Proyecto

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

**Integrantes del Grupo**

Ivana Porto

Roberto García

Marianela Pi

Índice

Contenido

[Desarrollo del Proyecto 3](#_Toc184230507)

[Exploración y Preprocesamiento de Datos 3](#_Toc184230508)

[Entrenamiento del Modelo 4](#_Toc184230509)

[Empaquetado y Despliegue del Modelo 6](#_Toc184230510)

[Implementación de una API en Fast: 6](#_Toc184230511)

[Despliegue en AWS SageMaker: 7](#_Toc184230512)

[Postman 8](#_Toc184230513)

[Monitoreo y Mantenimiento del Modelo 8](#_Toc184230514)

# Desarrollo del Proyecto

Notebook del proyecto

<https://raw.githubusercontent.com/Marian2057/Proyecto_Laboratorio2/refs/heads/main/Notebook-Final-Lab.ipynb>

## Exploración y Preprocesamiento de Datos

##### Carga de Datos:

* + Se cargó el archivo synthetic\_customer\_data.csv desde un bucket de S3 usando Pandas. Este archivo contiene información demográfica, comportamientos de compra, y datos de navegación de los clientes.

##### Análisis Inicial:

* + Se utilizaron los métodos .info() y .head() del dataframe de pandas para, por un lado, verificar su estructura y detectar la existencia o inexistencia de valores nulos y, por el otro, observar y analizar el contenido de los primeros 5 registros del mismo.
  + Se realizó un análisis estadístico básico usando el método .describe() para entender la distribución de las variables numéricas.

##### Limpieza de Datos:

* + **Tratamiento de Valores Faltantes:** Atento que el método .info() del dataframe de pandas informó que no hay valores nulos, no es necesario realizar ningún tratamiento.
  + **Normalización y Estandarización:** Se procedió a estandarizar las variables numéricas age, annual\_income, total\_spent, num\_purchases, avg\_purchase\_value, online\_activity\_score, days\_since\_last\_purchase y num\_site\_visits usando la técnica de preprocesamiento StandardScaler de la librería scikit-learn. Esta técnica funciona según el principio de normalización, en el que transforma la distribución de cada característica para que tenga una media de cero y una desviación estándar de uno, garantizando que todas las características estén en la misma escala, lo que evita que una sola característica domine el proceso de aprendizaje debido a su mayor magnitud.

##### Codificación de Variables Categóricas:

* + La variable gender se codificó a valores numéricos con la función map, asignando el valor 0 para “Male” y 1 para “Female”, atento que el modelo elegido, al ser un clasificador, no trabaja con variables categóricas.
  + Por las razones expuestas en el ítem precedente, se procedió a transformar (con la misma función), la variable customer\_segment asignando en este aco los valores 0 para "low\_value", 1 para "medium\_value", y 2 para "high\_value").

##### Análisis de Outliers:

* + Se utilizaron gráficos de cajas (boxplots) para identificar outliers en las variables clave. Ellos nos permitieron observar la inexistencia de valores extremos. Por lo tanto, no fue necesario efectuar ninguna corrección.

##### División de Datos:

* + Los datos se dividieron en conjuntos de entrenamiento (80%) y validación (20%) usando la función train\_test\_split de sklearn. Esto asegura una evaluación precisa en un conjunto de datos independiente. Paralelamente, al parámetro random\_state de esta función, le asignamos una semilla con el valor aleatorio 42, para que el generador de números aleatorios permita reproducir la función en forma idéntica cada vez que se reentrene el modelo.

##### Inicialización de SageMaker y Carga de Datos a S3

* + Se guardaron los datos en formato LIBSVM que facilita tanto la preparación de los datos como el entrenamiento de los modelos en XGBoost.

## Entrenamiento del Modelo

##### Configuración del Entorno de SageMaker:

* + Se definió el bucket de S3 para almacenar los datos y el modelo, y se inicializó una sesión de SageMaker para gestionar los recursos en AWS.

##### Selección del Algoritmo e Hiperparámetros:

* + Se eligió el modelo XGBoost de SageMaker, que es un clasificador multiclase, adecuado para la tarea de segmentación. En este caso para predecir en función a la categoría del cliente.

##### Ajuste de Hiperparámetros:

* + Se utilizaron los siguientes hiperparámetros de Sagemaker para mejorar así la precisión del modelo.

objective="multi:softmax":

Define la tarea del modelo como una clasificación multiclase.

El modelo devolverá la clase con mayor probabilidad.

num\_class=3:

Especifica que hay 3 clases posibles en la clasificación.

eta=0.1 (Learning Rate):

Controla la velocidad de aprendizaje del modelo.

Valores más pequeños hacen que el modelo entrene más lentamente pero con mayor precisión.

max\_depth=5:

Determina la profundidad máxima de los árboles.

Controla la complejidad del modelo y reduce el riesgo de sobreajuste.

subsample=0.8:

Especifica que el modelo usará el 80% de los datos de entrenamiento en cada iteración.

Esto ayuda a prevenir el sobreajuste.

colsample\_bytree=0.8:

Define que el modelo usará el 80% de las características disponibles para construir cada árbol.

num\_round=100:

Especifica el número de iteraciones (árboles) que construirá el modelo.

##### Entrenamiento y Evaluación:

* + El modelo fue entrenado en SageMaker y evaluado usando las métricas accuracy, precision, recall, y F1-score en el conjunto de validación. Esto permitió cuantificar el rendimiento del modelo y ajustar posibles desbalances en las predicciones.

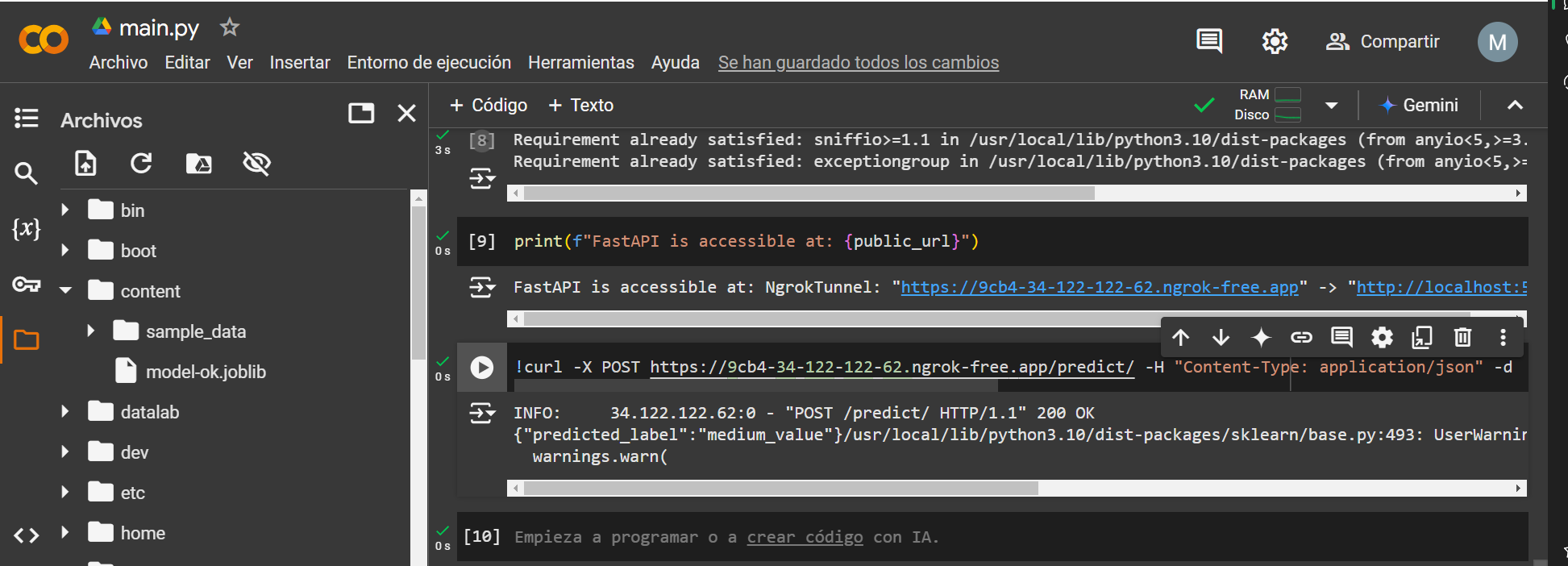
## Empaquetado y Despliegue del Modelo

##### Exportación del Modelo:

* + El modelo entrenado fue exportado a formato xgb, un formato estándar para modelos en Python, lo que facilita su reutilización y despliegue en producción.

### Implementación de una API en Fast:

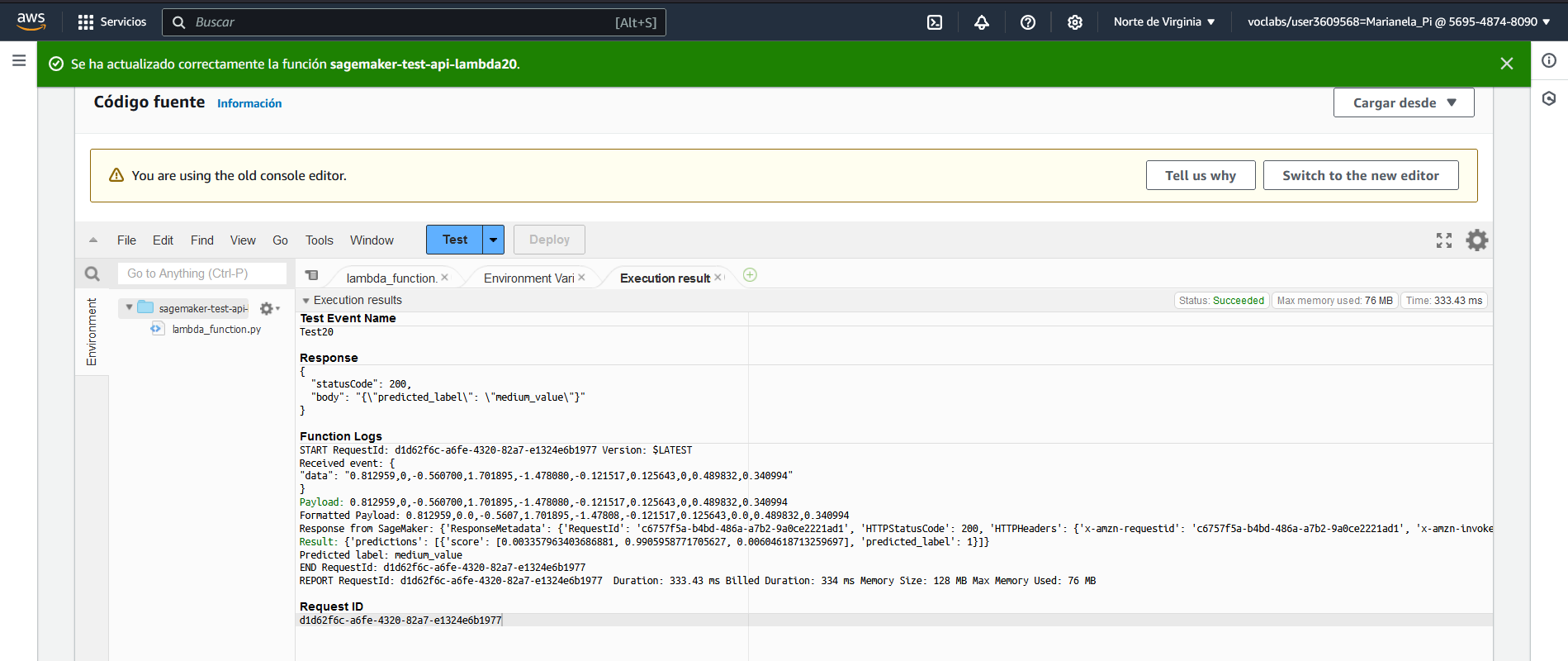
* + Se creó una API con Fast para interactuar con el modelo de clasificación. Esta API permite que el equipo de marketing envíe datos de clientes y reciba predicciones en tiempo real.
  + La API fue diseñada para recibir datos en formato JSON, procesarlos y devolver la categoría de cliente ((0 para "low\_value", 1 para "medium\_value", y 2 para "high\_value").
  + Estructura: curl -X POST https://a68c-35-204-238-224.ngrok-free.app/predict/ -H "Content-Type: application/json" -d '{"data": [0.812959, 0, -0.560700, 1.701895, -1.478080, -0.121517, 0.125643, 0, 0.489832, 0.340994]}



Código fastApi: <https://raw.githubusercontent.com/Marian2057/Proyecto_Laboratorio2/refs/heads/main/fastapi.py>

### Despliegue en AWS SageMaker:

* + El modelo empaquetado fue desplegado en un endpoint de SageMaker, utilizando una instancia de ml.t2.medium para equilibrar costos y rendimiento.
  + La función Lambda se configuró para gestionar las solicitudes de la API, asegurando que los datos se formatearan correctamente antes de llegar al modelo.



Código Función Lambda:

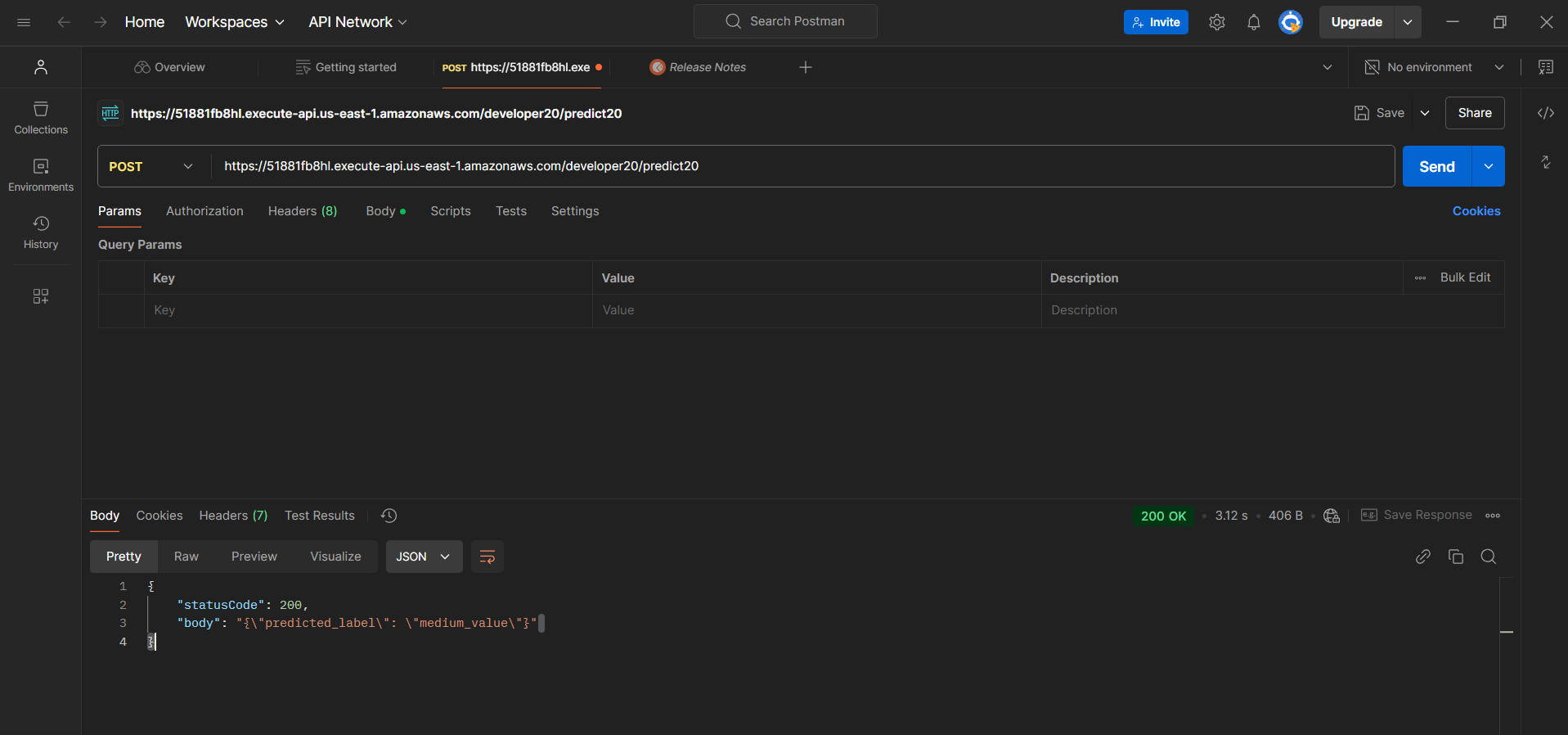
<https://raw.githubusercontent.com/Marian2057/Proyecto_Laboratorio2/refs/heads/main/funcion_Lambda.txt>

### Postman

Método POST: <https://51881fb8hl.execute-api.us-east-1.amazonaws.com/developer20/predict20>

Código json

* + {"data": [0.812959, 0, -0.560700, 1.701895, -1.478080, -0.121517, 0.125643, 0, 0.489832, 0.340994]}



## Monitoreo y Mantenimiento del Modelo

El monitoreo en AWS (Amazon Web Services) permite rastrear el rendimiento, la disponibilidad y el uso de los recursos de la infraestructura en la nube para asegurarse de que todo funcione correctamente. AWS ofrece herramientas como Amazon CloudWatch para monitorear métricas en tiempo real y AWS X-Ray para realizar un seguimiento de las aplicaciones.

Las métricas más importantes para mostrar en un dashboard de monitoreo de AWS son:

Uso de CPU: Mide el porcentaje de uso de la CPU de una instancia EC2. Una alta utilización puede indicar que se necesita más capacidad de procesamiento.

Memoria: El uso de memoria de las instancias EC2 es crucial para detectar problemas de rendimiento o de capacidad.

Disco: Métricas como el uso de disco (lecturas/escrituras) y la latencia del disco son importantes para monitorear la capacidad de almacenamiento y la velocidad de acceso.

Red: Se deben mostrar las métricas de tráfico de red (entradas y salidas) para detectar posibles cuellos de botella o problemas de conectividad.

Latencia de API Gateway: Monitorear la latencia de las solicitudes a través de API Gateway es importante para asegurar tiempos de respuesta rápidos.

Solicitudes y errores: Métricas como el número de solicitudes y errores (por ejemplo, 4xx/5xx) en servicios como Lambda, API Gateway o ALB.

Estado de instancias EC2: Información sobre las instancias EC2, como el estado de salud (en ejecución, detenido, etc.), es vital para la administración operativa.

Estas métricas permiten detectar problemas en tiempo real y ajustar recursos de manera eficiente para mantener un alto rendimiento y disponibilidad de la infraestructura en AWS.

