# 데이터 분석-http post-브라우저 파이프라인

| | |
|---|---|
| 생성자 | mejin |
| 생성 일시 | @2025년 10월 10일 오후 5:47 |
| 카테고리 | 백엔드 웹 개발 파이프라인 |
| 최종 편집자: | mejin |
| 최종 업데이트 시간 | @2025년 10월 10일 오후 6:01 |

리액트는 수신 서버가 아니라서, 주피터에서 리액트로 바로 보내기 위해서는 백엔드(HTTP API)가 필요하다.

구조는 다음과 같다

> 주피터 → HTTP POST → 백엔드(Node.js>Express) → 저장 → 리액트가 fetch 또는 axios로 읽어 랜더링

## Node/Express 백엔드 (가장 간단, 프런트 곁에 붙이기)

```
// server.js
import express from "express";
import cors from "cors";
import fs from "fs/promises";
const app = express();
app.use(cors());              // 개발 단계: 출처 허용
app.use(express.json({ limit: "5mb" }));

app.post("/api/hist", async (req, res) => {
  await fs.writeFile("./data/hist.json", JSON.stringify(req.body, null, 2), "utf-8");
```

```
  res.json({ ok: true });
});

app.post("/api/corr", async (req, res) => {
  await fs.writeFile("./data/corr.json", JSON.stringify(req.body, null, 2), "utf-8");
  res.json({ ok: true });
});

// React가 읽게 GET도 열어두기
app.get("/hist.json", async (_req, res) => {
  res.sendFile(process.cwd() + "/data/hist.json");
});
app.get("/corr.json", async (_req, res) => {
  res.sendFile(process.cwd() + "/data/corr.json");
});

app.listen(3001, () => console.log("API on http://localhost:3001"));
```

## 2) 리액트 (프런트)

- 기존처럼 fetch("/hist.json")를 쓰고 싶다면 프록시 설정(예: Vite devServer.proxy, CRA의 proxy)로 /hist.json → http://localhost:3001/hist.json 로 우회하거나, **URL을 직접** fetch("http://localhost:3001/hist.json")로 바꿔도 됩니다.

## 3) 주피터/코랩에서 "푸시" (POST)

```
import requests, json

# 앞서 만든 hist_payload, corr_payload 사용
requests.post("http://localhost:3001/api/hist", json=hist_payload, timeout=10)
requests.post("http://localhost:3001/api/corr", json=corr_payload, timeout=10)
print("업로드 완료")
```

> 주피터에서 post하는 코드를 추가해야 한다!

그리고 `json.dump()`를 사용해서 파일을 로컬에 저장할 필요는 없다. 곧바로 서버로 POST 전송하면 된다.

### 💡 POST 방식 구조 (요약)

```
[ Jupyter / Colab ]
   ↓ POST (JSON)
[ 백엔드 서버 (Express / FastAPI 등) ]
   ↓ 파일 또는 DB에 저장
[ React 프런트엔드 ]
   ↓ fetch("/hist.json") 또는 fetch("/api/hist")
   ↓ Plotly / ChartJS / ECharts 로 시각화
```

### 🔷 예시 코드 (파일 저장 없이 서버로 바로 전송)

```python
import numpy as np, pandas as pd, requests

# 예시 데이터
dd = pd.DataFrame({
    "ph": np.random.normal(7, 0.4, 1000),
    "temp": np.random.normal(40, 5, 1000),
    "current": np.random.normal(10, 1.5, 1000),
})

num_cols = [c for c in ["ph","temp","current"] if c in dd.columns]

# 히스토그램 데이터
hist_payloads = {}
for col in num_cols:
    counts, bin_edges = np.histogram(dd[col].dropna(), bins=30)
    hist_payloads[col] = {
```

```
        "title": f"{col} Histogram",
        "counts": counts.tolist(),
        "bin_edges": bin_edges.tolist(),
        "xlabel": col,
        "ylabel": "Count"
    }

# 상관계수 데이터
corr = dd[num_cols].corr()
corr_payload = {
    "title": "Correlation Heatmap",
    "z": corr.values.tolist(),
    "x": corr.columns.tolist(),
    "y": corr.index.tolist()
}

# ✅ 바로 서버로 POST (파일 저장 없이)
requests.post("http://localhost:3001/api/hist", json=hist_payloads)
requests.post("http://localhost:3001/api/corr", json=corr_payload)
print("서버로 전송 완료 ✅")
```

### ② 서버 (예: Express)

```
import express from "express";
import cors from "cors";
import fs from "fs/promises";
const app = express();
app.use(cors());
app.use(express.json({ limit: "5mb" }));

app.post("/api/hist", async (req, res) => {
  await fs.writeFile("./data/hist.json", JSON.stringify(req.body, null, 2));
  res.json({ ok: true });
});

app.post("/api/corr", async (req, res) => {
```

```
  await fs.writeFile("./data/corr.json", JSON.stringify(req.body, null, 2));
  res.json({ ok: true });
});

app.get("/hist.json", async (_req, res) => res.sendFile(process.cwd() + "/dat
a/hist.json"));
app.get("/corr.json", async (_req, res) => res.sendFile(process.cwd() + "/da
ta/corr.json"));

app.listen(3001, () => console.log("🚀 API 서버 구동: http://localhost:3001"));
```

### ③ 리액트 (프런트)

```
useEffect(() => {
  fetch("http://localhost:3001/hist.json")
    .then(res => res.json())
    .then(setHist);
}, []);
```

---

## 최종 파이프라인 요약

Jupyter/Colab에서 requests.post() → Express 서버(Node.js)

Express가 JSON 저장 →

React가 fetch()로 읽어서 시각화