perplexity

from langgraph.graph import Graph, START, END

from ..nodes import ingest_inputs, validate_schema, neuro_parse, score_engine, generate_report
from ..state import PipelineState

def build_graph() -> Graph:
graph = Graph()

```
# 노드 등록
graph.add_node("ingest_inputs", ingest_inputs.run)
graph.add_node("validate_schema", validate_schema.run)
graph.add_node("neuro_parse", neuro_parse.run)
graph.add_node("score_engine", score_engine.run)
graph.add_node("generate_report", generate_report.run)

# 연결 순서 (Edges)
graph.add_edge(START, "ingest_inputs")
graph.add_edge("ingest_inputs", "validate_schema")
graph.add_edge("validate_schema", "neuro_parse")
graph.add_edge("neuro_parse", "score_engine")
graph.add_edge("score_engine", "generate_report")
graph.add_edge("generate_report", END)

return graph
```

def run_graph(forms_csv: str, neuro_pdf: str):
graph = build_graph()
state = PipelineState()
input_payload = {"forms_csv": forms_csv, "neuro_pdf": neuro_pdf}

```
# Graph 실행
result = graph.invoke(input_payload, state)
print(f"✅ Graph completed. Report: {result.report}")
return result
```

해석해줘

이 코드는 **LangGraph** 라이브러리를 사용해 일련의 데이터 처리 과정을 그래프로 정의하고 실행하는 파이프라인을 만드는 예시입니다. 한 단계씩 해석해볼게요.

1. 주요 목적

이 스크립트는 `forms_csv`와 `neuro_pdf`라는 두 입력 파일을 받아서,
1️⃣ 데이터를 불러오고(`ingest_inputs`),
2️⃣ 형식을 검증(`validate_schema`),
3️⃣ 신경학적 분석(`neuro_parse`),
4️⃣ 점수 계산(`score_engine`),
5️⃣ 보고서 생성(`generate_report`)하는 **파이프라인(Graph)**을 정의합니다.

2. `build_graph()` 함수

이 함수는 실제로 그래프(노드 간 연결 구조)를 만듭니다.

- `Graph()` 객체를 생성합니다.
- 각 단계 함수를 **노드(node)**로 등록합니다:

```
graph.add_node("ingest_inputs", ingest_inputs.run)
graph.add_node("validate_schema", validate_schema.run)
...
```

- 각 단계 간의 **연결(Edge)**을 정의합니다. 즉, 어떤 노드가 끝난 후 어떤 노드로 이동할지 결정합니다:

```
graph.add_edge(START, "ingest_inputs")
graph.add_edge("ingest_inputs", "validate_schema")
...
graph.add_edge("generate_report", END)
```

➡ 이런 식으로 시작(START)부터 끝(END)까지의 처리 순서를 연결한 형태로 파이프라인을 구성한 것입니다.

3. `run_graph(forms_csv: str, neuro_pdf: str)` 함수

이 함수는 위에서 만든 그래프를 실제로 **실행**합니다.

- `build_graph()`로 그래프 구조를 만든 뒤,
- 초기 상태(`PipelineState()`)를 생성합니다.
- 입력 데이터를 사전(`dict`) 형태로 전달합니다.

```
input_payload = {"forms_csv": forms_csv, "neuro_pdf": neuro_pdf}
```

- `graph.invoke(input_payload, state)`를 통해 그래프를 실행하면,
  모든 노드들이 순서대로 실행되어 하나의 `result` 객체를 반환합니다.

마지막으로 결과 보고서(`result.report`)를 출력하고 반환합니다:

```
print(f"✅ Graph completed. Report: {result.report}")
return result
```

요약

이 스크립트는 **LangGraph 파이프라인 정의 예시**로, 다음과 같은 작업 흐름을 갖습니다:

```
START → ingest_inputs → validate_schema → neuro_parse → score_engine → generate_repor
```

즉, 데이터를 받아서 처리·분석·채점·보고서 생성까지 일련의 단계를 자동화한 구조입니다.