AI-FLOW

DATA ANALYSIS · 인천공항 T1 출국장 혼잡도 분석

이 문서는 공모전 제안서의 모든 정량 수치가 어떻게 도출되었는지를, 실제 분석 코드와 함께 단계별로 보여준다. 코드를 몰라도 읽을 수 있도록 각 단계마다 '무엇을 왜 했는지'를 먼저 설명하고, 그 아래에 코드와 결과 그래프를 둔다. 코드 부분(어두운 박스)은 건너뛰고 설명과 그래프만 따라가도 분석의 흐름과 결론이 그대로 전달된다. 전체 코드는 문서 끝 부록에 수록했다.

원자료	인천공항 공식 예상혼잡도 페이지 (2026.03.22 일요일, T1)
분석 도구	Python · pandas · numpy · matplotlib
분석 범위	출국장 6개 × 24시간 → 분산 가능한 4개 출국장 추출 후 분석
핵심 결론	4번 출국장 40% 집중 / 균등분배 시 피크 26.3% 감소(이론적 상한)

KEY FINDINGS

이 분석이 찾아낸 것

40.0% 4번 출국장 한 곳이 T1 전체 여객의 40%를 차지

9.3배

가장 붐비는 4번 vs 가장 한가한 5·6번의 격차

83%

하루 24개 시간대 중 20개에서 4번이 최대 혼잡 — 우연이 아닌 구조

26.3%

균등 분배 시 피크 4번 부하 감소폭 (이론적 상한)

약 8~10%

현실적인 앱 이용률 30%만 가정해도 나오는 감소폭

STEP 1

원자료를 있는 그대로 적재한다

무엇을 하나

공항 예상혼잡도 페이지의 표를 그대로 코드에 옮긴다. 출국장은 1~6번 전부 — 나중에 분석에서 뺄 게이트라도 일단 원본 형태 그대로 들고 시작한다.

1번은 교통약자 전용이라 일반 여객 수치가 0으로 잡히고, 5·6번은 페이지가 둘을 묶어 공개하므로 한 칸으로 둔다.

PYTHON

```
ALL_GATES = ["1번", "2번", "3번", "4번", "5·6번"]
raw = pd.DataFrame(raw_rows,
    columns=["시간대"] + ALL_GATES).set_index("시간대")
# 예: ("08~09", 0, 1157, 1116, 1268, 199)  <- 피크 시간대 한 줄
#      1번=0(교통약자)  2번  3번  4번=최대  5·6번
```

STEP 2

분석 대상만 추출·가공한다 — 데이터 분석의 첫 단계

무엇을 왜 하나

원본 6개 게이트 중 무엇을 분석할지 고르는 단계다. 여기서 내린 판단이 분석 전체의 전제가 된다.

(a) 1번 게이트 제외 — 교통약자 전용이라 일반 여객을 흘려보낼 수 없다. '분산 가능한 출국장'에서 뺀다.

> **중요한 원칙:** 모르는 수치를 채워 넣지 않는다. 1번의 실제 여객 수는 공개되지 않으므로, 값을 지어내는 대신 '제외했다'는 사실만 남긴다.

(b) 운영 안 하는 시간대 제거 — 닫힌 게이트가 평균을 왜곡하지 않도록 정리한다.

PYTHON

```
ACTIVE_GATES = ["2번", "3번", "4번", "5·6번"]   # (a) 1번 제외
df = raw[ACTIVE_GATES].copy()
operating = df[(df.sum(axis=1) > 0)]           # (b) 운영시간만
# 옮겨 적기 검산 — 한 칸 틀리면 뒤가 다 어긋나니 합계부터 맞춘다
EXPECT = {"2번":10577, "3번":14948, "4번":18297, "5·6번":1970}
# → 4개 게이트 합계 45,792명, 공항 페이지 하단값과 전부 일치(OK)
```

→ 이 추출·정제를 거쳐야 비로소 '분산을 논할 수 있는' 분석표가 된다.

STEP 3

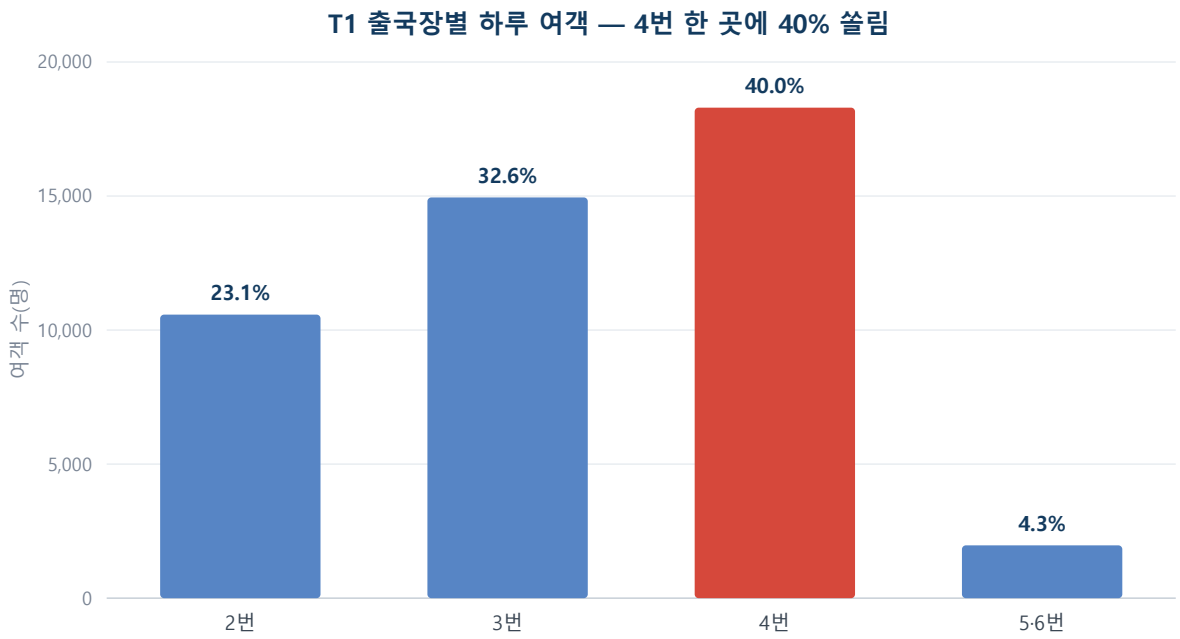
얼마나 쏠려 있나 — 집중률과 격차

무엇을 하나

출국장별 하루 총 여객을 내고 비율을 본다. 예상은 했지만 4번 하나가 40%를 차지하는 건 직접 계산해보니 다소 충격적인 수치였다.

PYTHON

```python
daily = df.sum()                                  # 출국장별 하루 총 여객
share = (daily / daily.sum() * 100).round(1)  # 집중률(%)
# 결과: 2번 23.1% / 3번 32.6% / 4번 40.0% / 5·6번 4.3%
gap = daily.max() / daily.min()                   # 9.3배
```



그래프 1 — 출국장별 하루 여객. 4번(빨강) 한 곳에 40%가 몰리는 반면 5·6번은 4.3%에 그친다.

STEP 4

균등 분배하면 피크가 얼마나 빠지나 (이론적 상한)

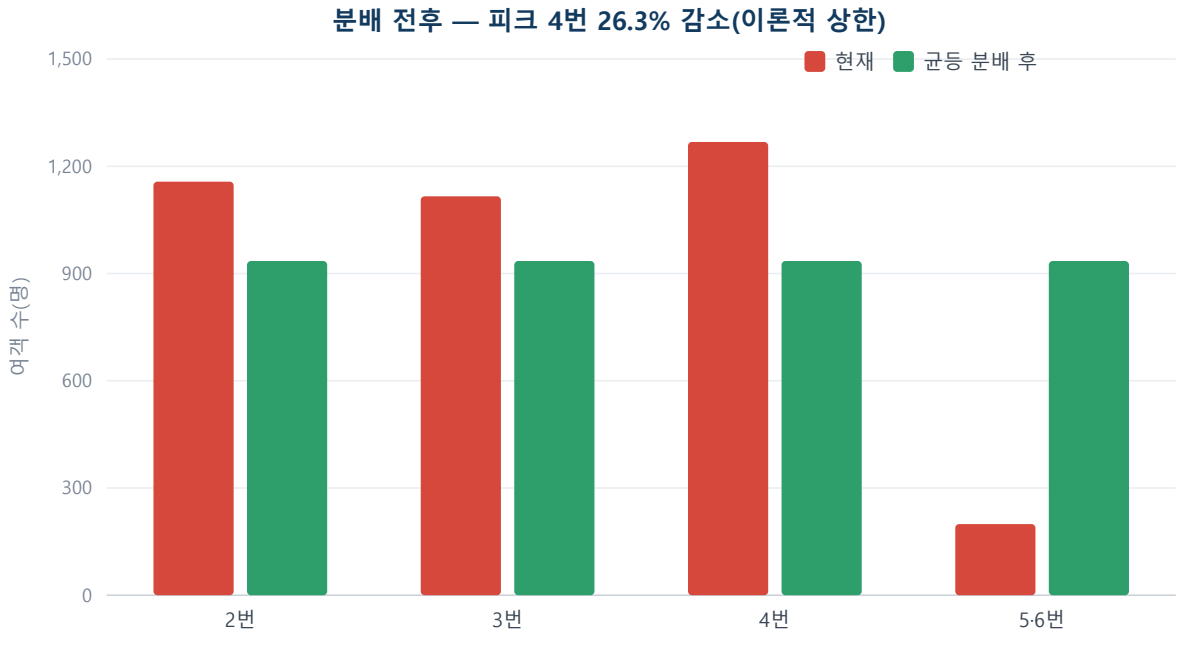
무엇을 하나

가장 붐비는 피크 시간대(08~09시)만 떼서, 모든 출국장에 똑같이 나누면 4번 부하가 얼마나 줄어드는지 계산한다.

'균등 분배'는 현실에는 없는 가정이므로 반드시 '이론적 상한'이라고 못박는다. 그래야 과장이 되지 않는다. 현실적 추정치는 다음 단계에서 따로 제시한다.

PYTHON

```
peak = df.loc["08~09"]                    # 피크 한 줄
even = peak.sum() / len(ACTIVE_GATES)     # 균등 분배 시 한 곳당 = 935명
worst = peak.max()                        # 현재 최대(4번) = 1,268명
reduction = (worst - even) / worst * 100  # = 26.3%
```



그래프 2 — 분배 전후 비교. 피크 4번의 1,268명(빨강)이 균등 분배 시 935명(초록)으로, 26.3% 감소한다.

STEP 5

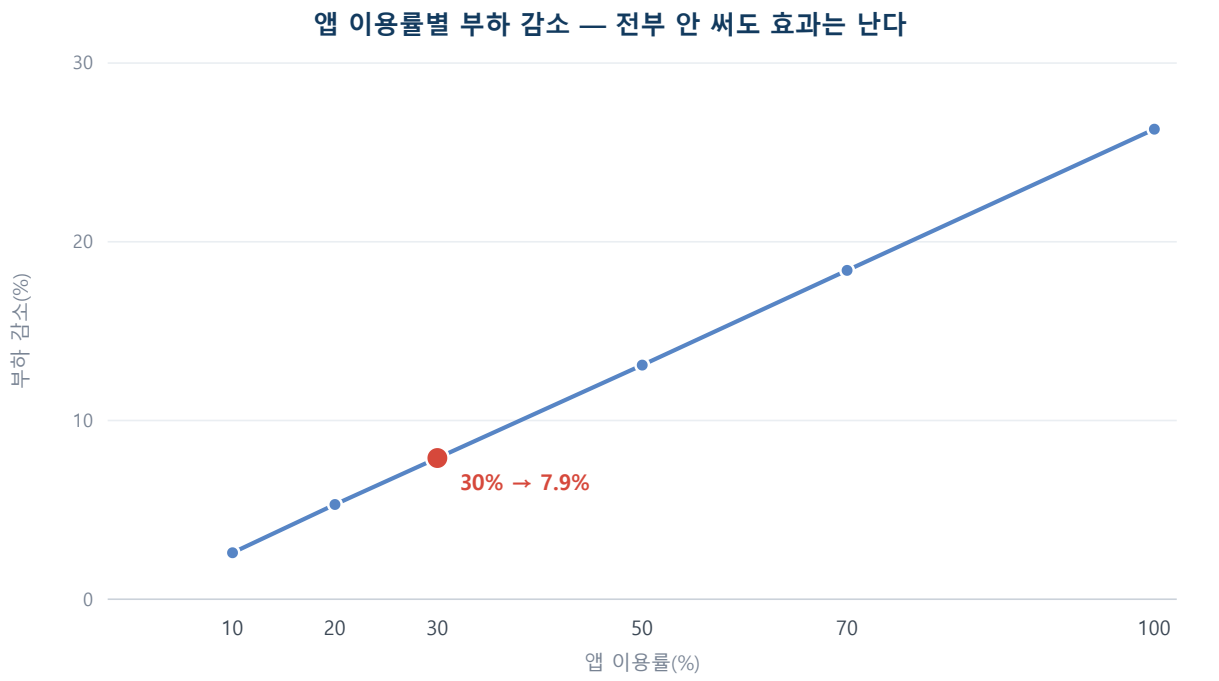
현실적인 이용률에서는 — 보수 추정

무엇을 하나

앱을 모든 여객이 쓸 리는 없다. 그래서 '30%만 써도 효과가 있나?'를 본다. 모델은 단순하다 — 효과 = 이론적 상한 × 이용률 (앱을 쓴 사람만 분산에 기여).

이용률 30%일 때 약 8~10%. 이 숫자가 오히려 제안서에서 가장 설득력 있었다. '모두가 쓰지 않아도 효과가 난다'를 보여주기 때문이다.

```python
rates = [0.1, 0.2, 0.3, 0.5, 0.7, 1.0]
sens = {r: reduction * r for r in rates}
# 이용률 30% -> 7.9% (제안서: 약 8~10%)
# 체감 환산: 피크 4번 1,268명 중 약 100명 분산 = 검색대 1개 라인 추가 효과
```



그래프 3 — 앱 이용률별 부하 감소 곡선. 이용률 30%(빨간 점)에서 약 8%, 이용률이 높아질수록 이론적 상한 26.3%에 가까워진다.

STEP 6

원자료를 다시 해석한다 — 파생 지표

무엇을 왜 하나

단순 집계를 넘어, 원자료를 '분산 설계'라는 용도에 맞게 다시 해석해 뽑은 지표들이다. 제안서의 주장을 데이터로 뒷받침하는 근거가 된다.

6-1 병목 자동 탐지

4번이 24개 시간대 중 20개(83%)에서 최대 혼잡 → 쏠림은 특정 시간의 우연이 아니라 구조

6-2 상시 vs 피크

4번 집중도가 한산한 때(34.0%)나 피크(33.9%)나 거의 동일 → 피크가 아니라 '구조'가 원인

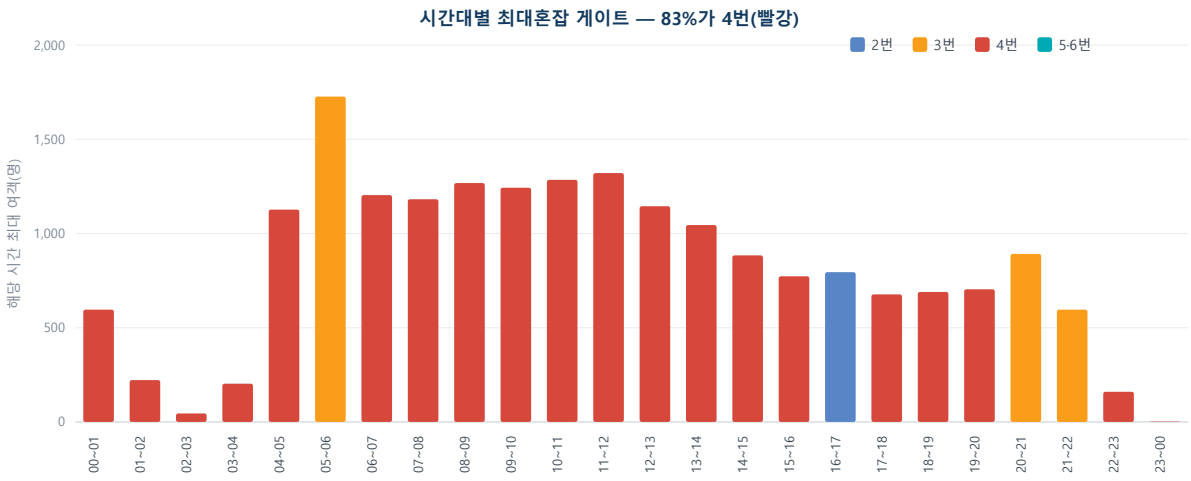
6-3 유휴 용량

5·6번을 흡수처로 보면, 평균 수준까지만 채워도 약 9,500명 추가 수용 여력

6-4 불균형 지표

게이트 간 변동계수 현재 0.62 → 균등분배 시 0.00 (분배가 줄여야 할 불균형의 크기)

특히 6-2는 처음 예상과 달랐다. '피크에 쏠림이 심해질 것'이라 봤는데 데이터는 시간대와 무관하게 일정하다고 말했다. 예상이 빗나갔지만, 이것이 오히려 '상시적·구조적 쏠림'이라는 더 강한 발견이 되었다. 데이터가 말하는 대로 해석을 바꾼 것이다.



그래프 4 — 시간대별 최대 혼잡 게이트. 거의 모든 막대가 4번(빨강)으로, 4번이 하루 종일 병목이라는 사실이 한눈에 드러난다.

LIMITS

분석의 한계

숫자를 신뢰하게 만드는 것은 숫자의 크기가 아니라, 그 숫자가 어디까지만 말할 수 있는지를 밝히는 일이다. 이 분석의 경계는 세 가지다.

01 단일 날짜 데이터

2026년 3월 22일(일) 하루치 예고 데이터가 기반이다. 요일·계절에 따른 변동은 담지 못한다. 다만 일요일은 가족 단위 여객이 많아 쏠림이 가장 선명하게 드러나는 날로 의도적으로 선택했고, '4번 쏠림과 유휴 용량이 공존한다'는 핵심 구조는 날짜가 바뀌어도 흔들리지 않는다.

02 미반영 변수

항공사별 출국장 배정 규칙, 체크인 카운터 위치, 단체 여행 스케줄 등 여객이 실제로 출국장을 고르게 만드는 변수들은 반영하지 않았다. 26.3%를 실현치가 아니라 '이론적 상한'으로만 쓰고, 보수 추정(약 8~10%)을 병기한 이유다.

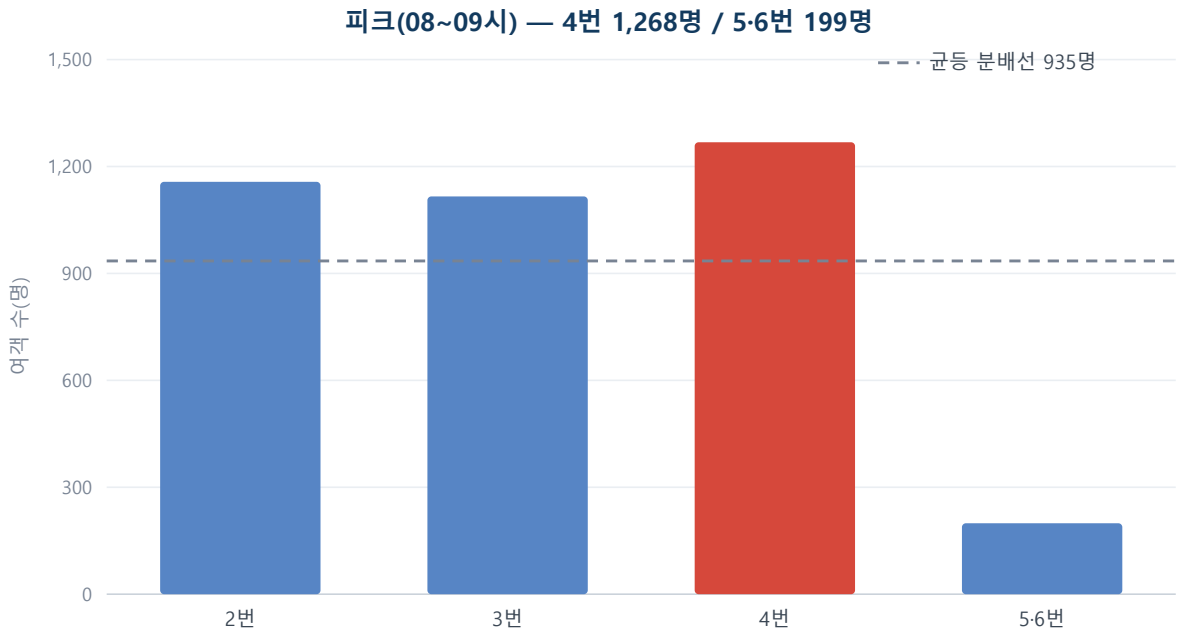
03 게이트 집계 기준의 차이

제안서는 공항 안내 기준을 따라 'T1 출국장 6개(교통약자 전용 2개 포함)'로 표기했다. 반면 예상혼잡도 데이터에는 1번만 별도로 잡히고, 5·6번은 일반 여객 수치로 묶여 공개된다. 이 분석은 데이터가 실제로 구분해 주는 기준을 따라 1번만 제외하고 5·6번은 통합된 일반 게이트로 다뤘다 — 문서 간 숫자가 다르게 보인다면 이 기준 차이 때문이다.

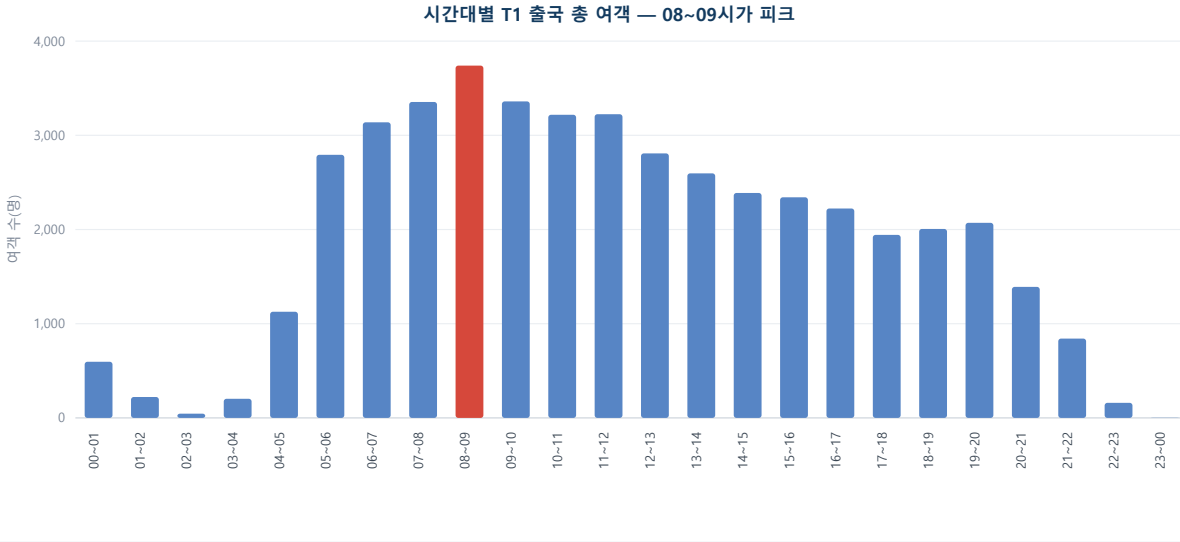
참고 — 그래프 종수: 공모전 당시 제작한 차트는 5종이며, 이 문서의 병목 그래프(그래프 4)는 분석을 코드로 재현하는 과정에서 추가한 것이다. 따라서 본 문서와 부록 코드는 6종을 담고 있다.

APPENDIX

참고 그래프



그래프 5 — 피크 시간대 출국장별 여객. 같은 시각, 4번은 1,268명이지만 5·6번은 199명이다.



그래프 6 — 시간대별 T1 출국 총 여객. 08~09시(빨강)가 하루 피크다.

APPENDIX · FULL CODE

전체 분석 코드

본문에서 발췌해 보여준 코드의 전문이다. 그대로 실행하면 위 모든 수치와 그래프 6종이 재현된다. (Python · pandas · numpy · matplotlib 필요)

aiflow_analysis.py — PYTHON

```
# -*- coding: utf-8 -*-
"""
AI-FLOW : 인천공항 T1 출국장 혼잡도 분석
=========================================
인천공항 AI-PORT 공모전 (2026.03) 제출작의 근거가 된 분석 전체.
원자료는 인천공항 공식 '예상혼잡도' 페이지에서 직접 가져왔다.
2026년 3월 22일(일) T1 기준. 일요일을 고른 건 가족 단위가 많아
쏠림이 제일 선명하게 나오기 때문.

이 스크립트의 흐름은 내가 실제로 분석한 순서 그대로다:
  STEP 1 원자료를 있는 그대로 적재 (출국장 6개 전부)
  STEP 2 분석 대상만 추출·가공 (왜 6개 중 4개만 보는가)
  STEP 3 기초 분석 — 집중률, 격차 [제안서 40%, 4.3%]
  STEP 4 피크 균등분배 효과 [제안서 1,268 / 935 / 26.3%]
  STEP 5 이용률별 보수 추정 + 체감 환산 [제안서 약 8~10%, 100명]
  STEP 6 파생 지표 — 병목·쏠림심화·유휴·불균형 (해석 레이어)
  STEP 7 시각화 6종

수치 끝의 [제안서] 표시는 그 값이 제출본 어디에 쓰였는지를 가리킨다.
데이터 분석의 핵심은 큰 원자료에서 필요한 부분을 뽑아 가공하고,
용도에 맞게 해석하는 것 — STEP 2와 STEP 6가 그 두 축이다.

실행: python aiflow_analysis.py (pandas, numpy, matplotlib 필요)
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rcParams

# 한글 안 깨지게. 환경마다 폰트 이름이 달라서 매번 이걸로 고생한다.
# 맥은 'AppleGothic', 윈도우는 'Malgun Gothic'. 둘 다 없으면 Noto/나눔으로 떨어진다.
for _f in ["AppleGothic", "Malgun Gothic", "NanumGothic", "Noto Sans CJK KR", "Noto
Sans CJK JP"]:
    if any(_f in f.name for f in font_manager.fontManager.ttflist):
        rcParams["font.family"] = _f
        break
rcParams["axes.unicode_minus"] = False
```

```
# ================================================================
# STEP 1. 원자료 적재
# 공항 예상혼잡도 페이지의 T1 출국장 표를 그대로 옮긴다.
# 게이트는 1~6번 전부 둔다. 실제 페이지에 6개가 다 있으니까
# 분석에서 뺄 거라도 일단 원본 형태 그대로 들고 시작한다.
#  - 1번: 교통약자 전용. 페이지에 일반 여객 수치가 따로 안 잡혀 0.
#  - 5,6번: 페이지가 둘을 묶어서 공개 -> 한 칸('5·6번')으로 둠.
# 표만 바꾸면 다른 날짜로도 똑같이 돌아간다.
# ================================================================
# 컬럼: 시간대, 1번, 2번, 3번, 4번, 5·6번
raw_rows = [
    ("00~01", 0, 0, 0, 596, 0),
    ("01~02", 0, 0, 0, 222, 0),
    ("02~03", 0, 0, 0, 45, 0),
    ("03~04", 0, 0, 0, 203, 0),
    ("04~05", 0, 0, 0, 1127, 0),
    ("05~06", 0, 0, 1727, 1066, 0),
    ("06~07", 0, 981, 870, 1204, 83),
    ("07~08", 0, 1007, 992, 1182, 173),
    ("08~09", 0, 1157, 1116, 1268, 199),  # 피크
    ("09~10", 0, 891, 1050, 1243, 176),
    ("10~11", 0, 705, 1029, 1285, 199),
    ("11~12", 0, 652, 1021, 1321, 230),
    ("12~13", 0, 572, 925, 1145, 166),
    ("13~14", 0, 580, 830, 1045, 141),
    ("14~15", 0, 626, 769, 884, 109),
    ("15~16", 0, 751, 708, 773, 110),
    ("16~17", 0, 795, 628, 710, 90),
    ("17~18", 0, 644, 560, 677, 62),
    ("18~19", 0, 607, 616, 690, 93),
    ("19~20", 0, 609, 619, 704, 139),
    ("20~21", 0, 0, 892, 499, 0),
    ("21~22", 0, 0, 596, 246, 0),
    ("22~23", 0, 0, 0, 160, 0),
    ("23~00", 0, 0, 0, 2, 0),
]
ALL_GATES = ["1번", "2번", "3번", "4번", "5·6번"]
raw = pd.DataFrame(raw_rows, columns=["시간대"] + ALL_GATES).set_index("시간대")
print(f"STEP1 원자료 적재: {raw.shape[0]}개 시간대 x {raw.shape[1]}개 게이트")

# ================================================================
# STEP 2. 분석 대상 추출·가공 <- 데이터 분석의 첫 번째 꽃
# 원본 6게이트 중 무엇을 분석할지 '고르는' 단계.
# 여기서 내린 판단이 분석 전체의 전제가 된다:
#
# (a) 1번 게이트 제외.
#     교통약자 전용이라 일반 여객 분산 대상이 아니다.
#     전체 여객을 여기로 흘려보내는 건 정책상 불가능하므로
#     '분산 가능한 출국장'에서 뺀다. (제안서 ⑤ '교통약자 전용 2개')
#     -> 값을 채워 넣지 않는다. 모르는 수치를 지어내면 분석이 아니라 창작이다.
```

```
#        '제외했다'는 사실만 남긴다.
#
# (b) 운영 안 하는 시간대(전 게이트 0) 제거.
#     심야엔 일부 게이트만 열어서, 닫힌 칸이 평균을 왜곡한다.
#     '운영 중'인 시간만 남겨야 혼잡도 비교가 공정하다.
#
# 이 두 가지를 거치면 비로소 '분산을 논할 수 있는' 분석표가 된다.
# ================================================================
ACTIVE_GATES = ["2번", "3번", "4번", "5·6번"]  # (a) 1번 제외
excluded = [g for g in ALL_GATES if g not in ACTIVE_GATES]
print(f"STEP2 (a) 분석 대상 게이트: {ACTIVE_GATES} / 제외: {excluded} (교통약자 전용)")
df = raw[ACTIVE_GATES].copy()
operating = df[(df.sum(axis=1) > 0)]  # (b) 운영시간만
removed_b = raw.shape[0] - operating.shape[0]
print(f"     (b) 분석 게이트가 모두 닫힌 시간대 제거: "
      f"{removed_b}개 (이 게이트 조합에선 24시간 내내 1곳 이상 운영)")
print(f"     * 1번을 포함했다면 심야가 걸러졌을 단계 — 게이트 선택이 전처리에 영향")
# 옮겨 적기 검산. 한 칸 틀리면 뒤가 다 어긋나니 합계부터 맞춘다.
daily = df.sum()
EXPECT = {"2번": 10577, "3번": 14948, "4번": 18297, "5·6번": 1970}
ok = all(int(daily[g]) == EXPECT[g] for g in ACTIVE_GATES)
print(f"     검산: 출국장 합계 {'전부 OK' if ok else '!! 불일치'} "
      f"(총 {int(daily.sum()):,}명)")

# ================================================================
# STEP 3. 기초 분석 — 집중률과 격차
# 예상은 했지만 4번 하나가 40%를 먹는 건 계산해보니 좀 충격이었다.
# -> [제안서] "4번 출국장 하나에 40% 집중 / 5·6번은 4.3%만 이용"
# ================================================================
total = int(daily.sum())
share = (daily / total * 100).round(1)
print("=" * 58)
print("STEP3 출국장별 집중률")
for g in ACTIVE_GATES:
    print(f"  {g:<5} {int(daily[g]):>6,}명  {share[g]:>5}%")
busiest, idlest = daily.idxmax(), daily.idxmin()
print(f"  → 4번 {share['4번']}% / 5·6번 {share['5·6번']}% [제안서]")
print(f"  → 쏠림 격차: {busiest} vs {idlest} = {daily[busiest]/daily[idlest]:.1f}배")

# ================================================================
# STEP 4. 피크 균등분배 효과 (이론적 상한)
# 피크(08~09시)만 떼서, 모두 똑같이 나누면 4번이 얼마나 빠지나.
# '균등 분배'는 현실엔 없는 가정이라 반드시 '이론적 상한'이라 못박는다.
# 안 그러면 과장이 된다.
# -> [제안서] "1,268명 -> 균등분배 시 935명, 이론적 상한 26.3% 감소"
# ================================================================
peak = df.loc["08~09"]
peak_total = int(peak.sum())
even = peak_total / len(ACTIVE_GATES)
worst = int(peak.max())
```

```
reduction = (worst - even) / worst * 100
print("=" * 58)
print("STEP4 피크(08~09시) 균등분배 시뮬레이션")
print(f"  피크 총 {peak_total:,}명 / 현재 최대(4번) {worst:,}명 "
      f"/ 균등 시 {even:.0f}명")
print(f"  → 부하 감소(이론적 상한) {reduction:.1f}% [제안서: 26.3%]")

# ============================================================
# STEP 5. 이용률별 보수 추정 + 체감 환산
# 앱을 다 쓸 리 없으니 30%만 써도 효과가 있나를 본다.
# 모델은 단순: 효과 = 상한 x 이용률 (쓴 사람만 분산에 기여).
# 이 8%가 오히려 제안서에서 제일 설득력 있던 숫자다.
# -> [제안서] "이용률 30% 가정 시 약 8~10%, 약 100명 = 검색대 1개 라인"
# ============================================================
print("=" * 58)
print("STEP5 앱 이용률별 부하 감소 (보수 추정)")
rates = [0.1, 0.2, 0.3, 0.5, 0.7, 1.0]
sens = {r: reduction * r for r in rates}
for r in rates:
    tag = "  ← [제안서] 약 8~10%" if r == 0.3 else ""
    print(f"  이용률 {int(r*100):>3}% -> {sens[r]:4.1f}%{tag}")
moved = worst * (reduction/100) * 0.3
print(f"  체감: 피크 4번 {worst:,}명 중 약 {moved:.0f}명 분산 "
      f"[제안서: 약 100명 = 검색대 1개 라인]")

# ============================================================
# STEP 6. 파생 지표 <- 데이터 분석의 두 번째 꽃 (용도에 맞는 해석)
# 원자료를 '용도(분산 설계)'에 맞게 다시 해석해서 뽑은 지표들.
# 단순 집계를 넘어, 제안서의 주장을 데이터로 받치는 근거가 된다.
# ============================================================
print("=" * 58)
print("STEP6 파생 지표 (해석 레이어)")
# (6-1) 시간대별 병목 게이트 자동 탐지
# 매 시간 어느 게이트가 최대 부하인지 찾는다.
# 4번이 거의 모든 시간대를 독식하면, 쏠림은 우연이 아니라 '구조'다.
bottleneck = operating.idxmax(axis=1)
share_of_4 = (bottleneck == "4번").mean() * 100
print(f"  [6-1] 시간대별 최대혼잡 게이트가 '4번'인 비율: "
      f"{share_of_4:.0f}% ({(bottleneck=='4번').sum()}/{len(bottleneck)}개 시간대)")
print(f"        → 쏠림은 특정 시간의 우연이 아니라 구조적 현상")
# (6-2) 쏠림이 '상시'인지 '피크 한정'인지
# 처음엔 피크에 쏠림이 심해질 거라 봤는데, 계산해보니 아니었다.
# 4번 집중도가 한산한 때나 피크 때나 거의 같다 = 피크만의 문제가 아니라
# 하루 종일 깔린 '구조적' 쏠림이다. 예상과 달랐지만 이게 더 중요한 발견.
# (제안서가 특정 시간이 아니라 '공간 구조' 문제라고 본 것과 같은 결론)
off_peak = operating.loc[["14~15", "15~16", "16~17"]].sum()  # 비교적 한산한 오후
off_share = off_peak["4번"] / off_peak.sum() * 100
peak_share = peak["4번"] / peak_total * 100
print(f"  [6-2] 4번 집중도: off-peak {off_share:.1f}% vs peak {peak_share:.1f}% "
      f"(차이 {abs(peak_share-off_share):.1f}%p)")
```

```
print(f"        → 시간대와 무관하게 일정 = 피크가 아니라 '구조'가 원인")
# (6-3) 유휴 용량 회수량 (보수적)
# 5·6번을 '흡수처'로 봤을 때 받아낼 수 있는 여객.
# 단, 4번 수준까지 채운다는 건 비현실적이라(그럼 5·6번이 새 4번이 됨),
# '전체 게이트 평균'까지만 끌어올리는 보수적 기준으로 본다.
gate_avg = operating[ACTIVE_GATES].sum().mean()  # 게이트당 하루 평균
gate56_total = operating["5·6번"].sum()
recoverable = gate_avg - gate56_total  # 평균까지 채울 여력
print(f"  [6-3] 유휴 용량: 5·6번 하루 {int(gate56_total):,}명 vs "
      f"게이트 평균 {gate_avg:,.0f}명")
print(f"        → 평균 수준까지만 흡수해도 약 {recoverable:,.0f}명 추가 수용 여력")
# (6-4) 게이트 간 불균형 — 변동계수(CV)
# 표준편차/평균. 분배가 잘 될수록 0에 가까워진다.
# '현재 vs 균등분배 후'를 한 숫자로 비교 = 개선 효과의 요약.
cv_now = operating[ACTIVE_GATES].sum().std() / operating[ACTIVE_GATES].sum().mean()
print(f"  [6-4] 게이트 간 불균형(변동계수): 현재 {cv_now:.2f} -> 균등분배 시 0.00")
print(f"        → 분배 설계가 줄여야 할 불균형의 크기를 한 숫자로 표현")

# ================================================================
# STEP 7. 시각화 6종
# 숫자만으론 안 와닿아서 전부 그림으로. 혼잡=빨강, 여유=초록.
# ================================================================
RED, ORANGE, BLUE, GREEN, GRAY = "#D6483B", "#F2A91E", "#2E6FB2", "#2E9E6B", "#B8C0CC"
def _save(fig, name):
    fig.tight_layout(); fig.savefig(name, dpi=130, bbox_inches="tight")
    plt.close(fig); print(f"  saved: {name}")
print("=" * 58)
print("STEP7 그래프 저장")
# (1) 집중률
fig, ax = plt.subplots(figsize=(7,4))
ax.bar(ACTIVE_GATES, [daily[g] for g in ACTIVE_GATES],
       color=[RED if g=="4번" else BLUE for g in ACTIVE_GATES])
for i,g in enumerate(ACTIVE_GATES):
    ax.text(i, daily[g]+200, f"{share[g]}%", ha="center", fontweight="bold")
ax.set_title("T1 출국장별 하루 여객 — 4번 한 곳에 40% 쏠림"); ax.set_ylabel("여객 수(명)")
_save(fig, "chart1_concentration.png")
# (2) 피크 비교
fig, ax = plt.subplots(figsize=(7,4))
ax.bar(ACTIVE_GATES, [peak[g] for g in ACTIVE_GATES],
       color=[RED if g=="4번" else BLUE for g in ACTIVE_GATES])
ax.axhline(even, color=GRAY, ls="--", label=f"균등 분배선 {even:.0f}명")
ax.set_title("피크(08~09시) — 4번 1,268명 / 5·6번 199명"); ax.set_ylabel("여객 수(명)");
ax.legend()
_save(fig, "chart2_peak.png")
# (3) Before/After
fig, ax = plt.subplots(figsize=(7,4))
x = range(len(ACTIVE_GATES))
ax.bar([i-0.2 for i in x], [peak[g] for g in ACTIVE_GATES], 0.4, label="현재",
color=RED)
ax.bar([i+0.2 for i in x], [even]*len(ACTIVE_GATES), 0.4, label="균등 분배 후",
```

```
color=GREEN)
ax.set_xticks(list(x)); ax.set_xticklabels(ACTIVE_GATES)
ax.set_title(f"분배 전후 — 피크 4번 {reduction:.1f}% 감소(이론적 상한)")
ax.set_ylabel("여객 수(명)"); ax.legend()
_save(fig, "chart3_before_after.png")
# (4) 이용률 곡선
fig, ax = plt.subplots(figsize=(7,4))
xs=[int(r*100) for r in rates]; ys=[sens[r] for r in rates]
ax.plot(xs, ys, marker="o", color=BLUE, lw=2)
ax.scatter([30],[sens[0.3]], color=RED, zorder=5, s=90)
ax.annotate(f"30% → {sens[0.3]:.1f}%", (30,sens[0.3]),
            textcoords="offset points", xytext=(10,-18), color=RED, fontweight="bold")
ax.set_title("앱 이용률별 부하 감소 — 전부 안 써도 효과는 난다")
ax.set_xlabel("앱 이용률(%)"); ax.set_ylabel("부하 감소(%)"); ax.grid(alpha=.3)
_save(fig, "chart4_adoption.png")
# (5) 시간대별 총 여객
fig, ax = plt.subplots(figsize=(9,4))
hourly = operating.sum(axis=1)
ax.bar(operating.index, hourly,
       color=[RED if h=="08~09" else BLUE for h in operating.index])
ax.set_title("시간대별 T1 출국 총 여객 — 08~09시가 피크"); ax.set_ylabel("여객 수(명)")
plt.xticks(rotation=90)
_save(fig, "chart5_hourly.png")
# (6) 시간대별 최대혼잡 게이트 — 파생지표 6-1 시각화
# 매 시간 어느 게이트가 1등인지 색으로. 4번(빨강)이 대부분을 덮으면
# 쏠림이 구조라는 게 눈으로 보인다.
fig, ax = plt.subplots(figsize=(9,4))
cmap = {"2번":BLUE, "3번":ORANGE, "4번":RED, "5·6번":GREEN}
ax.bar(operating.index, [operating.loc[h].max() for h in operating.index],
       color=[cmap[bottleneck[h]] for h in operating.index])
ax.set_title(f"시간대별 최대혼잡 게이트 — {share_of_4:.0f}%가 4번(빨강)")
ax.set_ylabel("해당 시간 최대 여객(명)")
handles=[plt.Rectangle((0,0),1,1,color=c) for c in cmap.values()]
ax.legend(handles, cmap.keys(), ncol=4, fontsize=9)
plt.xticks(rotation=90)
_save(fig, "chart6_bottleneck.png")
print("=" * 58)
print("끝. 40.0% / 격차 9.3배 / 상한 26.3% / 보수 30%=약8% / "
      f"4번이 최대혼잡 {share_of_4:.0f}% 시간대 독식")
```