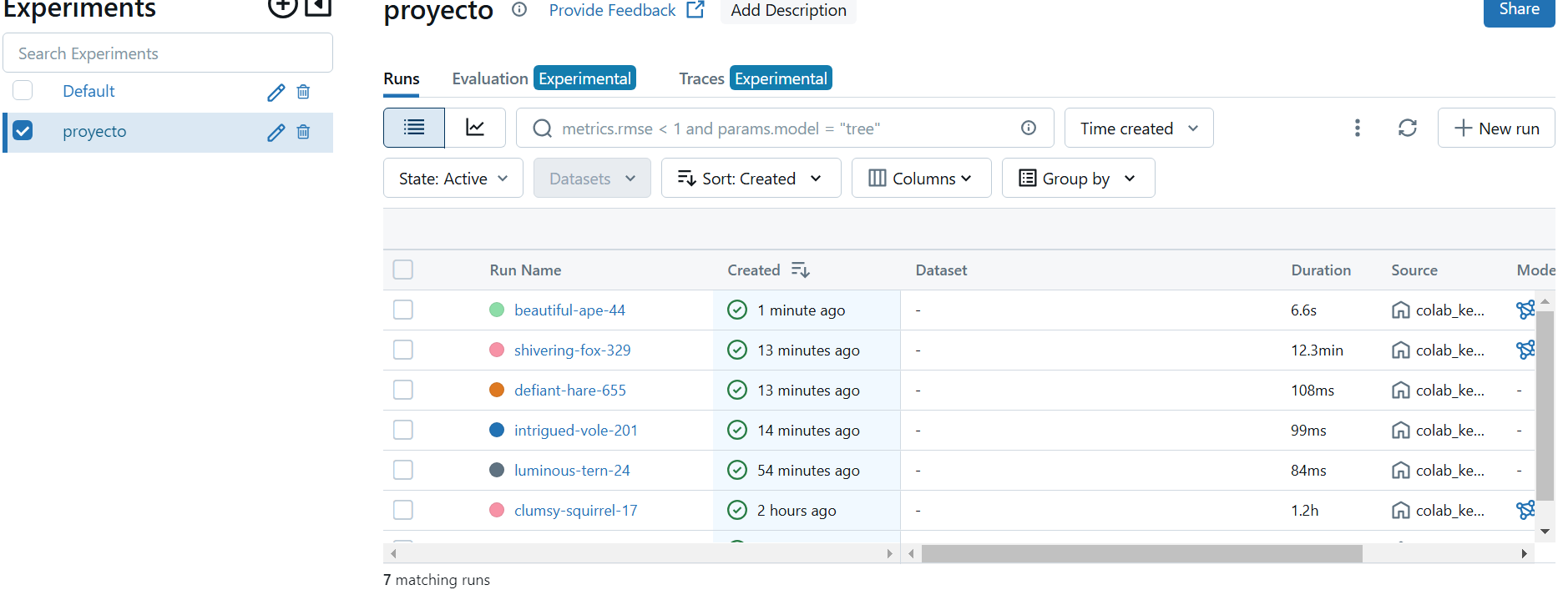
**print("Url para MLFLOW: ", ngrok\_tunnel.public\_url):** Imprime la URL pública generada por ngrok. Esta URL se puede utilizar para acceder a la interfaz de usuario de MLflow desde cualquier lugar.

[https://b0c5-34-125-244-145.ngrok-free.app](https://b0c5-34-125-244-145.ngrok-free.app/)



MlFlow





Model Metrics

