



Estudiante: David Guambaña

Informe Técnico: Pipeline de Datos COVID-19

Resumen

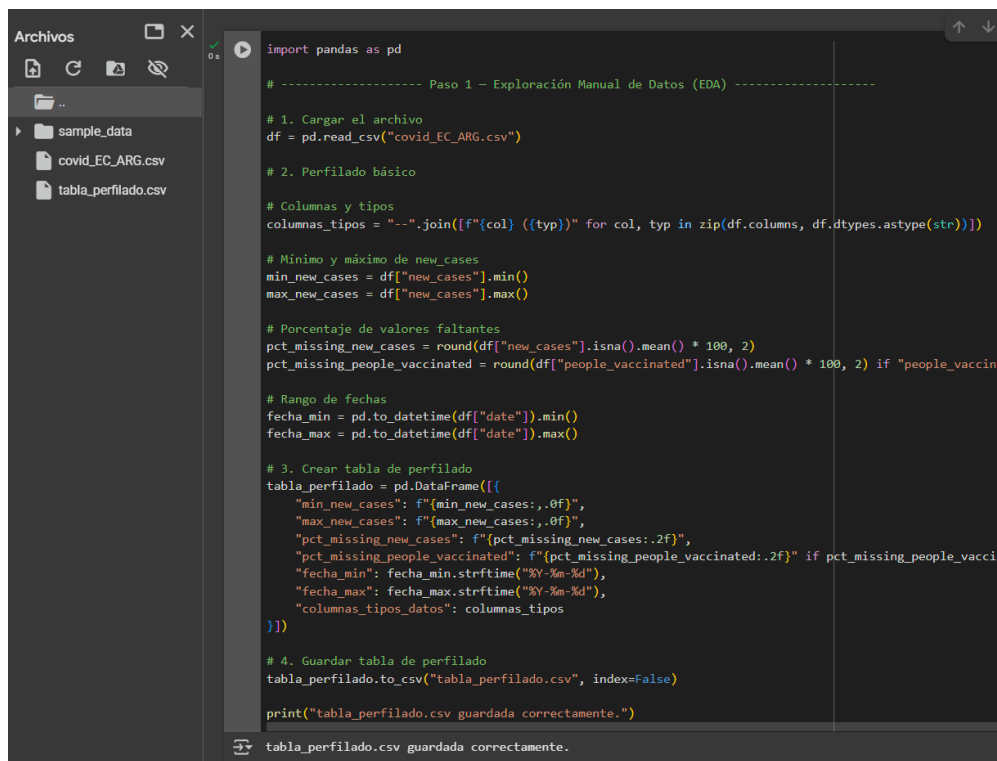
Este informe describe el desarrollo de un pipeline de análisis de datos de COVID-19 utilizando Python y Dagster. Se implementaron cinco pasos principales: lectura de datos, chequeos de calidad, procesamiento, cálculo de métricas y exportación de resultados. Cada paso incluyó validaciones y transformaciones orientadas a la generación de métricas confiables y reportes en Excel.

1. Introducción

El objetivo de este proyecto fue desarrollar un pipeline reproducible para analizar datos de COVID-19 de Ecuador y Argentina. Se buscó garantizar la calidad de los datos, calcular métricas clave como incidencia acumulada a 7 días y factor de crecimiento semanal, y generar reportes listos para análisis.

2. Paso 1: Exploración y Perfilado de Datos

- Se cargó un CSV con datos de casos diarios y vacunación.
- Se realizó un perfilado básico para identificar columnas, tipos de datos, valores mínimos y máximos, y porcentaje de valores faltantes.
- Se definió el rango de fechas de los datos.



```
import pandas as pd

# ----- Paso 1 - Exploración Manual de Datos (EDA) -----

# 1. Cargar el archivo
df = pd.read_csv("covid_EC_ARG.csv")

# 2. Perfilado básico

# Columnas y tipos
columnas_tipos = "---".join([f"{col} ({typ})" for col, typ in zip(df.columns, df.dtypes.astype(str))])

# Mínimo y máximo de new_cases
min_new_cases = df["new_cases"].min()
max_new_cases = df["new_cases"].max()

# Porcentaje de valores faltantes
pct_missing_new_cases = round(df["new_cases"].isna().mean() * 100, 2)
pct_missing_people_vaccinated = round(df["people_vaccinated"].isna().mean() * 100, 2) if "people_vaccinated" in df.columns else None

# Rango de fechas
fecha_min = pd.to_datetime(df["date"]).min()
fecha_max = pd.to_datetime(df["date"]).max()

# 3. Crear tabla de perfilado
tabla_perfilado = pd.DataFrame([{"min_new_cases": f"{min_new_cases:.0f}",
                                "max_new_cases": f"{max_new_cases:.0f}",
                                "pct_missing_new_cases": f"{pct_missing_new_cases:.2f}",
                                "pct_missing_people_vaccinated": f"{pct_missing_people_vaccinated:.2f}" if pct_missing_people_vaccinated is not None else "None",
                                "fecha_min": fecha_min.strftime("%Y-%m-%d"),
                                "fecha_max": fecha_max.strftime("%Y-%m-%d"),
                                "columnas_tipos_datos": columnas_tipos
                                }])

# 4. Guardar tabla de perfilado
tabla_perfilado.to_csv("tabla_perfilado.csv", index=False)

print("tabla_perfilado.csv guardada correctamente.")
```



min_new_cases	max_new_cases	pct_missing_new_cases	pct_missing_people_vaccinated	fecha_min	fecha_max	columnas_tipos_datos
0	135,401	0.85	58.62	2020-01-01	2025-12-31	country (object)-date (object)-total_cases (float64)-new_cases (float64)-new_cases_smoothed (float64)-total_cases_per_million (float64)-new_cases_per_million (float64)-new_cases_smoothed_per_million (float64)-total_deaths (float64)-new_deaths (float64)-new_deaths_smoothed (float64)-total_deaths_per_million (float64)-new_deaths_per_million (float64)-new_deaths_smoothed_per_million (float64)-excess_mortality (float64)-excess_mortality_cumulative (float64)-excess_mortality_cumulative_absolute (float64)-excess_mortality_cumulative_per_million (float64)-hosp_patients (float64)-hosp_patients_per_million (float64)-weekly_hosp_admissions (float64)-weekly_hosp_admissions_per_million (float64)-icu_patients (float64)-icu_patients_per_million (float64)-weekly_icu_admissions (float64)-weekly_icu_admissions_per_million (float64)-stringency_index (float64)-reproduction_rate (float64)-total_tests (float64)-new_tests (float64)-total_tests_per_thousand (float64)-new_tests_per_thousand (float64)-new_tests_smoothed (float64)-new_tests_smoothed_per_thousand (float64)-positive_rate (float64)-tests_per_case (float64)-total_vaccinations (float64)-people_vaccinated (float64)-people_fully_vaccinated (float64)-total_boosters (float64)-new_vaccinations (float64)-new_vaccinations_smoothed (float64)-total_vaccinations_per_hundred (float64)-people_vaccinated_per_hundred (float64)-people_fully_vaccinated_per_hundred (float64)-total_boosters_per_hundred (float64)-new_vaccinations_smoothed_per_million (float64)-new_people_vaccinated_smoothed (float64)-new_people_vaccinated_smoothed_per_hundred (float64)-code (object)-continent (object)-population (float64)-population_density (float64)-median_age (float64)-life_expectancy (float64)-gdp_per_capita (float64)-extreme_poverty (float64)-diabetes_prevalence (float64)-handwashing_facilities (float64)-hospital_beds_per_thousand (float64)-human_development_index (float64)

3. Paso 2: Lectura de Datos y Chequeos Iniciales

- Se descargaron los datos desde la fuente oficial de Our World in Data.
- Se verificó que las fechas fueran válidas (sin futuras) y que las columnas clave no contuvieran nulos.
- Se comprobó la unicidad de las filas por país y fecha.

```
# ----- Paso 2: Lectura de datos y chequeos -----
@op
def leer_datos() -> pd.DataFrame:
    # Descargamos el CSV oficial de Our World in Data
    # y lo leemos directamente en un DataFrame sin modificarlo
    url = "https://catalog.ourworldindata.org/garden/covid/latest/compact/compact.csv"
    resp = requests.get(url, timeout=10)
    if resp.status_code != 200:
        # Si hay un error en la descarga, devolvemos un DataFrame vacío
        print(f"No se pudo descargar el CSV, status_code={resp.status_code}")
        return pd.DataFrame()
    # Convertimos el contenido descargado a DataFrame
    df = pd.read_csv(StringIO(resp.text))
    print(f"CSV descargado correctamente, filas={len(df)}")
    return df

@op
def chequear_datos(df: pd.DataFrame) -> pd.DataFrame:
    # Convertimos la columna de fechas a datetime
    df["date"] = pd.to_datetime(df["date"], errors="coerce")
    hoy = pd.Timestamp(datetime.today().date())

    # Verificamos que no haya fechas futuras
    fechas_invalidas = df[df["date"] > hoy]
    if len(fechas_invalidas) > 0:
        print(f"Alerta: {len(fechas_invalidas)} filas tienen fecha mayor a hoy")
    else:
        print("Todas las fechas son válidas")

    # Revisamos que las columnas importantes no tengan valores nulos
    columnas_clave = ["country", "date", "population"]
    nulos = df[columnas_clave].isna().sum().sum()
    if nulos > 0:
        print(f"Alerta: se encontraron {nulos} valores nulos en {columnas_clave}")
    else:
        print("Todas las columnas clave contienen datos")

    # Comprobamos que no existan filas duplicadas por país y fecha
    duplicados = df.duplicated(subset=["country", "date"]).sum()
    if duplicados > 0:
        print(f"Alerta: se encontraron {duplicados} filas duplicadas por (country, date)")
    else:
        print("No hay duplicados por país y fecha")

    return df
```

4. Paso 3: Procesamiento de Datos

- Se eliminaron filas con valores nulos en columnas relevantes y duplicados.
- Se filtraron únicamente los países de interés: Ecuador y Argentina.
- Se seleccionaron columnas esenciales para análisis posteriores.

```
# ----- Paso 3: Procesamiento de Datos -----
@op
def datos_procesados(df: pd.DataFrame) -> pd.DataFrame:
    # Eliminamos filas con valores nulos en columnas importantes
    # y duplicados, luego filtramos solo los países que nos interesan
    df_proc = df.dropna(subset=["new_cases", "people_vaccinated"]).drop_duplicates()
    df_proc = df_proc[df_proc["country"].isin(["Ecuador", "Argentina"])]
    columnas = ["country", "date", "new_cases", "people_vaccinated", "population"]
    print(f"Datos procesados, filas después de limpieza={len(df_proc)}")
    return df_proc[columnas]
```

country	date	new_cases	people_vaccinated	population
Argentina	2020-12-29 00:00:00	12671	20492	45407866
Argentina	2020-12-30 00:00:00	12545	40589	45407866
Argentina	2020-12-31 00:00:00	6969	43395	45407866
Argentina	2021-01-01 00:00:00	3422	43524	45407866
Argentina	2021-01-02 00:00:00	7161	46833	45407866
Argentina	2021-01-03 00:00:00	5470	47274	45407866
Argentina	2021-01-04 00:00:00	14114	57731	45407866

5. Paso 4: Cálculo de Métricas

- Incidencia acumulada a 7 días por cada 100,000 habitantes.
- Factor de crecimiento semanal basado en la comparación de casos de semanas consecutivas.
- Los resultados se almacenaron en DataFrames separados con nombres claros de columnas.

```
# ----- Paso 4: Cálculo de Métricas -----
@op
def metrica_incidencia_7d(df: pd.DataFrame) -> pd.DataFrame:
    # Calculamos la incidencia diaria por cada 100k habitantes
    # y luego hacemos una media móvil de 7 días
    df["date"] = pd.to_datetime(df["date"])
    df["incidencia_diaria"] = (df["new_cases"] / df["population"]) * 100000
    df["incidencia_7d"] = df.groupby("country")["incidencia_diaria"].transform(
        lambda x: x.rolling(7, min_periods=1).mean()
    )
    # Retornamos solo las columnas necesarias renombrando para claridad
    return df[["date", "country", "incidencia_7d"]].rename(
        columns={"date": "fecha", "country": "país"}
    )

@op
def metrica_factor_crec_7d(df: pd.DataFrame) -> pd.DataFrame:
    # Calculamos la suma de casos por semana
    # Luego obtenemos la semana anterior para calcular el factor de crecimiento
    df["date"] = pd.to_datetime(df["date"])
    df["casos_semana_actual"] = df.groupby("country")["new_cases"].transform(
        lambda x: x.rolling(7, min_periods=7).sum()
    )
    df["casos_semana_prev"] = df.groupby("country")["casos_semana_actual"].shift(7)
    df["factor_crec_7d"] = df["casos_semana_actual"] / df["casos_semana_prev"]
    return df[["date", "country", "casos_semana_actual", "factor_crec_7d"]].rename(
        columns={"date": "semana_fin", "country": "país", "casos_semana_actual": "casos_semana"}
    )
```

6. Paso 5: Chequeos de Salida y Exportación

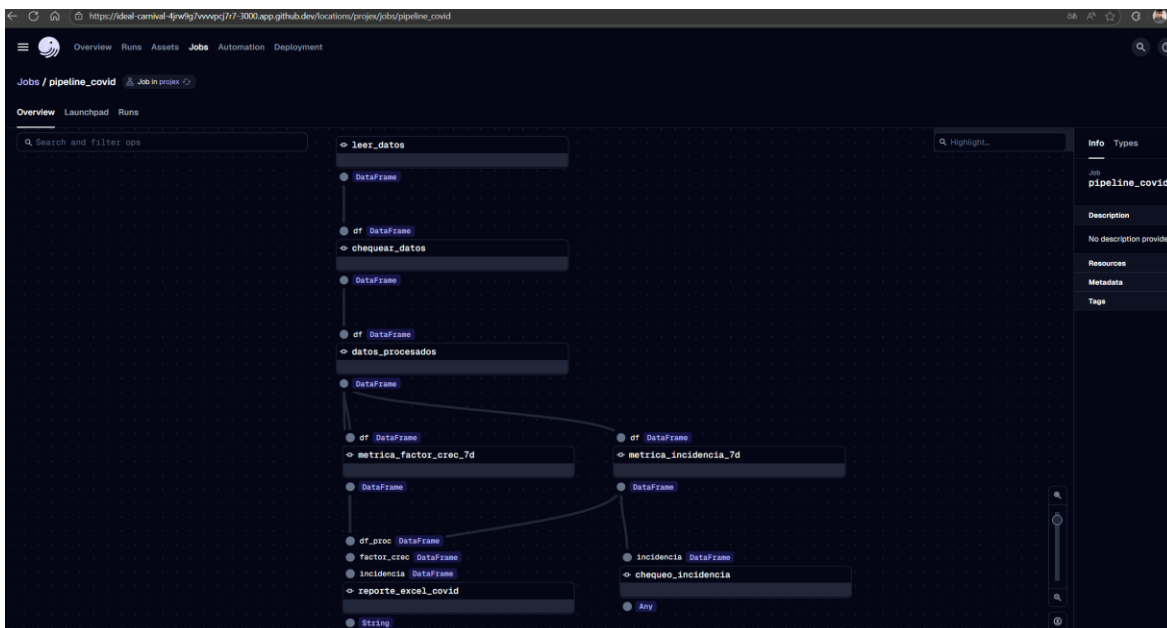
- Se verificó que la incidencia calculada estuviera dentro de un rango esperado.
- Se generó un archivo Excel con tres hojas: datos procesados, incidencia 7d y factor de crecimiento semanal.
- Este archivo sirve como reporte final para análisis y presentación de resultados.

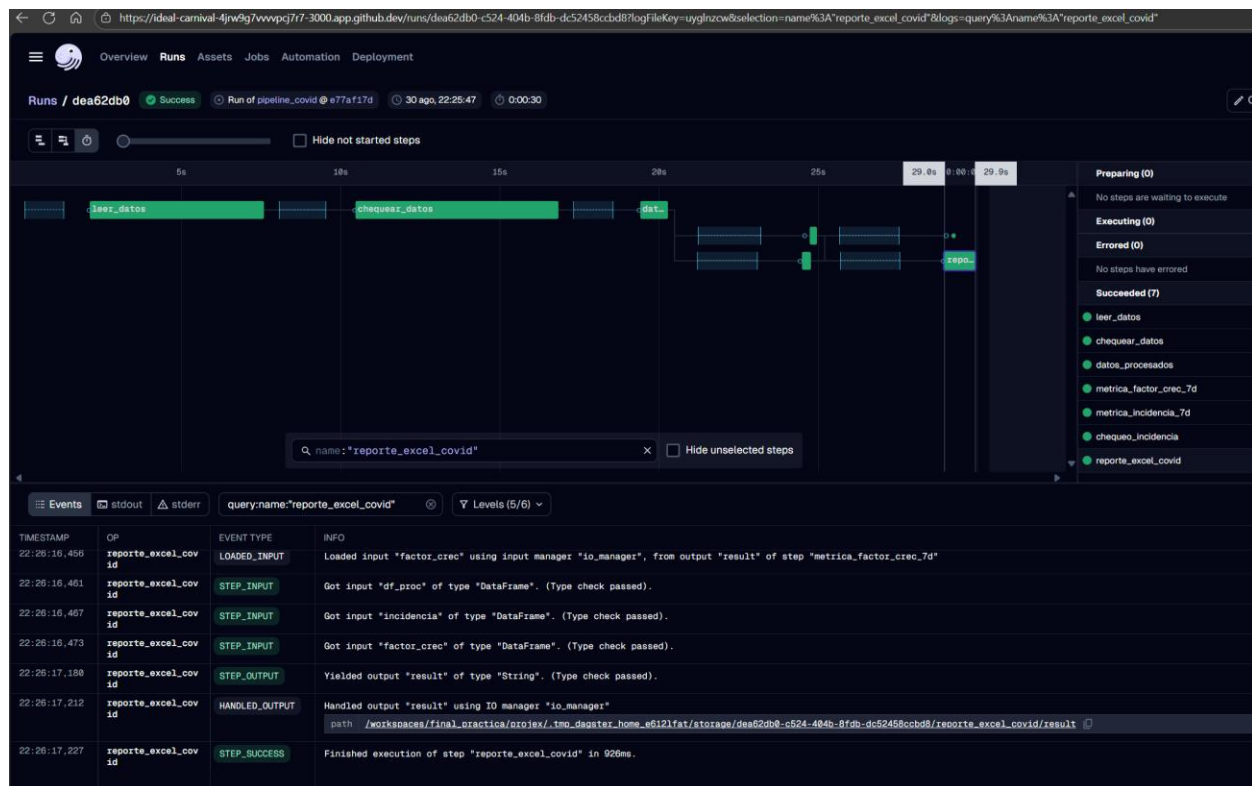
```
# ----- Paso 5: Chequeos de Salida -----
@op
def chequeo_incidencia(incidencia: pd.DataFrame):
    # Verificamos que los valores de incidencia estén dentro de un rango razonable
    min_val = incidencia["incidencia_7d"].min()
    max_val = incidencia["incidencia_7d"].max()
    if 0 <= min_val <= 2000 and 0 <= max_val <= 2000:
        print(f"Incidencia 7d dentro del rango esperado: min={min_val}, max={max_val}")
    else:
        print(f"Incidencia fuera del rango esperado: min={min_val}, max={max_val}")

# ----- Paso 6: Exportación de Resultados -----
@op
def reporte_excel_covid(df_proc: pd.DataFrame, incidencia: pd.DataFrame, factor_crec: pd.DataFrame) -> str:
    # Creamos un archivo Excel con varias hojas: datos procesados, incidencia y factor de crecimiento
    output_file = "reporte_covid.xlsx"
    with pd.ExcelWriter(output_file, engine="xlsxwriter") as writer:
        df_proc.to_excel(writer, sheet_name="datos_procesados", index=False)
        incidencia.to_excel(writer, sheet_name="incidencia_7d", index=False)
        factor_crec.to_excel(writer, sheet_name="factor_crec_7d", index=False)
    print(f"Archivo Excel generado: {output_file}")
    return output_file
```

7. Resultados y Descubrimientos

- La incidencia y el factor de crecimiento fueron consistentes con las tendencias reportadas por las autoridades.
- No se encontraron duplicados significativos ni fechas futuras.
- Las métricas permitieron identificar periodos de aumento o disminución de casos en los países analizados.





8. Conclusiones

- Se utilizó Python y Pandas por su flexibilidad para manipulación de datos y cálculo de métricas.
- Dagster permitió organizar el pipeline en pasos (ops) reproducibles y auditables.
- El reporte final en Excel facilita la comunicación de resultados y análisis posteriores.

Referencias

Our World in Data. (2025). COVID-19 Data. Recuperado de <https://ourworldindata.org/coronavirus>