# **Proyecto: Validador Inteligente de Documentos de Venta con IA**

**Nombre:** Víctor Sandoval Gómez  
**Curso:** Inteligencia Artificial Generación de Prompts  
**Comisión:** 84185

## **Introducción**

### **Nombre del proyecto**

Validador Inteligente de Documentos de Venta con IA.

### **Presentación del problema a abordar**

En el competitivo sector de la venta de servicios agrícolas, la eficiencia y la precisión del ciclo de ventas son cruciales. Actualmente, la empresa enfrenta un cuello de botella significativo: la validación de documentos. Cuando un ejecutivo de preventa ingresa los datos de una oportunidad de negocio en el CRM (como el modelo superficie, precios, RUT del cliente, etc.), se genera una oferta que debe ser verificada manualmente contra los datos del sistema.

**Esta problemática es relevante por varias razones:**

1. **Ineficiencia Operativa:** La validación manual es un proceso lento y repetitivo que consume valiosas horas de trabajo del personal administrativo o de ventas. Este tiempo podría dedicarse a tareas de mayor valor, como la captación de nuevos clientes o la gestión de relaciones.
2. **Riesgo de Error Humano:** La revisión manual es propensa a errores. Un dígito incorrecto en un precio, un RUT mal transcrito o un modelo de vehículo equivocado pueden generar graves problemas: desde la insatisfacción del cliente hasta pérdidas financieras y complicaciones legales.
3. **Impacto en la Experiencia del Cliente:** Los retrasos causados por la validación manual alargan el tiempo de respuesta para la entrega de una oferta formal. En un mercado donde la agilidad es clave, un proceso lento puede resultar en la pérdida de la oportunidad de negocio frente a un competidor más rápido.

Desarrollar una solución para automatizar este proceso no solo optimiza los recursos internos, sino que también mejora la fiabilidad y la velocidad del servicio, impactando directamente en la competitividad de la empresa.

### **Desarrollo de la propuesta de solución**

La solución propuesta se basa en el desarrollo de un sistema de **Procesamiento Inteligente de Documentos (IDP)** utilizando un modelo de lenguaje avanzado (LLM) como Google Gemini, implementado a través de un script de Python en un entorno de Jupyter Notebook.

El flujo de trabajo será el siguiente:

1. **Entrada de Datos:** El sistema recibirá dos entradas principales:
   * **Datos del CRM:** Un conjunto de datos estructurados (en formato JSON) que simulan la información ingresada por el ejecutivo de preventa.
   * **Documento de Oferta:** El texto de un documento (simulando un PDF o un contrato) que debe ser validado.
2. **Procesamiento con IA:** Un prompt cuidadosamente diseñado (utilizando técnicas de *Fast Prompting*) instruirá al modelo de IA para que realice las siguientes acciones en una **única consulta a la API**:
   * **Actuar como un experto:** Asumirá el rol de un "auditor de documentos".
   * **Extraer Entidades:** Leerá el texto del documento y extraerá los valores específicos correspondientes a los campos del CRM (ej: extraer el precio total, el nombre del cliente, el VIN del vehículo, etc.).
   * **Comparar y Validar:** Comparará cada valor extraído del documento con el valor correspondiente de los datos del CRM.
   * **Generar un Reporte Estructurado:** Devolverá un resultado en formato JSON que indique claramente qué campos coinciden (True), cuáles no (False), y mostrará tanto el valor esperado como el valor encontrado.
3. **Acción Final (Opcional en la POC):** Si se detectan discrepancias, el sistema podría ser extendido para enviar automáticamente un correo electrónico de notificación a los responsables, adjuntando el reporte de validación.

## **Justificación de la viabilidad del proyecto**

La viabilidad de este proyecto es alta por las siguientes razones:

* **Viabilidad Técnica:** Los modelos de lenguaje modernos, como la familia Gemini de Google, han demostrado una capacidad sobresaliente para la comprensión de texto no estructurado y la extracción de entidades (Named Entity Recognition - NER). La tarea de leer un contrato y extraer datos específicos es un caso de uso central para estas tecnologías. Las librerías de Python como google-generativeai simplifican enormemente la interacción con la API, haciendo que la implementación sea directa.
* **Recursos Disponibles:** El proyecto se desarrollará como una Prueba de Concepto (POC) en un notebook de Google Colab, que proporciona el entorno de ejecución y la potencia de cálculo necesarios sin costo. La API de Gemini ofrece un generoso nivel gratuito que es más que suficiente para el desarrollo y la demostración de esta POC.
* **Tiempo:** El alcance del proyecto está acotado a la creación de un script funcional en un notebook, lo cual es perfectamente realizable en el marco de tiempo del curso. No se busca crear una aplicación de producción completa, sino demostrar la efectividad del concepto.

## **Objetivos**

* **Objetivo General:** Desarrollar una Prueba de Concepto (POC) funcional que automatice la validación de datos entre un sistema CRM y un documento de oferta de venta, utilizando técnicas de *Fast Prompting* con un modelo de IA.
* **Objetivos Específicos:**
  1. Diseñar un prompt robusto y eficiente que instruya a la IA para extraer y comparar datos en una sola llamada a la API.
  2. Implementar la solución en un Jupyter Notebook utilizando Python y la API de Google Gemini.
  3. Validar la precisión del modelo en la extracción de diferentes tipos de datos (texto, números, identificadores).
  4. Demostrar la capacidad del sistema para identificar correctamente tanto las coincidencias como las discrepancias entre las dos fuentes de datos.

## **Metodología**

El proyecto se llevará a cabo siguiendo estos pasos:

1. **Definición del Caso de Uso:** Se creará un escenario realista de venta de un vehículo corporativo, incluyendo un documento de contrato de ejemplo y un conjunto de datos simulados del CRM.
2. **Preparación del Entorno:** Se configurará un notebook en Google Colab, instalando las librerías necesarias y gestionando la clave de la API de forma segura mediante los "Secrets" de Colab.
3. **Diseño de Prompt (Fast Prompting):** Se creará un prompt principal aplicando las siguientes técnicas para optimizar el rendimiento y el costo:
   * **Role-Playing:** Se le asignará a la IA el rol de "auditor experto" para contextualizar la tarea y mejorar la calidad de la respuesta.
   * **Structured Output (JSON):** Se le exigirá a la IA que la salida sea un JSON bien formado. Esto es crucial para que la respuesta sea directamente procesable por Python, eliminando la necesidad de análisis de texto complejo (parsing) y reduciendo la posibilidad de errores.
   * **Chain of Thought (implícito):** Se le pedirá a la IA que primero extraiga los datos y luego los compare, guiando su proceso de razonamiento para obtener un resultado más fiable.
4. **Implementación del Código:** Se escribirá el script en Python que:
   * Cargue el texto del documento y los datos a verificar.
   * Construya y envíe el prompt a la API de Gemini.
   * Reciba y procese la respuesta JSON.
   * Imprima un informe de validación claro y legible para el usuario.
5. **Pruebas y Validación:** Se ejecutarán pruebas con datos que coinciden perfectamente y con datos que contienen errores deliberados para verificar que el sistema funciona como se espera en ambos escenarios.

## **Herramientas y tecnologías**

* **Lenguaje de Programación:** Python 3.
* **Entorno de Desarrollo:** Google Colab (Jupyter Notebook).
* **Modelo de IA:** Google Gemini 1.5 Pro.
* **Librerías de Python:** google-generativeai (para interactuar con la API), json (para manejar los datos), re (para normalización de texto).
* **Técnicas de Prompting:**
  + **Role-Playing:** Justificación: Aumenta la fiabilidad y precisión al poner al modelo en un contexto específico.
  + **Structured Output (JSON):** Justificación: Es la técnica más importante para la automatización. Garantiza una salida consistente y legible por máquina, lo que **optimiza drásticamente los costos** al evitar llamadas adicionales a la API para formatear o corregir respuestas.
  + **Chain of Thought:** Justificación: Mejora la capacidad de razonamiento del modelo para tareas complejas de múltiples pasos, como extraer y luego comparar.

## **Implementación**

A continuación, se presenta el código de la Prueba de Concepto (POC).

# 1. INSTALACIÓN Y CONFIGURACIÓN

# --------------------------------------------------------------------------

!pip install -U google-generativeai

import google.generativeai as genai

from google.colab import userdata

import json

import re

# Configura tu API Key desde los Secrets de Colab (icono de la llave 🔑)

try:

GEMINI\_API\_KEY = userdata.get('GEMINI\_API\_KEY')

genai.configure(api\_key=GEMINI\_API\_KEY)

print("✅ API Key configurada correctamente.")

except Exception as e:

print(f"🛑 Error al configurar la API Key: {e}")

raise SystemExit("Deteniendo: La API Key es necesaria para continuar.")

# 2. DATOS DEL ESCENARIO DE VENTA

# --------------------------------------------------------------------------

# Simulación de los datos ingresados en el CRM por el preventa

datos\_crm = {

"Nombre Cliente": "Constructora Rápido S.A.",

"RUT Cliente": "78.555.123-4",

"Modelo Vehículo": "Camioneta Maxus T60 4x4",

"VIN": "LSV3T60A4XF123456",

"Precio Neto CLP": "18.500.000",

"Ejecutivo de Venta": "Ana Martínez"

}

# Texto del documento de oferta/contrato que se debe validar

# Se incluyen errores deliberados para la demostración.

texto\_oferta = """

--- OFERTA COMERCIAL N° OF-9876 ---

FECHA: 15 de Octubre de 2024

CLIENTE:

Razón Social: Constructora Rápido S.A.

RUT: 78.555.123-4

EJECUTIVO ASIGNADO: Ana Martínez

DETALLE DE LA OFERTA:

Por la presente, Automotora Corporativa S.A. tiene el agrado de presentar la siguiente oferta por el vehículo:

- Modelo: Camioneta Maxus T60 4x4

- Número de Chasis (VIN): LSV3T60A4XF123457 <-- ERROR EN EL ÚLTIMO DÍGITO

- Precio Neto: $ 18.500.000 CLP

- Condición: Vehículo nuevo, año 2024.

Esta oferta es válida por 10 días.

Atentamente,

Gerencia Comercial

Automotora Corporativa S.A.

"""

# 3. FUNCIÓN DE VALIDACIÓN CON IA

# --------------------------------------------------------------------------

def normalizar\_valor(valor):

"""Función auxiliar para limpiar y estandarizar los valores antes de comparar."""

if valor is None:

return ""

# Elimina espacios, convierte a minúsculas y quita caracteres que no sean letras, números o guiones.

return str(valor).strip().lower().replace(".", "").replace("$", "").replace(" ", "")

def validar\_documento\_con\_ia(datos\_crm, texto\_documento):

"""

Usa Gemini para extraer datos de un documento, compararlos con datos de un CRM

y devolver un reporte de validación.

"""

print("\n🤖 Contactando a Gemini para validar el documento...")

model = genai.GenerativeModel('gemini-1.5-pro-latest')

# El prompt utiliza las técnicas de Role-Playing y Structured Output (JSON)

prompt = f"""

Eres un auditor de documentos extremadamente meticuloso. Tu tarea es validar que la información de un documento de oferta coincida con los datos de un sistema CRM.

\*\*Datos del CRM (Fuente de la verdad):\*\*

```json

{json.dumps(datos\_crm, indent=2)}

```

\*\*Texto del Documento a Validar:\*\*

---

{texto\_documento}

---

\*\*Instrucciones:\*\*

1. Lee el "Texto del Documento a Validar".

2. Para cada clave en los "Datos del CRM", encuentra el valor correspondiente dentro del documento.

3. Compara el valor encontrado en el documento con el valor del CRM.

4. Devuelve tu análisis ÚNICAMENTE en formato JSON. El JSON debe tener la siguiente estructura: una clave para cada campo del CRM, y para cada clave, un objeto con tres campos: "valor\_crm", "valor\_encontrado\_doc", y "coincide" (un booleano).

Ejemplo de la salida esperada para un campo:

"Nombre Cliente": {{

"valor\_crm": "Constructora Rápido S.A.",

"valor\_encontrado\_doc": "Constructora Rápido S.A.",

"coincide": true

}}

"""

try:

response = model.generate\_content(prompt)

# Limpieza de la respuesta para asegurar que sea un JSON válido

cleaned\_json\_string = response.text.strip().replace('```json', '').replace('```', '')

resultados = json.loads(cleaned\_json\_string)

print("✅ Validación completada exitosamente.")

return resultados

except Exception as e:

print(f"🛑 Error al procesar la respuesta de Gemini: {e}")

return None

# 4. EJECUCIÓN Y VISUALIZACIÓN DE RESULTADOS

# --------------------------------------------------------------------------

# Llamamos a la función principal

reporte\_validacion = validar\_documento\_con\_ia(datos\_crm, texto\_oferta)

# Imprimimos los resultados de forma clara

if reporte\_validacion:

print("\n--- REPORTE DE VALIDACIÓN DE DOCUMENTO ---")

errores\_encontrados = False

for campo, data in reporte\_validacion.items():

estado = "✅ COINCIDE" if data.get("coincide") else "❌ NO COINCIDE"

if not data.get("coincide"):

errores\_encontrados = True

print(f"\n🔹 Campo: '{campo}'")

print(f" - Valor CRM: '{data.get('valor\_crm')}'")

print(f" - Valor en Documento: '{data.get('valor\_encontrado\_doc')}'")

print(f" - Resultado: {estado}")

print("\n--- RESUMEN FINAL ---")

if errores\_encontrados:

print("🔴 ¡ATENCIÓN! Se encontraron discrepancias. Se requiere revisión manual.")

else:

print("🟢 El documento ha sido validado exitosamente. Todos los campos coinciden.")

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