# Prueba Técnica – Data Scientist Backend Jr (revenew)

## Resumen de la Prueba

En esta prueba técnica desarrollarás una aplicación **Django** (solo backend, sin necesidad de frontend) para **revenew**. El objetivo es implementar un proyecto que permita **cargar un archivo CSV con datos**, **configurar parámetros de un modelo de optimización**, ejecutar dicho modelo desde un formulario web básico y mostrar los resultados obtenidos. El modelo de optimización (descrito en el documento proporcionado) se enfoca en la **maximización de ingresos diarios** de una empresa manufacturera, sujeto a **restricciones de capacidad** (tiempo disponible) en dos máquinas de producción.

## Objetivos técnicos

- Evaluar tus habilidades en desarrollo backend con Django.
- Manejo de datos (lectura/validación de archivos).
- Implementación de modelos matemáticos de optimización (usando técnicas o librerías de Python).
- Seguir buenas prácticas de código (diseño modular orientado a objetos, documentación, etc.).
- Se espera que tu solución demuestre claridad en la estructura, calidad en el código y buenas prácticas de desarrollo web (uso adecuado de views, templates de Django, validación de inputs, etc.).

## Instrucciones Técnicas

- **Estructura del proyecto sugerida:** Organiza el proyecto de forma **modular** y **orientada a objetos**. Se recomienda crear un proyecto Django con una aplicación (app) dedicada, por ejemplo llamada `optimizador`. Dentro de esta app, separa la lógica en módulos y clases según su responsabilidad. Por ejemplo, puedes tener módulos o paquetes para: lectura y carga de datos, ejecución del modelo de optimización, generación de resultados (y opcionalmente visualizaciones) y la capa web (views, templates). La idea es seguir buenas prácticas de diseño, manteniendo

una separación clara entre la lógica de negocio (la optimización) y la capa de presentación (Django views/templates).

- **Clases y componentes recomendados:** Implementa clases específicas para cada parte del flujo:
    - Una clase (ej. `DataLoader`) para la **carga del archivo CSV** y validación de datos. Debe leer el CSV (usando pandas u otra librería) y verificar que el formato y los valores sean correctos (por ejemplo, tipos de datos numéricos, columnas requeridas, etc.).
    - Una clase (ej. `OptimizationModel`) que encapsule el **modelo de optimización**. Esta clase debe tomar los datos de entrada (p. ej. parámetros desde el CSV y/o formulario) y ejecutar el algoritmo de optimización para obtener la solución. El modelo provisto es un problema de **maximización de ingresos** sujeto a restricciones de capacidad en dos máquinas, por lo que puedes implementar la solución usando métodos matemáticos o librerías (por ejemplo, resolver un problema lineal con `numpy`/`pandas` o utilizar librerías de optimización como PuLP, SciPy, ortools, etc.). Asegúrate de que la implementación calcule correctamente la combinación óptima de productos que maximiza el ingreso respetando las restricciones.
    - Una clase (ej. `ResultsHandler` o `OutputPresenter`) para **procesar y presentar los resultados**. Puede formatear los resultados del modelo (por ejemplo, cantidades óptimas a producir de cada producto y el ingreso total óptimo) para desplegarlos en la vista. Si se solicita o considera conveniente, esta clase también podría generar alguna **visualización básica** (por ejemplo, una gráfica de barras comparando ingresos o uso de capacidad, usando matplotlib) para enriquecer la presentación, aunque esto es opcional.
    - Adicionalmente, podrías incluir clases o funciones auxiliares para **validación** de parámetros (ej. verificar que la suma de tiempos requeridos no exceda la capacidad antes de correr el modelo, etc.) e incluso **formularios de Django** para manejar la entrada del usuario (por ejemplo, usando `forms.Form` para el archivo CSV y campos de parámetros adicionales).

- **Requerimientos mínimos (dependencias):** Tu proyecto debe usar **Python 3.10** y las siguientes librerías mínimas:
    - **Django** (framework web principal para crear las views, templates y routing necesarios).

    - **pandas** (para manipulación y validación de datos, lectura de CSV).
    - **numpy** (para cálculos numéricos que puedan ser necesarios en el modelo de optimización).
    - **matplotlib** (opcional, solo si decides incluir alguna visualización de resultados embebida en la web o para generar gráficos estáticos).
    - Cualquier otra librería estándar que requieras. *Nota:* Se permite y es válido usar librerías específicas de optimización (como `PuLP`, `scipy.optimize` o similares) si lo consideras adecuado para implementar el modelo matemático, aunque no es obligatorio. Incluye en el proyecto un archivo `requirements.txt` con Django, pandas, numpy, matplotlib y cualquier otra dependencia necesaria para ejecutar tu proyecto fácilmente.

- **Características opcionales (no obligatorias pero valoradas):** Además de los requerimientos mínimos, se valorará positivamente la incorporación de buenas prácticas adicionales:
    - Uso de **Docker:** Proveer un `Dockerfile` y/o `docker-compose.yml` que permita levantar fácilmente la aplicación (por ejemplo, una imagen con Python y las dependencias, configurando el proyecto Django). Esto facilitará la evaluación y despliegue de tu solución.
    - **Pruebas unitarias:** Incluye tests (por ejemplo, usando `unittest` o `pytest`) para partes clave de la lógica, especialmente la del modelo de optimización y las funciones de validación de datos. Tests para views de Django (usando el test client) también son bienvenidos.
    - **Visualización básica de resultados:** Aunque no se espera una interfaz gráfica compleja, cualquier esfuerzo por presentar claramente los resultados será apreciado. Esto podría ser en texto formateado en la página de resultados (por ej. "Producto A: 50 unidades, Producto B: 30 unidades, Ingreso total: $5000") y/o algún gráfico simple ilustrativo generado con matplotlib y mostrado como imagen estática en la página.
    - **Buenas prácticas de Django:** Por ejemplo, uso de **templates** para la página de carga de archivo y resultados (evitando generar HTML directamente en la view), separación de lógica pesada fuera de las views (manteniendo las views limpias, delegando cálculos a las clases mencionadas), manejo adecuado de errores (ej. si el CSV tiene formato inválido, mostrar un mensaje de error amigable en lugar de un traceback).

- **Documentación requerida:** La claridad en la documentación y explicaciones es fundamental. Asegúrate de:
    - Escribir un **README.md** claro (este mismo documento), que explique en español el objetivo del proyecto, cómo instalar las dependencias, cómo ejecutar la aplicación y cómo usarla (por ejemplo, cómo cargar el CSV y dónde ver los resultados). Debe incluir también cualquier consideración especial para el evaluador.
    - Incluir **docstrings** y comentarios en el código donde sea pertinente, para explicar partes no triviales: por ejemplo, documentar las clases y métodos que implementan la lógica de optimización.
    - Proporcionar **instrucciones de instalación y ejecución** detalladas. Por ejemplo: pasos para instalar dependencias (o construir la imagen Docker si aplicaste Docker), cómo correr migraciones si las hubiera (aunque puede que no uses un modelo de base de datos para esta prueba), cómo iniciar el servidor Django (`python manage.py runserver`) y cómo interactuar con la aplicación (qué URL abrir, qué pasos seguir para cargar el archivo y ejecutar el modelo).
    - Si incluiste pruebas unitarias, indica cómo ejecutarlas (por ejemplo `python manage.py test` o el comando de pytest).
    - Si incluiste Docker, explica cómo construir y levantar el contenedor para probar la app.

## Entrega

- **Repositorio:** Sube el código de tu solución a un repositorio **público** en GitHub (puede ser en tu cuenta personal). Comparte el enlace del repositorio para su revisión. La estructura del repo debe incluir todo el código fuente de la aplicación Django, el archivo README.md con la documentación, el `requirements.txt` con dependencias, y demás archivos necesarios (ej: Dockerfile, tests, etc. si los hay).
- **README e Instrucciones:** Asegúrate de que el repositorio cuente con el README.md explicado arriba, de modo que cualquier persona que llegue al repo (incluyendo los evaluadores de revenew) pueda seguir las instrucciones y ejecutar la aplicación sin inconvenientes. Incluye también ejemplos de cómo lucen los resultados o cómo se debe usar la aplicación si lo consideras útil.
- **Archivo principal (`main.py`):** Incluye un archivo `main.py` en la raíz del proyecto o en alguna ubicación evidente, que permita **ejecutar todo el flujo** de la optimización

de manera independiente. Por ejemplo, este archivo podría cargar un CSV de ejemplo, instanciar las clases de tu solución y ejecutar el modelo mostrando los resultados por consola. Esto sirve para poder probar la lógica principal sin necesidad de levantar el servidor web, y demostrar que la solución funciona de extremo a extremo. (Nota: Este `main.py` puede ser simplemente un pequeño script que use tus clases; en un contexto Django podrías configurarlo para ejecutar en el contexto del proyecto, o bien ubicarlo fuera de Django solo para la demostración de la lógica.)

- **Entrega final:** Una vez listo, envía el link del repositorio (y el Dockerhub link si publicaste una imagen, aunque no es necesario) al contacto designado. Asegúrate de que el repositorio esté público o que nos hayas dado acceso, y verifica que incluya todo lo necesario para ejecutar la solución.

## Criterios de Evaluación

Al evaluar tu prueba técnica, consideraremos los siguientes aspectos:

- **Implementación correcta del modelo de optimización:** El foco principal es comprobar que hayas implementado de forma correcta el modelo de maximización de ingresos con sus restricciones. Se validará con datos de prueba si al cargar diferentes CSV o parámetros, la aplicación encuentra la solución óptima respetando las restricciones de capacidad.
- **Calidad del código y estructura:** Se revisará la organización general del proyecto y del código. Buscamos un código limpio, bien estructurado, con separación apropiada de responsabilidades. El uso de clases y métodos debe facilitar la comprensión y mantenimiento. Aspectos como nomenclatura clara, modularidad, evitar repetición de código, etc., suman puntos.
- **Buenas prácticas en Django:** Evaluaremos si utilizaste adecuadamente las capacidades de Django. Por ejemplo, que las **views** sean simples y deleguen lógica a otras capas, que uses **Django templates** para la interfaz en lugar de HTML duro en las views, que manejes la subida de archivos de forma segura, y que apliques validaciones necesarias (por ejemplo, verificar el tipo de archivo subido, el tamaño, o validar los datos antes de procesarlos). No es necesario usar una base de datos para persistencia en este proyecto, pero si la utilizas para algo, se considerará también que esté bien justificado.
- **Documentación y claridad:** Damos mucha importancia a que la solución esté bien explicada. Un README completo, pasos de instalación claros, y comentarios/docstrings en el código serán parte de la nota. Queremos poder entender tu razonamiento y cómo usar tu aplicación sin tener que adivinar nada.

- **Extras (puntos adicionales):** Las características opcionales mencionadas (contenedor Docker funcional, suite de tests unitarios, inclusión de gráficos o elementos visuales, etc.) sumarán puntos extra si están presentes y bien implementadas. No son obligatorias, pero demuestran un esfuerzo adicional y conocimiento de buenas prácticas profesionales.
- **Presentación general:** Aunque el énfasis es el backend, se apreciará si la aplicación presenta de forma **amigable** la interacción de usuario (por ejemplo, mensajes claros en la interfaz web ante errores o al finalizar el procesamiento, instrucciones breves en la página de carga, etc.). No se evaluará diseño visual avanzado, pero sí la **experiencia de uso básica** y la corrección en la funcionalidad.

En resumen, la evaluación será integral: se tomará en cuenta no solo si "funciona", sino **cómo** está construida la solución. Un código limpio y bien documentado puede marcar la diferencia tanto como obtener el resultado correcto.

## Nosotros

**revenew** es una startup tecnológica enfocada en optimizar procesos de **Revenue Management y S&OP (Sales & Operations Planning)** para empresas del sector alimenticio. Nuestro equipo aplica **modelos matemáticos de inteligencia artificial, machine learning y optimización** para desarrollar soluciones de software que maximizan los ingresos y mejoran la eficiencia operativa de nuestros clientes. En la actualidad, revenew colabora con empresas líderes en la industria de alimentos, ayudándoles a optimizar la **asignación de precios** de todos sus productos en diversos canales y mercados, logrando resultados tangibles como incrementos de margen de hasta un 10% y mejoras en indicadores operativos (por ejemplo, aumento del fill rate y optimización de inventarios). Nuestra propuesta de valor se basa en un enfoque integral: combinamos análisis de grandes volúmenes de datos con modelos avanzados para determinar precios y cantidades óptimas, reduciendo la carga de trabajo manual de los equipos de pricing y brindando una solución escalable, segura y de alto impacto financiero. En resumen, revenew ofrece un software robusto y especializado, construido 100% con foco en empresas de alimentos, que actúa como un **optimizador global** de precios y mix de ventas para maximizar el beneficio de nuestros clientes.