

Entrega 3 Proyecto Desarrollo de soluciones

Predicción de la calidad del aire en Risaralda mediante modelos de aprendizaje automático

Equipo:

- Brayan Sthefen Gomez Salamanca
- Juan Sebastian Ordoñez Acuña
- María Alejandra Rojas Garzón
- Hainer Jair Torrenegra Jimenez

1. Resumen del problema

La calidad del aire es un factor determinante para la salud pública y el bienestar de la población. En el departamento de Risaralda, los niveles de material particulado (PM10 y PM2.5) han sido motivo de preocupación debido a su impacto en enfermedades respiratorias y cardiovasculares, especialmente en zonas urbanas con alta actividad vehicular e industrial.

El problema identificado consiste en la falta de herramientas que permitan analizar y predecir de forma oportuna los niveles de contaminación del aire, de manera que las autoridades y la ciudadanía puedan tomar decisiones informadas para mitigar riesgos.

La pregunta de negocio que orienta este trabajo es:

¿Es posible construir un modelo de aprendizaje automático que, a partir de datos históricos de calidad del aire, permita estimar y monitorear los niveles de PM10 y PM2.5 en los municipios de Risaralda, facilitando la generación de alertas y la planificación de medidas preventivas?

El alcance del proyecto incluye:

- Análisis exploratorio y preprocesamiento de los datos para identificar patrones espaciales y temporales, además de corregir posibles errores.
- Entrenamiento y evaluación de múltiples modelos de aprendizaje automático, con registro en MLflow.
- Implementación de una API que exponga el modelo seleccionado.
- Desarrollo de un dashboard interactivo que muestre predicciones y facilite la interpretación de resultados.
- Aplicar los conocimientos adquiridos sobre CI/CD para el despliegue y operación del modelo con plataformas como circleCI y railway

El proyecto se apoya en datos abiertos oficiales del portal datos.gov.co, que contienen 5,047 registros recolectados en estaciones municipales de Risaralda entre 2007 y 2023, con las variables: Municipio, Estación, Fecha, Diámetro aerodinámico (PM10 o PM2.5) y Medición ($\mu\text{g}/\text{m}^3$), algunos hallazgos relevantes vs la entrega anterior son la presencia de ceros en Medición, asimetría y outliers y la heterogeneidad temporal por estación/municipio.

2. Modelos desarrollados y su evaluación

Selección de características

Se trabajó con el dataset Calidad del Aire Enriquecido obtenido en el EDA (exploración de datos), que incluye variables de contexto temporal y de localización:

- **Fecha** (variable temporal de referencia).
- **Municipio, Estación, Diámetro aerodinámico, DíaSemana** (categóricas).
- **Día, Mes, Año** (numéricas).

Además, se generaron variables derivadas para capturar la naturaleza secuencial de la serie temporal:

- **Lags del objetivo** (lag_1, lag_7, lag_30) por grupo (Municipio, Estación, Diámetro aerodinámico), de forma que cada registro solo use información disponible en el pasado.
- **Promedios móviles (rolling means)** con ventanas de 7 y 30 observaciones, también calculados por grupo y desplazados un paso para evitar fuga de información.
- **Variables de calendario** derivadas de la fecha: día de la semana (dow), mes (month) y año (year).

El preprocesamiento se implementó mediante un pipeline de scikit-learn con las siguientes transformaciones:

- **Numéricas:** imputación por mediana y estandarización (StandardScaler).
- **Categóricas:** imputación por moda y codificación one-hot (OneHotEncoder).

Este esquema asegura consistencia entre entrenamiento, validación y despliegue.

Estrategia de validación

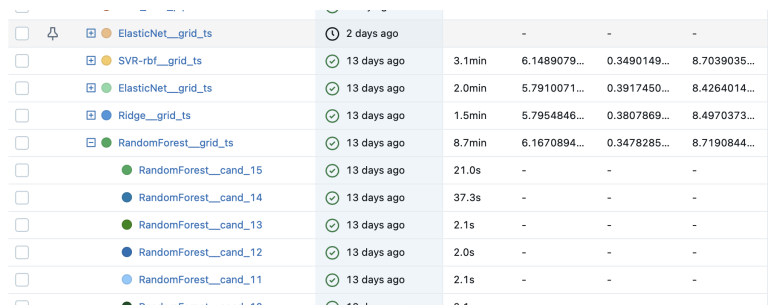
Se planteó un proceso de validación temporal (expanding window) mediante TimeSeriesSplit para respetar la naturaleza cronológica de los datos: los primeros años para entrenar al modelo y las ventanas recientes para validación; reservando el último año como **holdout final** para medir desempeño fuera de muestra.

Búsqueda de modelos e hiperparámetros

En la entrega anterior, se entrenaron y compararon los siguientes algoritmos:

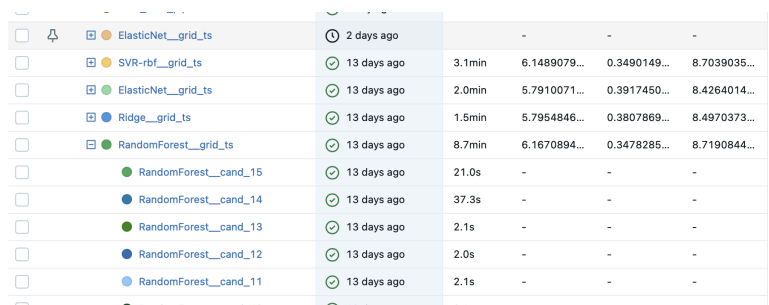
- **Modelos lineales:**
 - *Ridge Regression*: regresión lineal con regularización L2.
 - *ElasticNet*: combinación de regularización L1 y L2.
- **Modelos no lineales:**
 - *Random Forest Regressor*: ensamblado de árboles con bagging.
 - *Support Vector Regressor (SVR, kernel RBF)*: modelo de soporte con kernel gaussiano.

Para escoger el mejor modelo se usó grid search, una técnica de optimización de hiperparámetros que prueba exhaustivamente todas las combinaciones de valores para encontrar la configuración que produce el mejor rendimiento del modelo, como se observa en MLflow:



<input type="checkbox"/>		ElasticNet__grid_ts	🕒 2 days ago	-	-	-
<input type="checkbox"/>		SVR-rbf__grid_ts	🕒 13 days ago	3.1min	6.1489079...	0.3490149... 8.7039035...
<input type="checkbox"/>		ElasticNet__grid_ts	🕒 13 days ago	2.0min	5.7910071...	0.3917450... 8.4264014...
<input type="checkbox"/>		Ridge__grid_ts	🕒 13 days ago	1.5min	5.7954846...	0.3807869... 8.4970373...
<input type="checkbox"/>		RandomForest__grid_ts	🕒 13 days ago	8.7min	6.1670894...	0.3478285... 8.7190844...
<input type="checkbox"/>		RandomForest__cand_15	🕒 13 days ago	21.0s	-	-
<input type="checkbox"/>		RandomForest__cand_14	🕒 13 days ago	37.3s	-	-
<input type="checkbox"/>		RandomForest__cand_13	🕒 13 days ago	2.1s	-	-
<input type="checkbox"/>		RandomForest__cand_12	🕒 13 days ago	2.0s	-	-
<input type="checkbox"/>		RandomForest__cand_11	🕒 13 days ago	2.1s	-	-

Cada experimento individual se guardó como un candidato dentro de cada modelo, con esa base, se ejecutó una segunda ronda con los dos mejores candidatos hasta seleccionar el modelo con mejor desempeño.



<input type="checkbox"/>		ElasticNet__grid_ts	🕒 2 days ago	-	-	-
<input type="checkbox"/>		SVR-rbf__grid_ts	🕒 13 days ago	3.1min	6.1489079...	0.3490149... 8.7039035...
<input type="checkbox"/>		ElasticNet__grid_ts	🕒 13 days ago	2.0min	5.7910071...	0.3917450... 8.4264014...
<input type="checkbox"/>		Ridge__grid_ts	🕒 13 days ago	1.5min	5.7954846...	0.3807869... 8.4970373...
<input type="checkbox"/>		RandomForest__grid_ts	🕒 13 days ago	8.7min	6.1670894...	0.3478285... 8.7190844...
<input type="checkbox"/>		RandomForest__cand_15	🕒 13 days ago	21.0s	-	-
<input type="checkbox"/>		RandomForest__cand_14	🕒 13 days ago	37.3s	-	-
<input type="checkbox"/>		RandomForest__cand_13	🕒 13 days ago	2.1s	-	-
<input type="checkbox"/>		RandomForest__cand_12	🕒 13 days ago	2.0s	-	-
<input type="checkbox"/>		RandomForest__cand_11	🕒 13 days ago	2.1s	-	-

Posteriormente, se realizó una nueva búsqueda de hiperparámetros sobre el modelo escogido que para este proyecto fue XGBRegressor.

Run Name	Created	Metrics						
		Duration	best_cv_mae	best_cv_r2	best_cv_rmse	holdout_mae	holdout_r2	holdout_rmse
XGBRegressor_grid_ts	1 day ago	2.6h	5.9783476...	0.3743971...	8.5490412...	4.1888589...	0.5829475...	5.8832217...
XGBRegressor__cand_575	1 day ago	1.8s	-	-	-	-	-	-
XGBRegressor__cand_574	1 day ago	1.7s	-	-	-	-	-	-
XGBRegressor__cand_573	1 day ago	1.8s	-	-	-	-	-	-
XGBRegressor__cand_572	1 day ago	1.1min	-	-	-	-	-	-
XGBRegressor__cand_571	1 day ago	20.7s	-	-	-	-	-	-
XGBRegressor__cand_570	1 day ago	1.7s	-	-	-	-	-	-
XGBRegressor__cand_569	1 day ago	1.7s	-	-	-	-	-	-
XGBRegressor__cand_568	1 day ago	1.7s	-	-	-	-	-	-
XGBRegressor__cand_567	1 day ago	1.7s	-	-	-	-	-	-
XGBRegressor__cand_566	1 day ago	1.7s	-	-	-	-	-	-

Registro de experimentos en MLflow

Se configuró un **tracking server de MLflow** para almacenar:

- Hiperparámetros explorados en cada grid search.
- Métricas de validación cruzada (cv_rmse, cv_mae, cv_r2).
- Métricas de desempeño en el holdout (holdout_rmse, holdout_mae, holdout_r2).
- Artefactos asociados (modelo serializado, predicciones del holdout).

Cada combinación de hiperparámetros se registró como un *child run*, anidado dentro del *run principal* de cada modelo (ejemplo: ElasticNet__cand_5).

<input type="checkbox"/>	marvelous-carp-983	36 minutes ago	16.0s	C:\User...	-	-	-	-	-	-
<input checked="" type="checkbox"/>	SVR-rbf_grid_ts	41 minutes ago	3.1min	compar...	6.1489079...	-	-	-	8.7039035...	0.3490149...
<input type="checkbox"/>	indecisive-hog-520	41 minutes ago	11.9s	C:\User...	-	-	-	-	-	-
<input type="checkbox"/>	glamorous-doe-149	42 minutes ago	8.7s	C:\User...	-	-	-	-	-	-
<input checked="" type="checkbox"/>	ElasticNet_grid_ts	43 minutes ago	2.0min	compar...	5.7910071...	-	-	-	8.4264014...	0.3917450...
<input checked="" type="checkbox"/>	Ridge_grid_ts	44 minutes ago	1.5min	compar...	5.7954846...	-	-	-	8.4970373...	0.3807869...
<input type="checkbox"/>	capricious-shoat-3...	45 minutes ago	8.5s	C:\User...	-	-	-	-	-	-
<input type="checkbox"/>	treasured-grub-212	48 minutes ago	14.3s	C:\User...	-	-	-	-	-	-
<input type="checkbox"/>	bright-eel-32	50 minutes ago	16.6s	colab_k...	-	-	-	-	-	-
<input type="checkbox"/>	rumbling-bee-659	50 minutes ago	20.4s	colab_k...	-	-	-	-	-	-
<input checked="" type="checkbox"/>	RandomForest_gri...	53 minutes ago	8.7min	compar...	6.1670894...	-	-	-	8.7190844...	0.3478285...

Modelo	best_cv_rmse	best_cv_mae	best_cv_r ²	holdout_rmse	holdout_mae	holdout_r ²
ElasticNet	8.4264	5.7910	0.3917	5.9673	4.1953	0.5709
Ridge	8.4970	5.7955	0.3809	5.9850	4.2218	0.5684
RandomForest	8.7191	6.1671	0.3478	6.0551	4.4264	0.5582
SVR (RBF)	8.7039	6.1489	0.3490	6.1347	4.3444	0.5465
XGBRegressor	6.9360	4.5050	0.5877	5.5972	3.9224	0.6205

Lectura de los resultados

A partir del tracking en MLflow, la comparación se centró en ElasticNet y XGBRegressor:

- **ElasticNet:** en la entrega anterior ya había mostrado un desempeño competitivo, con holdout aproximado de $RMSE \approx 5.97$, $MAE \approx 4.20$ y $R^2 \approx 0.57$, sirviendo como referencia robusta para los modelos subsecuentes.
- **XGBRegressor (modelo campeón):** en la nueva ronda de experimentación evidenció mejoras consistentes tanto en CV como en holdout frente a ElasticNet, con menor error (RMSE/MAE) y mayor capacidad explicativa (R^2). Estas mejoras se atribuyen a su habilidad para capturar no linealidades e interacciones entre variables temporales y categóricas.

Selección del mejor modelo:

El modelo seleccionado fue **XGBRegressor**, con los siguientes hiperparámetros:

- `reg_alpha = 0.1`
- `reg_lambda = 5.0`
- `learning_rate = 0.01`
- `min_child_weight = 5`
- `n_estimators = 600`
- `max_depth = 3`
- `subsample = 0.9`
- `colsample_bytree = 1.0`
- `random_state = 13`
- `tree_method = "hist"`
- `n_jobs = -1, verbosity = 0`

Esta configuración favorece un árbol poco profundo y regularizado (profundidad moderada, `reg_alpha` y `reg_lambda` > 0), con tasa de aprendizaje baja y muchos estimadores, lo que mejora la capacidad del ensamble para capturar no linealidades e interacciones manteniendo el control del sobreajuste por `subsample` y el `min_child_weight`.

El desempeño fue:

- **Validación cruzada (mejor fold):** $RMSE \approx 6.9360$, $MAE \approx 4.5050$, $R^2 \approx 0.5877$.
- **Conjunto de prueba (holdout):** $RMSE \approx 5.5972$, $MAE \approx 3.9224$, $R^2 \approx 0.6205$.

En consecuencia, XGBRegressor con esta configuración se considera el modelo final para despliegue en la API, reemplazando a ElasticNet como referencia previa.

3. Observaciones y conclusiones sobre los modelos

Principales hallazgos

- En la primera ronda, los cuatro modelos evaluados (ElasticNet, Ridge, RandomForest y SVR) mostraron un desempeño similar en validación cruzada ($RMSE \approx 8.4\text{--}8.7$), con ElasticNet como baseline competitivo en holdout ($RMSE \approx 5.97$, $MAE \approx 4.20$, $R^2 \approx 0.57$).
- Tras la segunda ronda enfocada en los finalistas y una búsqueda exhaustiva de hiperparámetros, XGBRegressor superó a ElasticNet tanto en CV como en holdout.
- La mejora puede explicarse por la capacidad de XGBRegressor para capturar no linealidades e interacciones entre variables temporales y categóricas, manteniendo la estabilidad y tiempos de entrenamiento razonables.

Limitaciones

- **Disponibilidad y calidad de datos:**
 - Se identificaron **saltos temporales en los datos**, es decir, periodos donde no se registraron mediciones. Esto genera interrupciones en la secuencia temporal, lo que dificulta capturar tendencias continuas y reduce la calidad de los lags y promedios móviles generados.
 - No se incorporaron variables externas (ej. meteorología, tráfico, actividad industrial) que podrían mejorar el poder predictivo.
- **Overfitting:**
 - La cercanía entre métricas de CV y holdout sugiere generalización estable; no obstante, existe margen de mejora con mayor cobertura temporal y variables exógenas.
- **Tiempos de entrenamiento:**
 - Los modelos lineales (Ridge, ElasticNet) se entrenaron rápidamente.
 - RandomForest y SVR tuvieron tiempos de entrenamiento mayores sin lograr mejores métricas.
 - XGBRegressor implica coste computacional moderado, pero justificado por el mejor rendimiento y su viabilidad para despliegue con la configuración actual

Conclusión

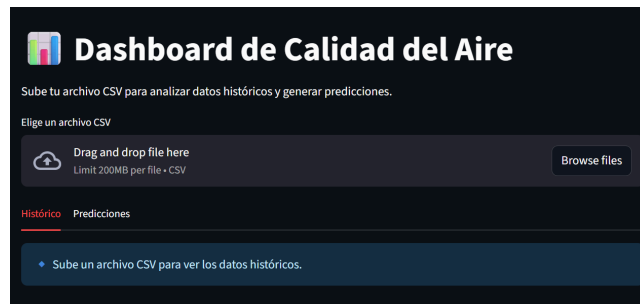
- El modelo **XGBRegressor** se consolida como la mejor alternativa para la predicción de mediciones de calidad del aire en este proyecto, ya que ofrece el mejor equilibrio entre **precisión, y viabilidad operativa**.
- En relación con la **pregunta de negocio** —“¿Es posible construir un modelo de aprendizaje automático que, a partir de datos históricos de calidad del aire, permita estimar y monitorear los niveles de PM10 y PM2.5 en los municipios de Risaralda, facilitando la generación de alertas y la planificación de medidas preventivas?”—, la respuesta es positiva: se logró construir un modelo con un **R^2 cercano al 0.62**, lo

cual significa que explica aproximadamente un **62% de la variabilidad** en los datos recientes.

- Esto abre la puerta a **integrar nuevas variables externas** y **afinar estrategias de imputación** y **ventanas temporales**.

4. Tablero desarrollado

Descripción: El tablero es una aplicación web desarrollada en Streamlit diseñada para analizar series históricas de calidad del aire y generar predicciones de manera interactiva, facilitando la interpretación de resultados para usuarios técnicos y no técnicos.



Estructura y flujo de uso: La interfaz se organiza en dos pestañas: Histórico y Predicciones. El usuario inicia cargando un archivo CSV con datos de medición; la aplicación normaliza nombres de columnas, convierte tipos y limpia registros no numéricos. A partir de esta carga, se habilitan controles de filtro por Municipio, Estación y rango de fechas.

Funcionalidad:

- **Exploración histórica:** En la pestaña **Histórico**, el usuario puede:
 - Filtrar dinámicamente por municipio, estación y período.
 - Consultar tres métricas de resumen (promedio, máximo y mínimo de PM10) para el subconjunto filtrado.
 - Visualizar una serie temporal interactiva (Plotly) de PM10 por estación, útil para detectar picos, tendencias y estacionalidades.
 - Explorar la tabla filtrada con los datos subyacentes.



- **Predicción y apoyo a decisiones:** En la pestaña **Predicciones**, el usuario selecciona municipios, estaciones y un rango de fechas objetivo, con un clic en “Realizar Predicción”, el tablero:
 - Construye automáticamente las características de calendario (Año, Mes, Día, DíaSemana) por fecha/estación/municipio.
 - Envía la solicitud al servicio de inferencia vía API REST.
 - Recibe y muestra las predicciones de PM10 en una tabla y en un gráfico de línea por estación, además de métricas de resumen (promedio, máximo y mínimo de la predicción) para una lectura rápida del riesgo.



5. **Empaquetamiento y despliegue:** La solución se compone de API y Dashboard, ambos empaquetados en Docker y desplegados en Railway mediante CI/CD con CircleCI. Cada cambio en main dispara un pipeline que construye imágenes, ejecuta validaciones y publica artefactos antes del despliegue.

- **API:** Dockerfile raíz con Python 3.13-slim, instalación vía requirements.txt, copia del código y arranque con Uvicorn (puerto 8001).

```

# Dockerfile
FROM python:3.13-slim
ENV PYTHONUNBUFFERED=1 PYTHONPATH=/app PIP_NO_CACHE_DIR=1
RUN apt-get update && apt-get install -y --no-install-recommends build-essential && rm -rf /var/lib/apt/lists/*
WORKDIR /app

# 1) Deps del MODELO
COPY model_requirements.txt /app/model_requirements.txt
RUN python -m pip install --upgrade pip && pip install -r /app/model_requirements.txt

# 2) Copia el wheel ANTES de instalar api_requirements
COPY dist/ /app/dist/
# (Opcional) falla temprano si no llegó el wheel
RUN test -f /app/dist/calidad_aire_model-0.2.0-py3-none-any.whl || (echo "Falta el wheel en /app/dist" && ls -ls /app/dist && exit 1)

# 3) Instala deps de la API
COPY api_requirements.txt /app/api_requirements.txt
RUN pip install -r /app/api_requirements.txt

# 4) Resto del código y modelo local (si aplica)
COPY . /app
ENV MODEL_URI="file:/app/dist"
EXPOSE 8001
CMD ["uvicorn", "app.api.main:app", "--host", "0.0.0.0", "--port", "8001"]

```

- **Dashboard:** app/Dockerfile con Python 3.13-slim, dependencias de Streamlit, copia del código y CMD para streamlit run, expuesto en 8501 (0.0.0.0).


```

Dockerfile x
app > Dockerfile
1 # syntax=docker/dockerfile:1
2 FROM python:3.13-slim
3
4 ENV PYTHONUNBUFFERED=1
5 WORKDIR /srv/app
6
7 # Instala dependencias del dashboard (archivo dentro de /app)
8 COPY requirements_dashboard.txt /tmp/requirements.txt
9 RUN pip install --upgrade pip -y \
10 && pip install --no-cache-dir -r /tmp/requirements.txt
11
12 # Copiamos solo lo necesario para el dashboard
13
14 COPY . /srv/app/
15
16 # Streamlit respetará PORT que inyecte Railway / el run local
17 ENV PORT=8501
18 EXPOSE 8501
19
20 # La URL de la API la pasaremos por env en runtime (API_URL)
21 CMD ["sh", "-c", "streamlit run app_streamlit.py --server.address 0.0.0.0 --server.port ${PORT}"]

```

- **CircleCI:** Por medio de CircleCI se realizará la automatización de construcción y despliegue de nuestros servicios. En el archivo de configuración `/.circleci/config.yml` establecemos que los cambios generados en el repositorio en la rama `main` van a disparar el despliegue de los jobs `deploy_app_to_railway` y `deploy_dashboard_to_railway`, estos jobs construirán las imágenes de docker de los componentes previamente revisados (API y Dashboard) y serán desplegados en railway en los servicios `acceptable-amazement` y `humble-freedom` respectivamente.

Pipeline	Status	Workflow	Checkout source	Trigger event	Start	Duration	Actions
Microprojecto-DS 21	Success	deploy_pipeline	main b0a5788 Merge pull request #111 from SebastianOrd/DashboardV2	Push Commit pushed	14m ago	45s	...
Microprojecto-DS 20	CREATED	No workflow	DashboarV2 a0936ad Dashboard-v2 - Se incluye promedio en prediccion y ya no necesita csv inicialmente.	Push Commit pushed	17m ago	---	...
Microprojecto-DS 19	Success	deploy_pipeline	main 47e33eb limpieza de repositorio	Push Commit pushed	18m ago	52s	...

Dentro de la plataforma se puede observar el histórico de los cambios recibidos en el repositorio y que disparan la ejecución de los pipelines junto con el estado final de la ejecución del mismo.

- **Despliegue API y Dashboard:**

```

> _ acceptable-amazement / dace58fd (Active) ... Sep 20, 2025 at 6:15 PM x
acceptable-amazement-production.up.railway.app

Details Build Logs Deploy Logs HTTP Logs

Search build logs

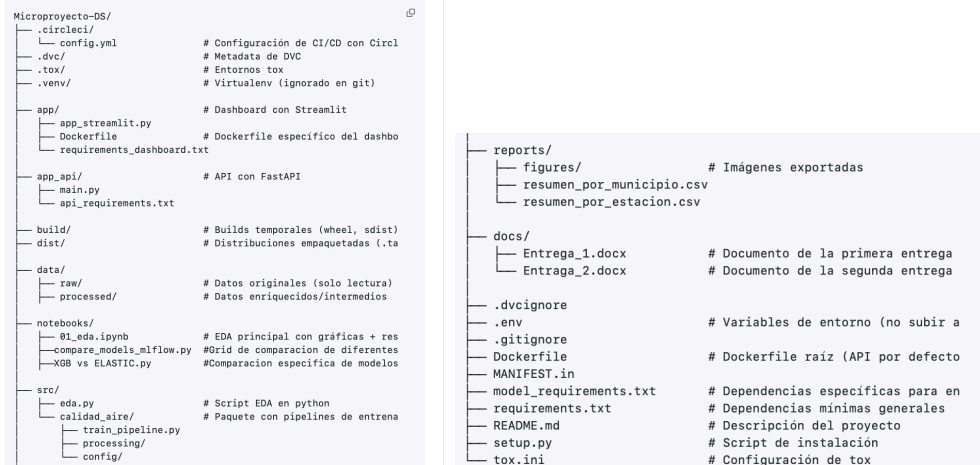
Time (GMT-5) Message
Sep 20 2025 at 18:15:32 [Region: us-east-1]
Sep 20 2025 at 18:15:32 =====
Sep 20 2025 at 18:15:32 Using Detected Dockerfile
Sep 20 2025 at 18:15:32 =====
Sep 20 2025 at 18:15:32 context: c2sk-w8XF
Sep 20 2025 at 18:15:32 ✓ internal load build definition from Dockerfile 8ms
Sep 20 2025 at 18:15:32 ✓ internal load metadata for docker.io/library/python:3.13-slim 62ms
Sep 20 2025 at 18:15:32 ✓ internal load .dockerignore 8ms
Sep 20 2025 at 18:15:32 [ 1/10] FROM docker.io/library/python:3.13-slim@sha256:58c30f5bfaa710b880a3e339319809c68bd517c44cdc94c1b6c8c172b6fad040 7ms
Sep 20 2025 at 18:15:32 ✓ internal load build context 8ms
Sep 20 2025 at 18:15:32 [ 9/10] RUN pip install -r /app/api_requirements.txt cached 3ms
Sep 20 2025 at 18:15:32 [ 8/10] COPY api_requirements.txt /app/api_requirements.txt cached 3ms

```

6. Reporte de trabajo en equipo

- **Evidencia repositorio:**

En el repositorio quedó con la siguiente estructura para el proyecto



Los trabajos fueron repartidos entre todos los miembros del equipo como se expone más adelante, así que cada uno creó una rama donde fue subiendo los cambios realizados para al final hacer un merge de todo, esto se puede observar directamente en el repositorio: <https://github.com/SebastianOrd/Microproyecto-DS>

- **Alejandra – Desarrollo de modelo (MLFlow) y documento final:** Pruebas con múltiples modelos, redacción del documento final y consolidación del reporte de trabajo en equipo.
- **Hainer – Desarrollo de la API (Backend):** Implementación de la API en FastAPI/Flask, creación de endpoints, despliegue.
- **Bryan – Desarrollo del Dashboard (Frontend):** Construcción del tablero interactivo, visualizaciones y documentación de uso.
- **Sebastián – Desarrollo de modelo (MLFlow), pruebas, análisis de resultados:** Validación integral del repositorio, gestión de las actividades, análisis de resultados, pruebas con múltiples modelos.

Anexos:

1. Video: <https://www.youtube.com/watch?v=3vNvOOyCXQM>
2. Manual del usuario: MANUAL_DASHBOARD.pdf
3. manual de instalacion: MANUAL_INSTALACION.pdf