💼 Presentación interna – Proyecto OPTIMET-BCN

🌐 Descripción general

OPTIMET-BCN es una aplicación interactiva desarrollada en **Python + Streamlit**, pensada para ser usada dentro de la **intranet de Telefónica Tech**.
Su objetivo es **explorar, visualizar y simular la movilidad metropolitana** de Barcelona, combinando datos de Telefónica con información externa (meteorología y eventos).

La app actúa como un **Digital Twin** del área metropolitana, donde los analistas pueden **navegar los datos**, **entender los patrones de movilidad**, **predecir tendencias** y **simular escenarios optimizados**.

🧱 Estructura general de la app

Cada sección está diseñada como un módulo independiente (uno por persona del equipo), para cubrir todo el ciclo de análisis — desde la exploración hasta la acción:

Sección	Nombre	Descripción
1. Exploración de datos	"Vista cruda"	Navegación directa por el dataset, con estadísticas básicas y exploración textual.
2. Visualizaciones generales	"Vista analítica"	Gráficas y comparativas que facilitan la comprensión de patrones temporales y espaciales.
3. Heatmap de movilidad	"Vista geoespacial"	Representación visual de los flujos entre municipios y detección de zonas con alta movilidad.
4. Clima y eventos	"Vista contextual"	Analiza cómo la meteorología y los eventos externos afectan los desplazamientos.
5. Predicción	"Vista de forecasting"	Modelos de predicción que estiman la movilidad futura a partir de los datos históricos.
6. Optimización	"Vista de simulación"	Simula políticas de mejora y escenarios "qué pasaría si" para reducir la sobrecarga de movilidad.

🎯 Objetivo final

Crear una herramienta útil, modular y visualmente atractiva que permita a los equipos de Telefónica:

- Explorar los datos de movilidad con facilidad.
- Detectar patrones y anomalías de forma visual.

- Medir el impacto de eventos y condiciones meteorológicas.
- Predecir la demanda de movilidad futura.
- Evaluar estrategias de optimización dinámica ante congestión.

🧩 ESPECIFICACIÓN DETALLADA POR SECCIÓN

1️⃣ Exploración y Calidad de Datos (`1_data_explorer.py`)

Objetivo: Dar una visión clara del contenido, calidad y cobertura temporal de los tres datasets principales.
Datasets usados: Los tres de movilidad (`municipios`, `barrios`, `mun_barrios`).

Implementaciones concretas:

- ✅ Cargar datos con `st.file_uploader()` o directamente desde `/data/` (parquet o CSV comprimido).
- ✅ Mostrar info básica:
  - Número de registros
  - Rango de fechas (`min(day)` → `max(day)`)
  - Número de municipios/barrios distintos
  - Columnas principales y tipos
- ✅ `st.metric()` para KPIs:
  - Total viajes registrados
  - Municipios con más viajes emitidos / recibidos
- ✅ Histogramas (`plotly` o `st.bar_chart`) de:
  - Viajes por día
  - Viajes por día de la semana
  - Distribución por “origen” (`Residente`, `Regional`, etc.)
- ✅ Detección rápida de outliers:
  - Días con volumen anómalo (z-score de viajes diarios)

    - Faltantes / ceros sospechosos
- ✅ Tabla resumen por dataset:
    - Nº registros, tamaño (GB), fechas, unidades espaciales
- ✅ Mini mapa de “densidad de registros por municipio” (`geopandas + pydeck`)

2️⃣ Visualizaciones Generales (`2_visual_plots.py`)

Objetivo: Mostrar las principales tendencias y patrones temporales de movilidad entre municipios y tipos de origen.
Dataset: `movilidad_municipios_2023-01_origen`

Implementaciones:

- ✅ Selector de municipio origen/destino (`st.selectbox`).
- ✅ Serie temporal de viajes diarios (Plotly line chart).
- ✅ Promedio semanal de viajes (por día de la semana).
- ✅ Comparativa por tipo de “origen” (`Residente`, `Regional`, `Nacional`, `Internacional`):
    - Gráfico de barras apiladas.
    - Evolución temporal comparada.
- ✅ Mapa de burbujas (municipios más conectados con Barcelona, por ejemplo).
- ✅ Top N municipios emisores y receptores (tabla + barras).
- ✅ Proporción de movilidad intra vs intermunicipal.
- ✅ Ratio fines de semana / laborales (barras comparativas).
- ✅ Exportar resumen visual a PNG o CSV.

3️⃣ Heatmap de Movilidad (`3_heatmap_mobility.py`)

Objetivo: Visualizar sobre el mapa los flujos de movilidad y detectar hotspots dinámicamente.
Dataset: `movilidad_municipios` o `movilidad_mun_barrios` (según escala deseada).

Implementaciones:

- ✅ Cargar shapefile o GeoJSON de municipios (ICGC o carto.cat).

- ✅ Selector de fecha o rango de fechas (`st.slider` o `st.date_input`).
- ✅ Capa **HeatmapLayer** de destinos:
    - `get_position`: coordenadas destino
    - `get_weight`: viajes
- ✅ Alternar vista “viajes totales” o “índice de carga” (`IC = viajes / media_histórica`)
- ✅ Tooltips con nombre de municipio, viajes, IC.
- ✅ Filtro de tipo de origen (`Residente`, `Regional`, etc.)
- ✅ Animación temporal (slider → día a día)
- ✅ Capa de flujos (líneas OD) para top 10 pares (Pydeck `ArcLayer`).
- ✅ Mapa centrado dinámicamente en Barcelona.
- ✅ Posible vista doble: real vs optimizada (si se integra con la sección 6).

4️⃣ Clima y Eventos (`4_weather_events.py`)

Objetivo: Analizar cómo el clima y los eventos alteran los patrones de movilidad.
Datasets:

- `movilidad_municipios` (agregado diario total)
- `weather.csv` (día, precipitación, temperatura, viento)
- `eventos.csv` (día, lat, lon, tipo, nombre, asistencia)

Implementaciones:

- ✅ Merge diario (`day`) entre movilidad y clima.
- ✅ Scatter o línea:
    - Viajes vs precipitación (efecto de la lluvia)
    - Viajes vs temperatura (calor o frío extremos)
- ✅ Calcular correlaciones simples:
    - Corr(precip, viajes)
    - Corr(temp, viajes)
- ✅ Detección visual de eventos:

    - Días con eventos grandes → resaltar en serie temporal.
- ✅ Capa mapa con eventos:
    - `ScatterplotLayer` para eventos (color por tipo: cultura, deporte, música…)
    - `HeatmapLayer` para viajes ese día.
- ✅ Ranking de impacto:
    - Δ viajes = viajes_día_evento − media_7_días_antes
- ✅ Tooltip: nombre evento, tipo, asistencia, impacto estimado.
- ✅ Mini dashboard “efecto combinado clima + evento” (por ejemplo: lluvia + concierto → caída +30%).

5️⃣ Predicción de Movilidad (`5_prediction_model.py`)

Objetivo: Crear un modelo simple para predecir el volumen de viajes futuros.
Dataset: `movilidad_municipios` (agregado diario o por par OD relevante).

Implementaciones:

- ✅ Selector:
    - Municipio origen/destino
    - Tipo de origen (Residente, Regional, etc.)
- ✅ Mostrar histórico (línea azul).
- ✅ Entrenar modelo Prophet:
    - `ds = day`
    - `y = viajes`
- ✅ Mostrar predicción ± intervalo (línea naranja + banda de confianza).
- ✅ Métricas:
    - MAE, RMSE, MAPE (últimos 30 días)
- ✅ Permitir incluir variables exógenas (lluvia, festivos) si hay tiempo.
- ✅ Gráfica “forecast horizon” configurable (1–30 días).
- ✅ Posibilidad de guardar el modelo entrenado (`pickle`) y recargarlo.
- ✅ “Comparar escenarios”: predicción normal vs con evento/lluvia.

6️⃣ Simulación y Optimización (`6_simulation_optimizer.py`)

Objetivo: Evaluar y simular qué pasaría si se aplican medidas dinámicas ante zonas con sobrecarga de movilidad.
Dataset: Cualquiera agregado diario; ideal `movilidad_municipios`.

Implementaciones:

- ✅ Calcular índice de carga (IC): ICOD,day=viajesOD,daymedia(viajesOD,histo´rico)IC_{OD,day} = \frac{viajes_{OD,day}}{media(viajes_{OD, histórico})}ICOD,day=media(viajesOD,histo´rico)viajesOD,day
- ✅ Detectar zonas “overwhelmed” (`IC > 1.2`)
- ✅ Mostrar mapa con municipios coloreados por IC.
- ✅ Políticas simulables (inputs del usuario):
    - Aumentar oferta en X% para zonas saturadas.
    - Desviar Y% de viajes a municipios vecinos.
    - Simular efecto del clima adverso (reducción del 10–20%).
- ✅ Función de optimización (simplificada):
    - Minimizar el número de OD con `IC>1` bajo restricciones de oferta.
- ✅ Mostrar resultados:
    - Heatmap “antes vs después”
    - KPIs: IC medio, nº zonas saturadas, viajes totales
- ✅ Gráfico de barras “IC real vs optimizado”
- ✅ Exportar resultados a CSV
- ✅ (Extra opcional) Usar `ortools` o `scipy.optimize.minimize` para resolver un MIP pequeño.

🧱 BONUS (COMÚN A TODAS LAS SECCIONES)

Funciones utilitarias (en `/utils/`):

- `load_data()`: carga y cachea datasets (`st.cache_data`)
- `merge_weather_events()`: combina movilidad + clima + eventos

- `get_geo_data()`: devuelve geometrías de municipios
- `compute_IC()`: índice de carga histórico
- `plot_time_series(df, municipio)`: función común de líneas
- `map_heat(df)`: mapa estándar con Pydeck

💡 Distribución de trabajo sugerida

Persona	Rol principal	Dependencia
1	Limpieza / Exploración	Base para todos
2	Plots / Tendencias	Usa dataset limpio
3	Mapa / Heatmap	Usa datos de #1 y #4
4	Clima / Eventos	Cruza datos de #1
5	Predicción	Usa series de #1 y #4
6	Simulación	Usa resultados de #1 y #5