Data Science

Advanced Visualization

노기섭 교수

(kafa46@hongik.ac.kr)

Lecture Goals

- 시각화에 대한 기본 지식
- 대시보드 설계 원칙
- 언어별 시각화 도구
- Plotly 소개
- Dash 소개
- Plotly와 Dash 연동
- 대시보드 구현을 위한 다양한 선택
- 대시보드 구현

시각화에 대한 기본 지식

시각화 - 커뮤니케이션 도구

시각화는 커뮤니케이션 도구이다!

- 복잡한 수치, 표, 모델 결과를 한눈에 이해할 수 있도록 돕는 핵심 커뮤니케이션 도구
- 이해관계자가 정확하게 판단할 수 있도록 돕는 역할
- 좋은 시각화
  - 별도의 설명이 없어도 핵심 메시지가 자연스럽게 전달
  - 숫자의 나열을 이해에서 공감으로 전환시키며, 시각적 설득력을 높여 줌
- 효과적인 시각화
  - 리포트, 프레젠테이션, 대시보드에서 핵심 정보의 요약, 강조, 경고를 시각적 신호로 전달

커뮤니케이션 도구

대시보드 – 인사이트 전달

- 이해와 행동을 이끌어내는 스토리라인
- 단계와 역할
  - **Context** 단계

    배경과 가설을 제시하며 시각화의

    목적과 분석의 방향성 제시
  - **Insight** 단계

    차트를 통해 데이터가 말하는 핵심 결과를 전달
  - **Action** 단계: 분석을 통해 도출된 실행 방안 또는 개선 방향을 제시

왜 이 차트를 선택했는가?

■ 차트 선택 기준 체크리스트

고려 요소	핵심 질문	예시
데이터 유형	데이터는 수치형인가, 범주형인가?	연속형 vs 범주형
분석 목표	비교, 분포, 추세, 상관, 구성 중 무엇인가?	추세 파악, 상관 분석 등
강조 포인트	무엇을 가장 강조할 것인가?	이상치, 변화율, 상관구조
대상의 문해력	독자의 데이터 이해 수준은 어떤가?	단순 vs 고급 차트
맥락/제약	어떤 환경에서 사용되는가?	보고용, 인쇄용, 모바일 등

왜 이 차트를 선택했는가?

목적에 따른 차트 선택

목적	흔한 실수(부적절)	권장 차트(적절)
시간 추세	막대 그래프	선(line), 영역(area)
두 변수 관계	막대 그래프	산점도(scatter), 회귀선
분포 파악	선 또는 막대 그래프	히스토그램, KDE, 바이올린
구성비 비교	선 그래프	파이/도넛, 스택형 막대
순위 비교	파이 차트	가로 막대
다변량 상관	다중 선 그래프	히트맵, 페어플롯, 평행좌표

좋은 시각화

좋은 시각화

- 정보를 명확하게 전달하며, 관찰자가 올바르게 해석할 수 있도록 설계
  - 축, 단위, 범례가 명확하게 표시
  - 불필요한 시각적 요소를 줄이고 데이터에 집중할 수 있도록 설계
  - 색상은 의미를 전달하기 위해 사용되며 색약·색맹을 고려
  - 관찰자가 다음 행동을 떠올릴 수 있도록 방향성을 제시
  - 핵심 메시지를 주석(Annotation)으로 강조

나쁜 시각화

나쁜 시각화: 데이터의 의미를 흐리거나 왜곡

- 3D 효과나 과도한 장식으로 정보 전달이 방해
- 파이 차트 조각이 많아 비교가 어려워짐
- 메시지 없이 차트만 나열되어 해석 어려움
- Y축 절단 등으로 시각적 왜곡을 유발

대시보드 설계 원칙

대시보드(Dashboard)?

대시보드

- 다양한 데이터를 한눈에 파악할 수 있도록 핵심 정보를 시각적으로 정리해 제공하는 화면
- 사용자가 빠르게 현재 상황을 이해하고, 필요한 의사결정을 내릴 수 있도록 돕는 것

조금 더 생각해 보는 대시보드

대시보드의 중요성

- 문제 해결과 의사결정에 최적화된 정보 구조를 제공하는 데이터 커뮤니케이션 도구
  - 어떤 지표를 보여줄지,
  - 어떤 시각화 형태를 사용할지,
  - 어떤 흐름으로 배치할지에 대한 설계가 매우 중요

대시보드의 활용

- 경영, 마케팅, 운영 관리, 제조, 교육, IT 서비스 모니터링 등 다양한 분야에서 활용
- 사용자 역할에 따라 구성 방식이 다름.
  - 경영진: 요약된 KPI 중심의 전략적 대시보드
  - 실무자: 세부 지표와 실행 기반의 운영형 대시보드

좋은 대시보드 사례

Click me!

나쁜 대시보드 사례

How to generate Bad Dashboard UI Examples

Click me!

Find bad dashboard from gallery

비교 요약

항목	좋은 대시보드	나쁜 대시보드
정보 우선순위	핵심 지표 위주, 시선 흐름 설계	너무 많은 지표가 동시에 노출
시각적 복잡성	단순, 가독성 중심, 여백 활용	장식 과다, 3D, 시각적 피로 유발
해석 용이성	축/단위/범례 명확, 직관적	정보 해석 난해, 혼란 초래
의사결정	인사이트 → 행동으로 연결	보여주기용, 실행 방향 제시 없음
사용자 경험	빠른 이해 및 즉시 대응 가능	사용자가 외부로 데이터 추출해 재가공

레이아웃 구성 원칙

대시보드 레이아웃

- 레이아웃 구성 → 정보 전달력과 사용자의 해석 속도가 달라짐
- 효과적인 레이아웃은 사용자의 시선 흐름을 고려
  - 핵심 정보 → 세부 정보 → 인사이트 확인
  - 의미 연결이 자연스러운 구조를 만드는 것이 필수

시각적 배분 전략

- 시각적 우선순위와 정보 배치를 전략적으로 구성
  - 3-Panel Layout
  - F-Pattern
  - Z-Pattern

3-Panel Layout

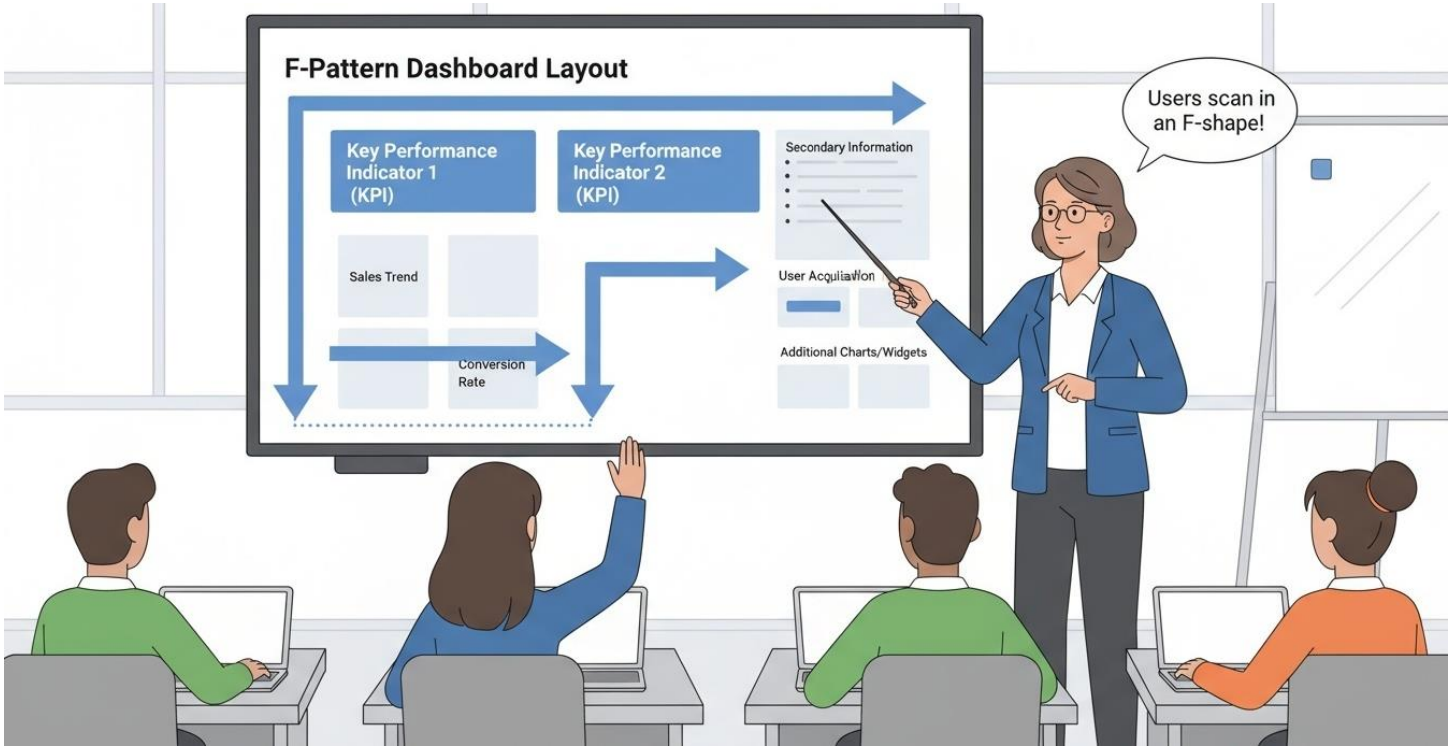
■ **화면을 상단, 좌측, 우측으로 나누어 정보 계층을 명확하게 구조화하는 방식**

- 상단: 핵심 KPI와 전체 상황 요약을 배치 → 첫눈에 현황을 파악 도움
- 좌측: 세부 지표, 추세 그래프, 상세 분석 정보를 배치 → 원할 때 더 깊이 들어가 확인
- 우측: 알림, 인사이트, 액션 아이템, 컨트롤 패널을 구성 → 해석 이후 행동 연결

F-Pattern Layout

■ 사용자의 시선 흐름이 “F” 형태로 움직인다는 웹·UI 연구 결과에 기반한 배치 방식

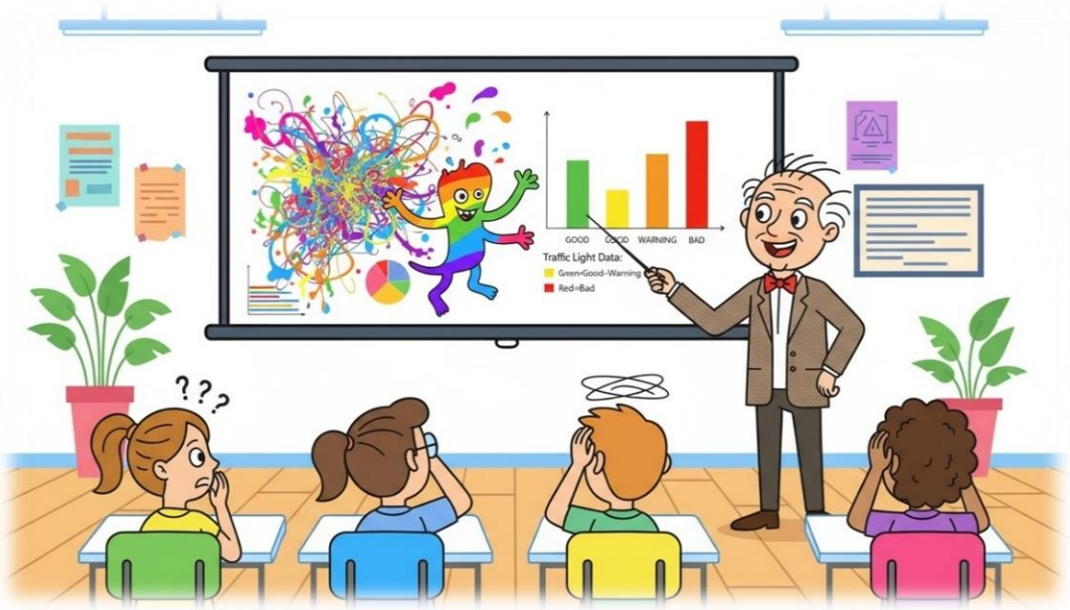
- 사용자는 화면의 상단 왼쪽에서 시작해 오른쪽으로 시선을 이동 → 다시 왼쪽으로 내려오며 스캔
- 핵심 KPI와 제목, 중요한 메시지를 상단 왼쪽에 배치하고, 그 옆에 보조 정보 배치, 인식 속도 향상



Z-Pattern Layout

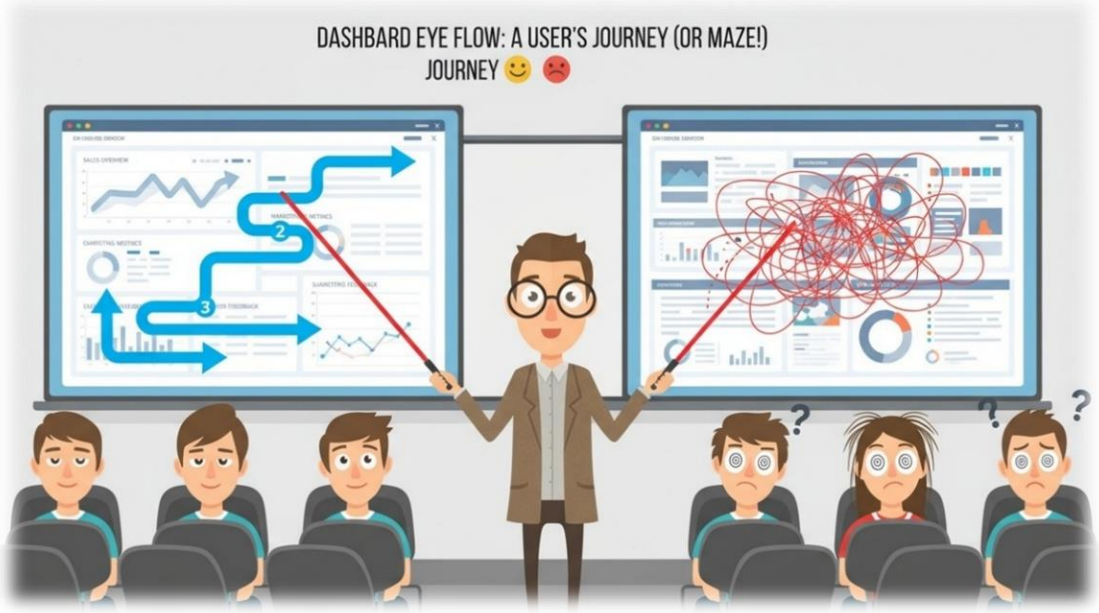
- **시선이 “Z” 형태로 상단 왼쪽 → 상단 오른쪽 → 하단 왼쪽 → 하단 오른쪽으로 이동**
- **콘텐츠 양 ↓, 메시지 전달이 명확해야 하는 발표형 대시보드, 단일 화면 요약 보고서**
  - 상단: 제목, 기간, 필터, 핵심 메시지를 배치
  - 중앙: 주요 시각 정보를 배치
  - 하단: 결론, 다음 행동, 또는 CTA(Call to Action) 배치

색상의 활용



- 색상은 의미를 전달하기 위한 도구이며 장식용이 아니다.
- 데이터의 상태나 중요도에 따라 색상을 일관성 있게 사용 (초록: 좋음, 노랑: 주의, 빨강: 위험)
- 색상 수는 최소화 → 시각적 복잡성 ↓, 색각 이상 사용자를 고려한 팔레트를 선택
- 강조해야 할 데이터를 다른 색상으로 처리 → 사용자의 눈이 해당 요소에 집중되도록 설계

색상의 활용



- 사용자의 시선 이동 경로를 고려하여 정보 배치를 설계
- 일반적으로 시선은 왼쪽 상단에서 시작해 오른쪽과 아래 방향으로 이동
- 중요한 정보는 시선이 가장 먼저 닿는 위치에 배치, 세부 정보는 그 다음 순서에 배치
- 여백(Whitespace)과 정렬을 활용해 자연스러운 시선 흐름을 유도

데이터 강조(Highlighting)

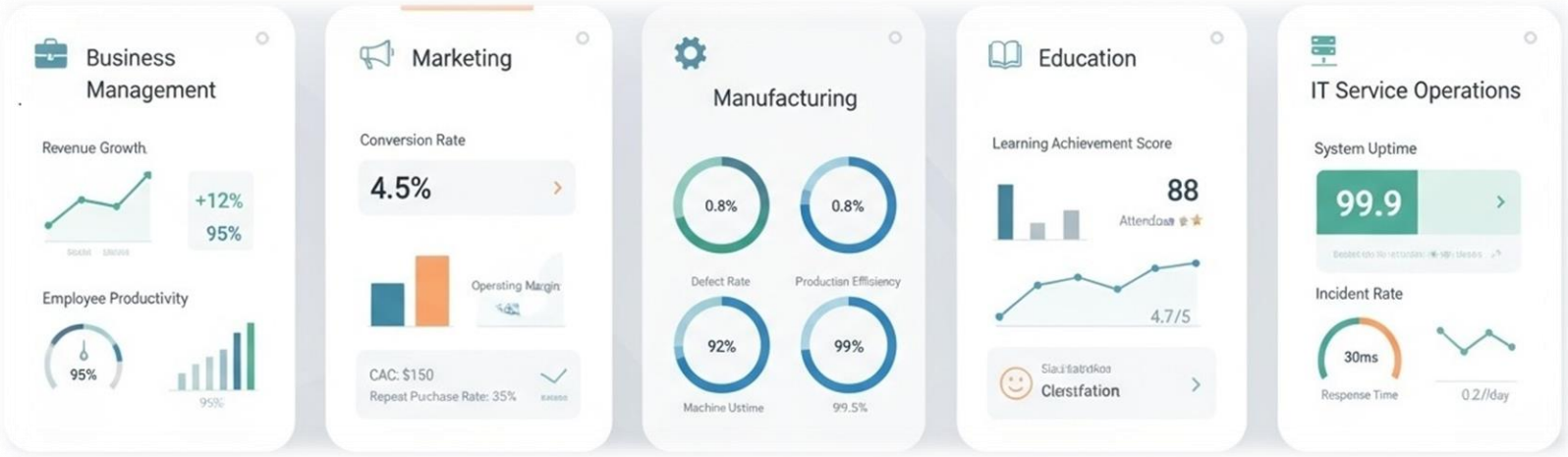
- 차트나 지표에서 핵심 데이터 포인트를 색상, 굵기, 크기 변화, 주석으로 명확하게 강조
- 트렌드 변화나 이상치 등 주목해야 하는 요소는 시각적 신호를 통해 부각시키도록 처리
- 불필요한 요소는 최소화하여 데이터 자체가 강조될 수 있는 환경으로 구성
- 강조는 한 화면에 한두 개만 사용해 과도한 강조로 인한 혼란을 방지

KPI 선정 전략

KPI(Key Performance Indicator, 핵심 성과 지표)?

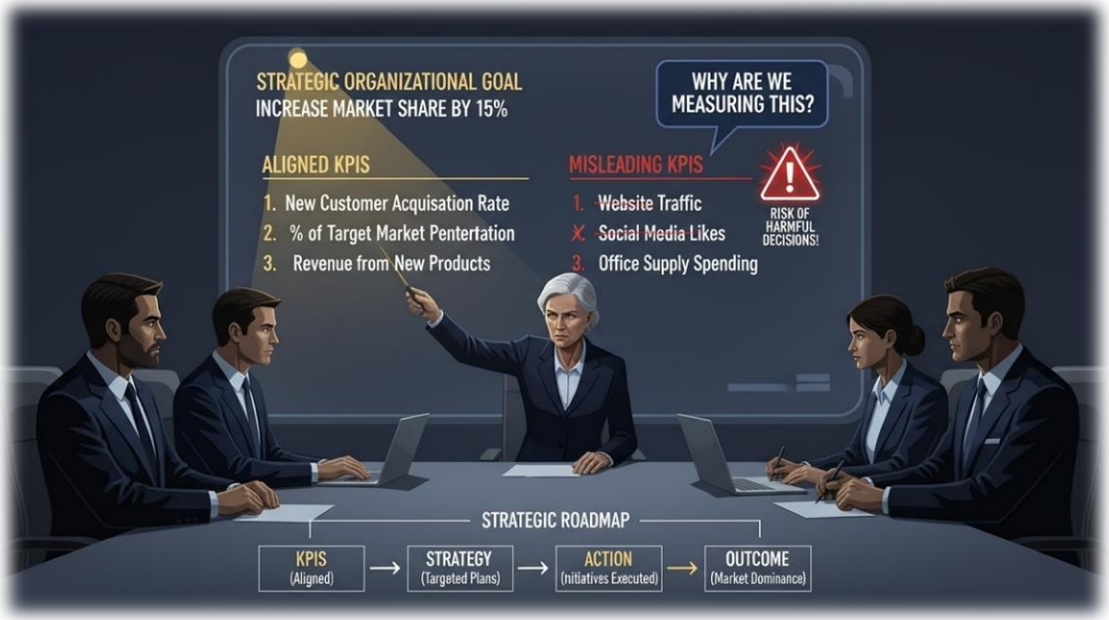
- 조직이나 팀이 달성해야 할 목표의 성과를 측정하고 평가하기 위한 핵심 지표
    - 단순한 데이터 지표가 아니라, 전략과 실행을 연결하는 나침반 역할
- KPI는 무엇을 달성해야 하는지 방향성을 제시
- 조직 구성원이 동일한 목표를 향해 움직일 수 있도록 기준을 제공
- "성과가 좋다" 또는 "나쁘다"를 확인하는 수준을 넘어, 목표 달성 과정에서 필요한 행동을 이끌어내는 실행 중심의 지표

KPI의 활용 분야



- 기업 경영, 마케팅, 영업, 재무, 인사, 교육, IT 서비스 운영, 제조 등 다양한 분야에서 활용
- 마케팅: 전환율, 고객 획득 비용, 재구매율 등을 통해 캠페인의 성과를 측정
- 제조: 불량률, 생산성, 장비 가동률 등으로 운영 효율성과 품질을 관리
- 교육: 학습 성취도, 출석률, 수업 만족도 등을 기반으로 교육의 효과성과 개선 방향을 판단
- IT 서비스: 시스템 가용성, 응답 속도, 장애 발생률 등으로 서비스 품질과 사용자 경험을 관리

좋은 KPI: 조직 목표와 직접 연결된 KPI 선택

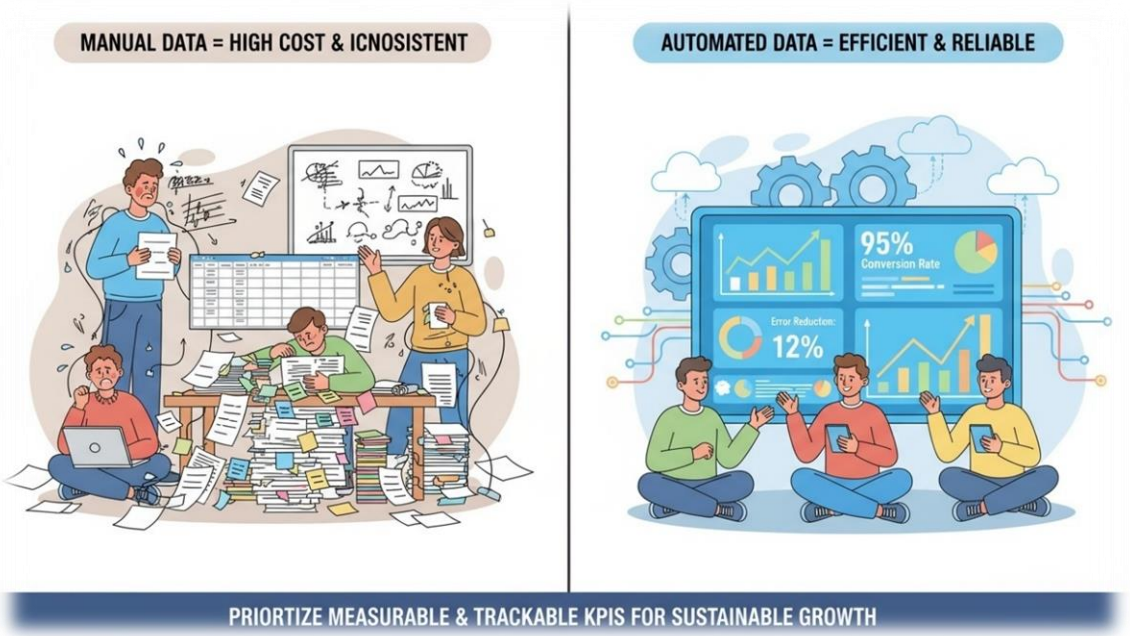


- KPI는 회사 또는 팀의 핵심 목표와 직접적으로 연관되어야 한다.
- "왜 이 지표를 측정하는가?"라는 질문에 명확히 답할 수 있어야 한다.
- 조직의 전략 방향성과 일치하지 않는 지표는 과감히 제외한다.

좋은 KPI: 성과에 영향을 미치는 선행지표와 후행지표 균형

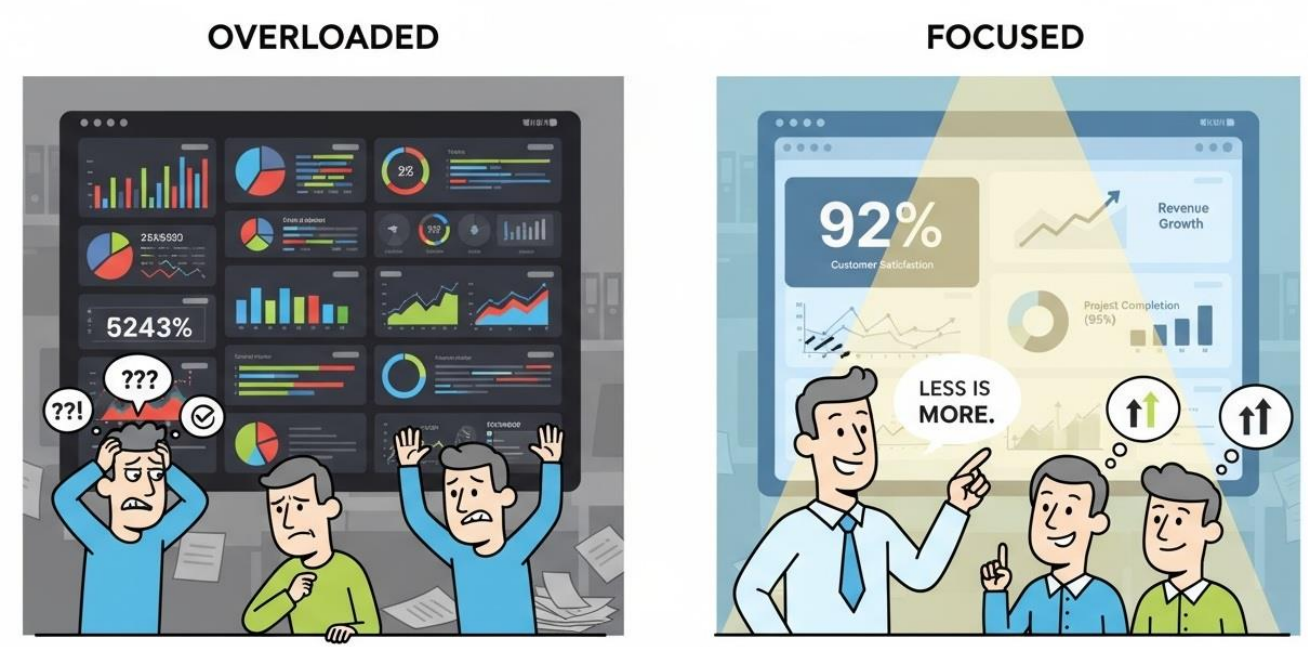
- KPI는 결과를 보여주는 후행지표(Lagging KPI)와 성과를 촉진하는 선행지표(Leading KPI)를 함께 고려해야 한다.
- 후행지표는 성과 확인용이라면, 선행지표는 개선 행동을 이끌어내는 역할을 한다.

좋은 KPI: 측정 가능성과 데이터 수집 용이성 확인



- KPI는 반드시 지속적으로 측정 가능해야 하며, 데이터 수집·갱신 비용이 과도해서는 안 된다.
- 측정이 불가능하거나 일관된 데이터 확보가 어려운 지표는 KPI에서 제외한다.
- 자동 수집·자동 업데이트가 가능한 지표를 우선 고려하면 운영 효율성이 높아진다.

좋은 KPI: KPI 개수 최소화



- KPI는 적을수록 명확해진다.
- 일반적으로 조직 단위 KPI는 3~5개, 팀 단위 KPI는 5~7개 이내가 적정하다.
- 지표가 많아질수록 초점이 흐려지고 구성원들이 무엇을 중요하게 봐야 하는지 혼란이 생긴다.

언어별 시각화 도구

데이터 시각화 도구

■ 데이터 시각화 도구

- 복잡한 데이터를 차트, 그래프, 지도, 대시보드 등의 시각적 형태로 표현하여 한눈에 이해할 수 있도록 돕는 소프트웨어 또는 라이브러리
  - 데이터를 숫자, 표 형태 그대로 전달하는 대신,
  - 시각적으로 구조화해 정보의 흐름과 패턴, 인사이트를 빠르게 파악할 수 있도록 지원
- 데이터 분석 과정 전반에서 의사결정을 지원하는 핵심 도구로 활용
- 목적에 따라
  - 통계 그래프,
  - 인터랙티브 차트,
  - BI 대시보드,
  - 지도 기반 시각화 등 다양한 형태의 시각 표현을 제공

언어별 대표적인 시각화 라이브러리/프레임워크

언어	대표 도구	주요 활용	제약/비고
Python	Matplotlib / Seaborn	통계 차트, 과학적 시각화, 이미지 저장	기본은 정적 출력, 복잡한 인터랙션 한계
R	ggplot2 / Shiny	통계 분석, 리포트, 대화형 웹앱	Shiny는 서버 구성 필요, 성능 튜닝 요구
JavaScript	D3.js / ECharts	맞춤형 웹 시각화, 대화형 대시보드	초기 학습 부담 큼, DOM 성능 고려 필요
Java	JFreeChart	엔터프라이즈 리포트, 데스크톱 앱	디자인 커스터마이즈 제약, 웹 통합 추가 작업
C# (.NET)	Power BI	BI 대시보드, 데스크톱/모바일 리포트	라이선스 비용 발생, 복잡 커스터마이징 시 추가 개발 필요

대시보드 도구

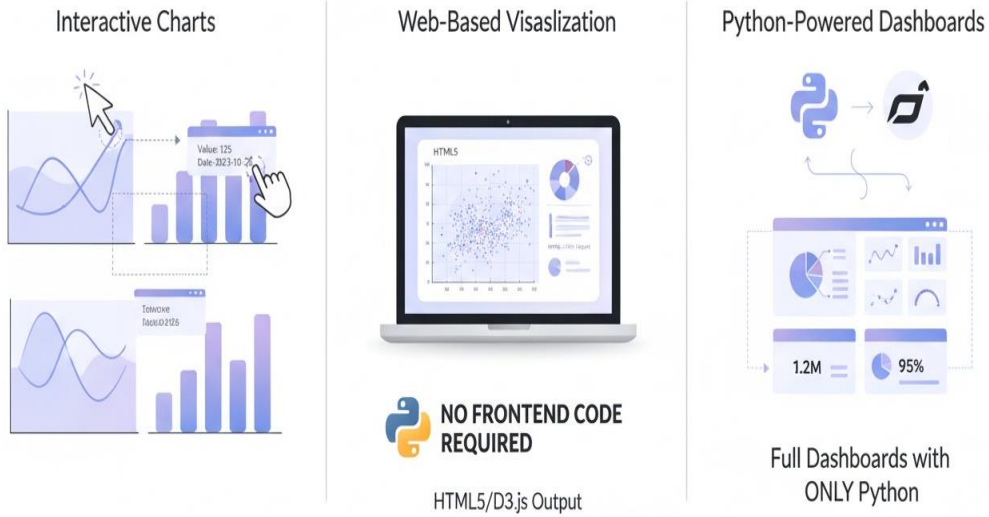
언어	대표 대시보드 도구	주요 활용	제약/비고
Python	Dash, Streamlit	데이터 분석·AI 모델 결과를 공유하는 웹 대시보드 개발	커스텀 UI 한계(특히 Streamlit), 복잡 서비스 확장 시 추가 개발 필요
R	Shiny	통계 분석 리포트, 연구용 인터랙티브 앱	서버 구성 필요, 사용자 증가 시 성능 튜닝 필수
JavaScript	Dash for JS, Looker, Apache Superset	프로덕션급 웹 대시보드, 맞춤형 UI/UX, BI 환경 구축	JS 기술 스택 학습 필요, 유지보수 비용 증가
Java	Tableau 연동, JSP 기반 Custom Dashboard	엔터프라이즈 데이터 리포트 및 BI 시스템 연계	직접 대시보드 구성보다 외부 BI 솔루션과 연계가 일반적
C# (.NET)	Power BI	기업 BI 대시보드, 실시간 매출/운영 모니터링	라이선스 비용 발생, 고도 커스터마이징 시 추가 개발 필요
No-Code /Low-Code	Looker Studio, Tableau	빠른 리포트 제작, 마케팅/운영 지표 공유, 비개발자 친화적	복잡한 로직 및 커스텀 기능 구현에 한계

Plotly 소개

Plotly?

Plotly

- 대화형(Interactive) 데이터 시각화 라이브러리



- 웹 기반 차트를 손쉽게 만들 수 있도록 지원하는 파이썬 도구
- 정적인 matplotlib 기반 그래프와 달리,
  - 줌 확대, 마우스 오버 툴팁, 범례 제어, 필터링 등 사용자 상호작용 기능을 기본 제공

공식 문서

- [Plotly Python Graphing Library]

Plotly의 특징

■ 대화형 시각화 지원

- 마우스로 차트를 클릭, 이동, 확대, 축소하고 세부 데이터를 확인 가능

■ 웹 기술 기반

- HTML, CSS, JavaScript 코드를 작성하지 않아도 Python 코드만으로 웹용 인터랙티브 시각화를 제작

■ Dash 프레임워크와 강력한 연계

- Plotly와 Dash를 함께 사용 → 프론트 기술 없이도 웹 대시보드 및 데이터 분석 서비스를 Python만으로 개발 가능

■ 다양한 차트 유형 제공

- 기본 차트: Line, Bar, Scatter, Pie
- 고급 차트: 지도(Choropleth), 3D Surface, Treemap, Sankey, Funnel, Network Graph 등

■ 연구·산업·교육 전반에서 활용

- 데이터 분석, 비즈니스 인텔리전스(BI), AI 서비스 개발, 연구 데이터 시각화, 교육용 시각화 등 영역에서 활발히 사용

Plotly 장단점 요약

장점	단점
• 대화형 시각화 기본 지원, • 마우스 오버·줌·패닝 탐색 용이	• 대규모 데이터·복잡 상호작용 시 • 렌더링 지연
• Python 기반 웹 차트 구현, • 보고서·대시보드·발표 활용	• 고급 커스터마이징·콜백 설계 시 • 학습 곡선 존재
• 기본 차트부터 3D·트리맵·지도 등 • 고급 시각화 제공	• 라이브러리 용량 큼, • 브라우저·모바일 로드 부담
• Dash 연동으로 웹 대시보드 신속 구축	• 고급 기능 일부 유료, • 엔터프라이즈 비용 발생

Dash 소개

Dash?

Dash?

- Python 기반의 웹 대시보드 프레임워크
- 대화형 웹 애플리케이션을 손쉽게 개발

- Python 코드만으로 웹 서비스 형태의 데이터 분석 대시보드를 제작할 수 있다는 것이 큰 장점
  - HTML, CSS, JavaScript와 같은 프론트 기술을 몰라도 됨.
  - Plotly 그래프를 중심으로 다양한 UI 컴포넌트(입력창, 버튼, 슬라이더, 드롭다운 등)를 조합
  - 상호작용형(Interactive) 데이터 분석 환경을 구축하는 데 특화

Dash의 특징

■ 프론트엔드 기술 없이 웹앱 개발 가능

- Python 코드만으로 웹 UI 구성, 그래프 출력, 사용자 입력 처리, 반응형 업데이트 등을 구현

■ 상태 변화에 따른 인터랙티브 동작 지원

- 콜백 함수(Callback)를 활용하여 화면 요소가 자동으로 갱신

■ Plotly 그래프와 자연스러운 통합

- Dash는 Plotly와 동일한 개발사가 제공
- Plotly 차트를 대시보드에 삽입하고 상호작용 기능과 연동하기에 최적화

■ 데이터 기반 의사결정 환경 구축에 최적화

- 실시간 데이터 모니터링, BI 대시보드, AI/ML 모델 시각화, 실험 결과 분석용 웹앱 등에 적용

Dash 장단점 요약

장점	단점
Python만으로 웹 대시보드 구축 가능, 프론트엔드 불필요	복잡한 UI/UX 구성 시 코드 길어질 수 있음
콜백 기반 인터랙션 구현으로 사용자 입력 반응형 웹앱 제작 용이	대규모 사용자 트래픽 대응 시 서버 성능 및 확장 고려 필요
Plotly 연동 최적화, 분석·시각화·서비스 통합 개발 가능	컴포넌트 디자인 제한적, 고급 커스텀 시 CSS/JS 필요
Flask 기반 구조로 배포, Docker, 클라우드 연계 용이	Streamlit 대비 초기 학습 진입 장벽 다소 존재

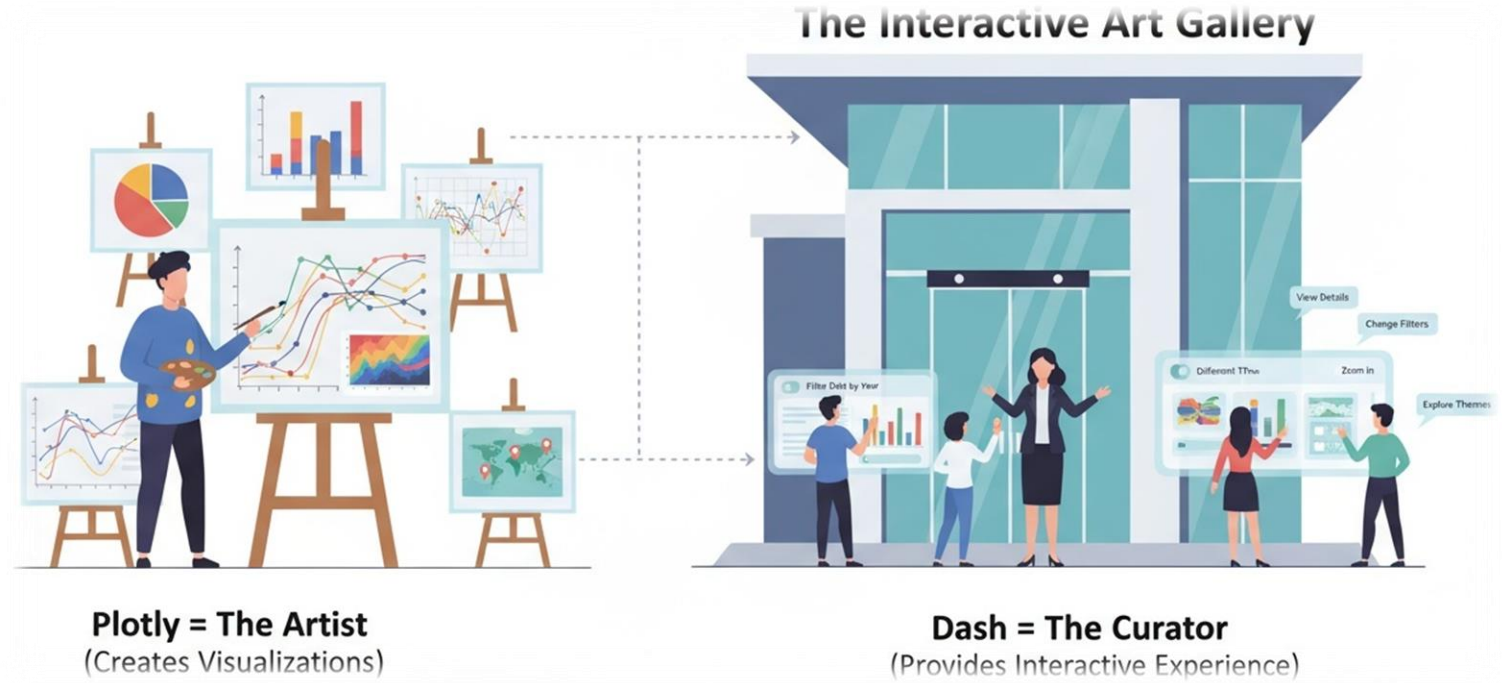
Plotly와 Dash 연동

시각화와 대시보드 개발을 하나의 파이썬 생태계에서 완성

■ 두 도구는 동일한 개발사(Plotly社)에서 제공하며 자연스럽게 연계

- Plotly와 Dash 연동은 데이터 분석 파이프라인 구축에 사용

시각화 → **공유** → **대시보드화** → **서비스화**로 확장하는 과정이 매우 효율적



임무와 역할

Plotly → 그래프 그리는 도구 (Visualization Library)

- Python, R, JavaScript 모두 지원
- 인터랙티브 차트: zoom, pan, hover, tooltip
- Jupyter Notebook이나 HTML에서 바로 사용 가능

Dash → 웹앱·대시보드 만드는 프레임워크 (Framework)

- Dash는 웹페이지의 UI 구조(레이아웃, 버튼, 입력창 등)를 만든다.
- Dash는 Plotly 그래프를 웹 UI에 넣고
- 버튼, 드롭다운, 슬라이더 같은 UI와 연결
- Dash는 사용자 입력에 따라 Plotly 차트가 자동 업데이트되도록 연동
- Dash는 웹 서버 + 콜백 시스템 제공

대시보드 구현을 위한 다양한 선택

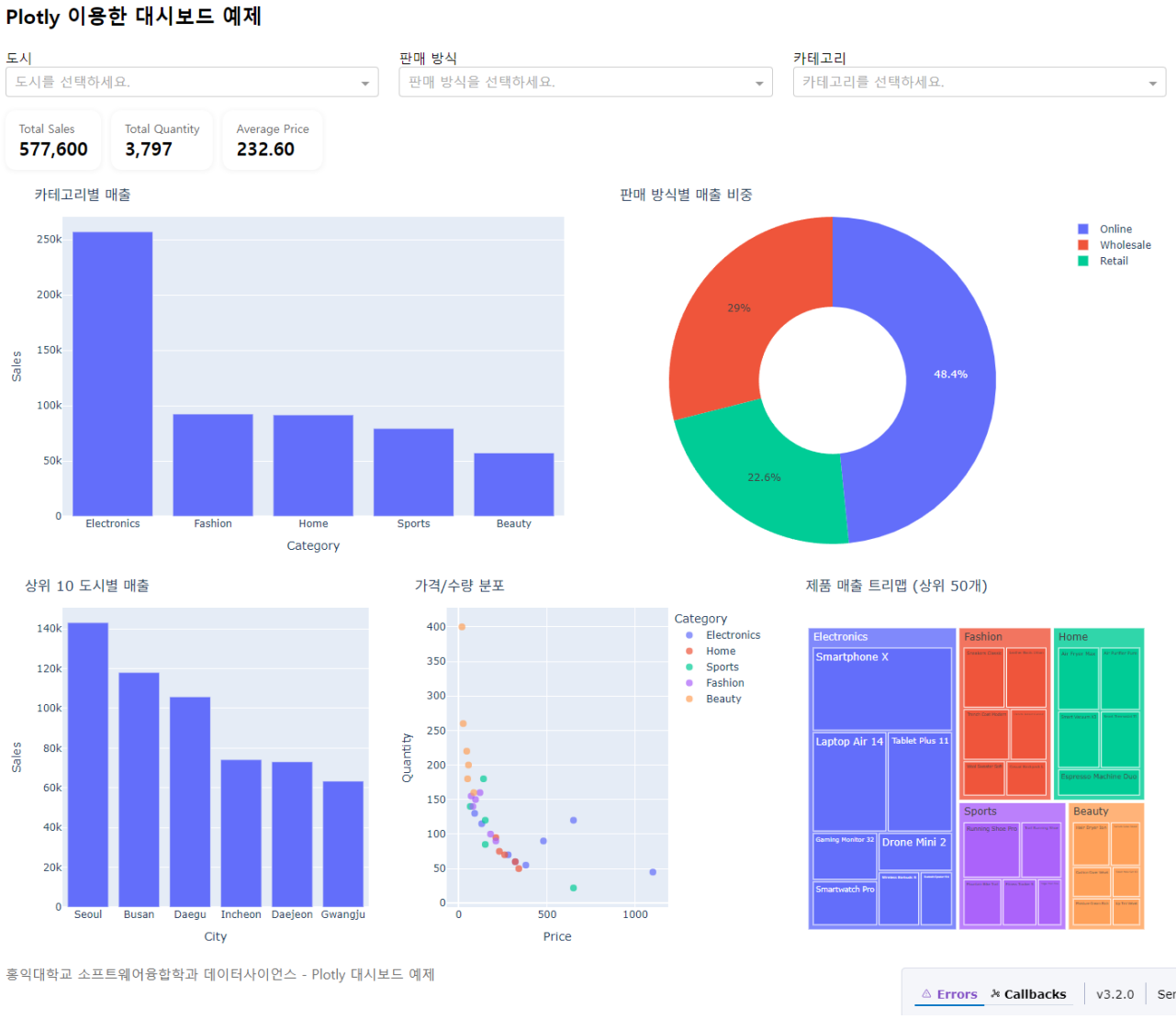
고려사항 (데이터 분석 → 시각화 → 서비스 파이프라인 구축)

개발 목적, 난이도, 배포 방식, 확장성, 협업 환경에 따라 다양한 접근 방식이 존재!

선택 기준	추천 기술 스택	특징
빠른 프로토타입 개발이 필요한 상황	Streamlit, Shiny	설치 즉시 실행 가능, 코드 몇 줄로 데모/시제품 제작에 최적
교육 · 연구, AI 모델 시각화	Plotly + Dash	Python만으로 대화형 시각화 및 대시보드 제작, 실습/강의용에 적합
엔터프라이즈 확장성 · 운영성	Power BI, React + D3.js / ECharts, Spring Boot + Frontend	보안·권한·배포·대규모 사용자 운영에 최적화
정교한 커스텀 UI/UX	JavaScript 생태계 (React + D3.js/ECharts)	자유도와 표현력이 가장 뛰어난 맞춤형 대시보드 구축 가능
리포트 중심, 문서화 기반	Power BI, R Markdown, JasperReports	정형 리포트, 자동 보고서 생성, 문서 기반 분석 공유에 강점

대시보드 구현

목표 UI


Plotly 이용한 대시보드 예제
도시
도시를 선택하세요.
판매 방식
판매 방식을 선택하세요.
카테고리
카테고리를 선택하세요.
Total Sales
577,600
Total Quantity
3,797
Average Price
232.60
카테고리별 매출
Sales
250k
200k
150k
100k
50k
0
Electronics
Fashion
Home
Sports
Beauty
Category
판매 방식별 매출 비중
Online
Wholesale
Retail
29%
48.4%
22.6%
상위 10 도시별 매출
Sales
140k
120k
100k
80k
60k
40k
20k
0
Seoul
Busan
Daegu
Incheon
Daejeon
Gwangju
City
가격/수량 분포
Category
Electronics
Home
Sports
Fashion
Beauty
Quantity
400
350
300
250
200
150
100
50
0
0
500
1000
Price
제품 매출 트리맵 (상위 50개)
Electronics
Smartphone X
Laptop Air 14
Tablet Plus 11
Gaming Monitor 32
Drone Mini 2
Smartwatch Pro
Fashion
Home
Espresso Machine Duo
Sports
Running Shoe Pro
Beauty
홍익대학교 소프트웨어융합학과 데이터사이언스 - Plotly 대시보드 예제
Errors
Callbacks
v3.2.0
Server

실습 환경

- **Dataset: 바로 다운로드 (click me)**
- **필요한 패키지 설치**

```
pip install dash plotly pandas
```

- **로컬 서버-클라이언트 구조 (실무환경)**
  - 사용자 (users, clients)
  - 웹서버(nginx, apache)
  - 미들웨어:
    - wsgi (gunicorn, uwsgi, ...), asgi (uvicorn, fastapi, ...)
  - 애플리케이션 (개발 언어에 따라 선택)
    - Python Apps: Flask, Django, FastAPI,
    - Java Apps: Spring Boot, Jakarta EE, Micronaut,
    - Node.js Apps: Express, NestJS, Fastify,
    - C++ Apps: Drogon, Pistache, cpprestsdk,

우리의 실습 환경

- Nginx, 리버스 프록시, 배포형 구조는 제외, 단순하게 로컬 PC에서 바로 실행되는 서버-클라이언트 구조 사용
- 실제로는 Python App (Flask)에서 제공하는 개발용 웹서버 (dev-webserver) 사용하는 것임
- Dash는 Plotly + Flask + React를 묶어 만든 웹앱 프레임워크

웹 브라우저로 접속
(아래 주소 중 하나로 접속)

- http://127.0.0.1:8050
- http://localhost:8050

권장 폴더 구조

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

app.py

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

■ Dash 엔트리 포인트 모듈

- 애플리케이션 초기화 및 인스턴스 생성 역할
- 분리된 기능 모듈을 하나로 연결하는 허브 역할
- 실행 진입점(Entry Point) 제공

  `if __name__ == "__main__":` 블록을 통해

  로컬 개발 환경에서 서버를 실행

■ 최종 통합 지점

- 애플리케이션의 시작, 조립, 실행을 담당하는 핵심 모듈
- 여러 파일로 기능을 분리하여 구조를 유지하더라도,
- 그 모든 요소를 결합하여 웹 서비스로 구동하는 포인트

config.py

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

전역 설정 모듈

- `config.py`는 애플리케이션 전반에서 공통으로 사용되는 설정 정보를 한 곳에 모아 관리
- 데이터 경로, 서버 실행 환경, 공통 스타일과 같은 설정 값이 여러 파일에 중복 작성되면 유지보수가 어려워 짐
- **주요 책임 (Responsibilities)**
  - 설정 값 관리의 일원화
  - 환경 변화에 대한 유연한 대응
  - 코드 가독성 및 역할 분리 개선
  - 스타일 재사용과 UI 일관성 확보

data_loader.py

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│   ├── data_loader.py
│   ├── figures.py
│   ├── layout.py
│   └── callbacks.py
└── files/
    └── sales_data.csv
```

데이터 로딩 및 전처리 모듈

- `data_loader.py`는 대시보드에서 사용할 원본 데이터를 불러오고, 시각화에 적합한 형태로 정규화하는 역할 수행
- 다양한 소스와 형식의 데이터를 불러올 수 있어야 함.
- 결측값, 잘못된 자료구조를 Cleaning 할 수 있어야 함.
- **주요 책임 (Responsibilities)**
  - 데이터 정규화의 중앙 관리
  - 데이터 품질 확보 및 오류 예방
  - 파생 지표 생성으로 분석 가치 향상
  - 데이터 스키마 변경에 대한 유지보수 용이성 보증

figures.py

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│   ├── data_loader.py
│   ├── figures.py
│   ├── layout.py
│   └── callbacks.py
└── files/
    └── sales_data.csv
```

Plotly 시각화 모듈

- `figures.py`는 데이터프레임을 입력받아 Plotly Figure 객체를 생성하는 역할을 전담하는 모듈
- 시각화 코드를 별도의 모듈로 분리하면 다른 로직과 혼합되지 않기 때문에 코드의 책임이 명확해지고 유지보수가 쉬워진다.
- **주요 책임 (Responsibilities)**

  5개 함수: 각 함수는 "하나의 차트" 생성만 담당

  - **막대 차트(Bar Chart)** 시각화
  - **도넛 차트(Donut/Pie Chart)** 시각화
  - **Top-N 막대 차트** 시각화
  - **산점도(Scatter Plot)** 시각화
  - **트리맵(Treemap)** 시각화

layout.py (1/2)

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

■ 데이터 로딩 및 전처리 모듈

- `layout.py`는 대시보드 화면을 구성하는

  UI 골격을 정의하는 모듈

- 데이터 로직이나 콜백 계산 없이,

  무엇을 어디에 배치할지만을 선언적으로 표현

- 레이아웃과 로직을 분리하면 협업, 테스트, 확장이 쉬워짐

- **주요 책임 (Responsibilities)**

  - 관심사의 분리(Separation of Concerns)
  - 반응형(Responsive) 레이아웃과 일관된 스타일
  - 재사용 가능한 UI 패턴 캡슐화
  - 확장성과 모듈화

layout.py (2/2)

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

■ 역할 매핑

- **필터 행(Filters Row)**

  City, Channel, Category 드롭다운을 좌→우로 배치

- **KPI 행(KPI Row)**

  html.Div(id="kpi-row")만 미리 배치하고 내용은 콜백으로 주입

- **상단 차트 행(Top Charts Row)**

  카테고리 매출 바 차트, 채널 비중 도넛

  → 두 개의 핵심 요약 차트를 나란히 배치

- **하단 차트 행(Bottom Charts Row):**

  g-city(도시 Top-N), g-scatter(가격-수량 산점도),

  g-treemap(제품 트리맵)을 배치

- **푸터(Footer):** 출처와 맥락을 명시

callbacks.py (1/3)

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│       ├── data_loader.py
│       ├── figures.py
│       ├── layout.py
│       └── callbacks.py
└── files/
        └── sales_data.csv
```

데이터 로딩 및 전처리 모듈

- `callbacks.py`는 사용자 입력(필터 변경) 받아 데이터를 갱신
- KPI와 모든 차트를 업데이트하는 대시보드의 동적 제어 센터
- 레이아웃이 화면의 뼈대를 정의한다면, 콜백은 사용자 상호작용에 반응하는 데이터 흐름과 출력을 담당
- 입력 → 순수 계산 → 출력의 흐름을 일관되게 유지
  - 테스트 용이, 디버깅 파이프라인 단순화 가능

콜백 함수란?

- "나중에 특정 시점에 호출되도록 다른 함수에 넘겨주는 함수"
- 개발자가 직접 부르는 함수가 아님!
- 시스템/프레임워크/이벤트가 호출하는 함수

callbacks.py (2/3)

```
your-workspace/
├── app.py
├── requirements.txt
├── config.py
├── services/
│   ├── data_loader.py
│   ├── figures.py
│   ├── layout.py
│   └── callbacks.py
└── files/
    └── sales_data.csv
```

데이터 로딩 및 전처리 모듈

특징	설명
지연 실행 Deferred Execution	콜백은 즉시 실행되지 않고, 특정 조건이나 이벤트가 발생했을 때 나중에 실행된다.
함수 전달 Function as Argument	함수 자체를 다른 함수의 인자로 전달하여, 실행 흐름을 위임할 수 있다.
이벤트 기반 처리 Event-Driven	버튼 클릭, 데이터 수신, 작업 완료 등 이벤트가 발생할 때 자동 호출되는 방식과 궁합이 좋다.
제어 흐름 역전 Inversion of Control	언제 실행할지 개발자가 직접 결정하지 않고, 시스템 또는 프레임워크에 제어권을 넘긴다.

callbacks.py (3/3)

일반 콜백 vs Dash 콜백 비교

관점	콜백 의미	누가 호출하는가?	주요 목적
일반 콜백	나중에 실행될 함수를 등록해두고, 실행 시점을 위임하는 함수	시스템, 라이브러리, 이벤트 핸들러	코드 재사용, 비동기 처리, 이벤트 처리
Plotly+Dash 콜백	UI 컴포넌트(Input) 변경 시 자동 실행되어, 데이터 계산 후 화면 Output을 갱신하는 함수. 사용자가 분석할 데이터를 변경한 경우에 적용됨 (예: 옵션을 매출액 → 지역으로 항목을 바꾼 경우 등)	Dash 프레임워크	대화형 시각화, 반응형 대시보드

콜백 흐름

- **단일 진입점 콜백 register_callbacks():** 앱 내부에 `update_dashboard()` 콜백을 등록
- `update_dashboard()`: **대시보드 전체 갱신**

  입력값 수신 → apply_filters()로 데이터 준비 → compute_kpis()로 KPI 산출

  → KPI 카드를 보기 좋은 문자열 포맷으로 구성 → 각 fig_* 함수로 그래프 생성

  → KPI 컴포넌트와 5개 Figure를 정해진 출력 순서대로 반환

서버 실행 및 접속

```
# 서버 실행

  python app.py


# 브라우저 접속

  http://127.0.0.1:8050

  http://localhost:8050
```

Homeworks

https://www.deepshark.org/courses/data_science/w/12_advanced_visualization#homeworks

수고하셨습니다 ..^^..