Manual: Creación, Planificación y Ejecución de Pipelines con Aitea Building LAB

Este manual le guiará a través del proceso de configuración y ejecución de un pipeline.

1. Conceptos Clave

- **Pipeline:** Una secuencia de pasos de procesamiento de datos, donde la salida de un paso se convierte en la entrada del siguiente. Aitea Building Lab utiliza la implementación de sklearn.pipeline.Pipeline.
- **pipeline_executor.py:** La clase principal (PipelineExecutor) para gestionar la creación, configuración y ejecución de pipelines. Se encarga de cargar la configuración, establecer conexiones a las fuentes de datos, preparar los datos y ejecutar las etapas del pipeline, posiblemente en paralelo.
- **pipe_plan.json:** Un archivo de configuración que define la estructura y los parámetros de su pipeline, incluyendo los pasos y las fuentes de datos.
- **confort_analytics.py:** Contiene las clases ConfortAnalyticsFuse y ConfortAnalytics, que son componentes (pasos) que se pueden integrar en un pipeline para realizar análisis de confort.

2. Planificación del Pipeline (pipe_plan.json)

El archivo pipe_plan.json es crucial para definir su pipeline. Aquí se detalla la estructura del ejemplo proporcionado:

```
JSON
{
  "confort": { // Nombre del pipeline
    "steps": {
      "confort_analytics.ConfortAnalyticsFuse": {}, // Primer paso: preparación y fusión de datos
      "confort_analytics.ConfortAnalytics": { // Segundo paso: análisis de confort
        "thresholds": {"a": 1.0} // Parámetros para este paso
      }
    },
    "data_sources": { // Configuración de las fuentes de datos
      "influxdb": {
        "buckets": [
          "castellana_163"
        ],
```

```
"range": {
  "start": "2025-07-10T09:00:00.000Z",
  "stop": "2025-07-10T10:00:00.000Z"
},
"filter_measurement": [{
  "measurement": "climatization"
}, {"measurement": "climate_control"}],
"tag_is": [
  {
    "tag_name": "element",
    "tag_value": "vrv"
  },
  {
    "tag_name": "element",
    "tag_value": "fancoil"
  },
  {
    "tag_name": "element",
    "tag_value": "tricombined_probe"
  }

],
"filter_field": [
  {
    "field": "room_temperature"
  },
  {
    "field": "room_humidity"
  },
  {
    "field": "general_condition"
  },
  {
    "field": "room_co2"
  }


],
"window_aggregation": {
    "every": "10m",
    "function": "mean",
    "create_empty": false
},
"keep_columns": {
  "columns": [
    "element",
```

```
            "_time",
            "_value",
            "_field",
            "floor"
          ]
        }
      },
      "postgresql": [
        "SELECT tobm.* from public.typical_ocupation_by_metric AS tobm"
      ],
      "local": [
        "building_id.csv"
      ]
    }
  }
}
```

- **"confort":** Es el nombre de su pipeline. Puede definir múltiples pipelines dentro del mismo archivo JSON.
- **"steps":** Define los pasos del pipeline en orden secuencial.
    - Cada clave es la ruta completa de la clase del paso (por ejemplo, "confort_analytics.ConfortAnalyticsFuse").
    - El valor es un diccionario de parámetros para inicializar esa clase.
- **"data_sources":** Especifica las fuentes de datos que cada paso del pipeline utilizará.
    - **"influxdb":** Configura una conexión a InfluxDB, especificando buckets, rangos de tiempo, filtros de medición y campos, agregación y columnas a mantener.
    - **"postgresql":** Define una consulta SQL para obtener datos de PostgreSQL.
    - **"local":** Lista los archivos locales que se utilizarán como fuentes de datos.

3. Ejecución del Pipeline (pipeline_executor.py)

La clase PipelineExecutor en pipeline_executor.py es la encargada de orquestar la ejecución de los pipelines.

Inicialización:

Para crear una instancia de PipelineExecutor, necesita proporcionar la ruta a su archivo de configuración del pipeline:

```
Python
```

```
pipeline = PipelineExecutor(pipeline_config_file="pipes_schedules/pipe_plan.json", generate_so=False)
```

- pipeline_config_file: La ruta a su archivo JSON de configuración del pipeline.
- total_processing (opcional): El número de procesos a utilizar para la ejecución en paralelo (por defecto es 4).
- generate_so (opcional): Si se debe generar un objeto compartido (.so) después de que el modelo se haya ajustado (por defecto es True).
- save_in_joblib (opcional): Si se debe guardar el pipeline en formato joblib (por defecto es False).

Proceso de Ejecución:

1. **Carga de Configuración:** El PipelineExecutor carga la configuración global y la configuración del pipeline del archivo JSON.
2. **Creación de Conexiones:** Se establecen conexiones a las fuentes de datos (InfluxDB, PostgreSQL, archivos locales) según lo definido en la configuración global y del pipeline. Si las "Aitea Connectors" no están disponibles, solo se podrán usar archivos locales.
3. **Creación de Pipelines:** Por cada pipeline definido en pipe_plan.json, se crea un objeto sklearn.pipeline.Pipeline y se asocia la información de las fuentes de datos necesarias para su entrenamiento.
4. **Preparación de Datos (data_preparation):** Antes de ejecutar cada pipeline, se prepara la data. La clase ConfortAnalyticsFuse juega un papel clave aquí, fusionando los datos de las diferentes fuentes (InfluxDB, PostgreSQL, local) en un único DataFrame. Esto es manejado por el primer paso del pipeline que hereda de MetaFuse.
5. **Ejecución de Pipelines (pipes_executor):**
    a. Se utiliza un multiprocessing.Pool para ejecutar las tareas del pipeline en paralelo, si total_processing es mayor que 1.
    b. Para cada pipeline, se llama al método lab_fit (una función auxiliar) con los datos preparados y el objeto pipeline.
    c. **action:** Define la acción a realizar ("fit", "predict", "fit_predict", etc.). En el ejemplo, se usa "fit_predict".
6. **Manejo de Tareas (task_handler, error_handler):**
    a. task_handler: Se llama cuando una tarea de pipeline finaliza con éxito. Guarda el pipeline ajustado y opcionalmente crea un objeto compartido.
    b. error_handler: Se llama si ocurre un error durante la ejecución de una tarea del pipeline.

Ejemplo de Ejecución:

```python
if __name__ == "__main__":
    pipeline = PipelineExecutor(pipeline_config_file="pipes_schedules/pipe_plan.json", generate_so=False)
    pipeline.pipes_executor(action="fit_predict")
```

Este bloque de código creará una instancia de PipelineExecutor, cargará la configuración del pipeline desde pipe_plan.json y luego ejecutará el pipeline "confort" en modo "fit_predict".

4. Cálculos en confort_analytics.py (Visión General)

El archivo confort_analytics.py contiene dos clases principales que actúan como pasos en el pipeline: ConfortAnalyticsFuse y ConfortAnalytics.

- **ConfortAnalyticsFuse (Fusión de Datos):**
  - Hereda de MetaFuse, lo que indica que su función principal es fusionar diferentes fuentes de datos.
  - El método fuse_data_sources es el responsable de:
    - Obtener datos de InfluxDB (datos de climatización como temperatura, humedad, CO2, y condición general de los elementos).
    - Obtener datos de PostgreSQL (ocupación típica por métrica).
    - Obtener datos de archivos locales (ID de edificios).
    - Combinar estos datos en un único DataFrame para su posterior análisis. Esto incluye filtrar los datos de ocupación basándose en el mes, día de la semana y hora de los datos de climatización, y mapear los IDs de los edificios.
- **ConfortAnalytics (Análisis de Confort):**
  - Hereda de MetaModel, lo que sugiere que es un modelo que puede ser "ajustado" y "predecido".
  - **fit y fit_predict:** Estos métodos preparan los datos para el análisis de confort.
    - El método clave es _calculate_data_matrix, que:
      - Identifica los momentos en que la "planta" (sistema de climatización) está "encendida" basándose en la condición general de los fancoils y un umbral de porcentaje.
      - Filtra los datos de temperatura, humedad y CO2 solo para cuando la planta está activa.
      - Aplica umbrales a los valores de humedad, CO2 y temperatura para asegurar la validez de los datos.

- Calcula las medias de humedad, temperatura y CO2 por piso y hora.
- Utiliza la función _pmv_ppd_extimate para calcular los valores de PMV (Predicted Mean Vote) y PPD (Predicted Percentage of Dissatisfied) para verano e invierno, basados en la temperatura y la humedad. PMV y PPD son métricas comunes para evaluar el confort térmico.
- Genera dos DataFrames principales: uno con la "matriz de datos" a nivel de piso y hora, y otro con las medias a nivel de bucket (edificio) y hora.

- **predict:**
  - Utiliza la data_matrix calculada durante el fit para comparar los valores actuales de PMV/PPD con un "estado base" o "típico" del edificio/piso.
  - Calcula las diferencias entre los valores de PMV/PPD observados y los valores de la matriz de datos, tanto a nivel de edificio (_predict_entire_bucket) como a nivel de piso (_predict_floor_bucket).
  - Estos resultados indican qué tan lejos están las condiciones de confort actuales de las condiciones "típicas" o "esperadas" en diferentes partes del edificio y en diferentes horas.

En resumen, el pipeline definido en pipe_plan.json utiliza ConfortAnalyticsFuse para recopilar y unificar los datos de diversas fuentes, y luego ConfortAnalytics procesa esos datos para calcular el confort térmico (PMV y PPD) y determinar las desviaciones de un estado de confort de referencia, proporcionando información valiosa sobre el rendimiento del sistema de climatización y el bienestar de los ocupantes. Como hemos ilustrado con este ejemplo es sumamente fácil construir una librería compilada y que contiene toda la información necesaria para ser ejecutada en un entorno de producción. En posteriores formaciones ilustraremos como usar el resto de los elementos de Aitea Building Lab para testear la librería creada.