

수요 예측 모델 개발 프로젝트

편의점 수요예측 & 발주 추천

박종훈, 이다미, 이혜지, 전민경

2025.11.3 ~ 2025.11.28

목차

- 1 CSV 업로드 기능(데이터)
- 2 data 폴더 선택 기능(데이터)
- 3 자동 컬럼 매핑(데이터)
- 4 Optuna(학습/모델)
- 5 검증 및 학습 시작(학습/모델)
- 6 예측/발주
- 7 강수량에 따른 우산 판매량 분석(그래프)
- 8 기온에 따른 군고구마 판매량 분석(그래프)
- 9 우산/군고구마 제외 모든 상품 판매량 분석(그래프)
- 10 진단 및 로그

CSV 업로드 기능(데이터)

CSV 업로드 또는 선택

☒ 파일명(source) 열 추가 ②

CSV 파일 업로드(여러 개 가능)

Drag and drop files here
Limit 200MB per file • CSV

Browse files

```
# --- 다중 파일 업로드 ---
with cols[0]:
    up_multi = st.file_uploader("CSV 파일 업로드(여러 개 가능)", type=["csv"], accept_multiple_files=True, key="multi_up")
    if up_multi:
        dfs = []
        for f in up_multi:
            raw = f.read()
            df_i = read_csv_flexible(io.BytesIO(raw))
            if add_source:
                df_i["source"] = f.name
            dfs.append(df_i)
            # data/에 저장
            save_path = os.path.join(DATA_DIR, f.name)
            try:
                with open(save_path, "wb") as fp:
                    fp.write(raw)
            except Exception as e:
                st.warning(f"파일 저장 경고({f.name}): {e}")
        try:
            list_data_files.clear() # 캐시 무효화
        except Exception:
            pass
        df = pd.concat(dfs, axis=0, ignore_index=True, sort=True)
        st.session_state["df"] = df
        st.success(f"업로드/결합 완료: {df.shape} (파일 {len(dfs)}개)")
        st.dataframe(df.head(20), use_container_width=True)
```

1. 다중 업로드 지원

- st.file_uploader로 여러 CSV 동시 업로드
- 각 파일을 read_csv_flexible()로 로딩 후 concat

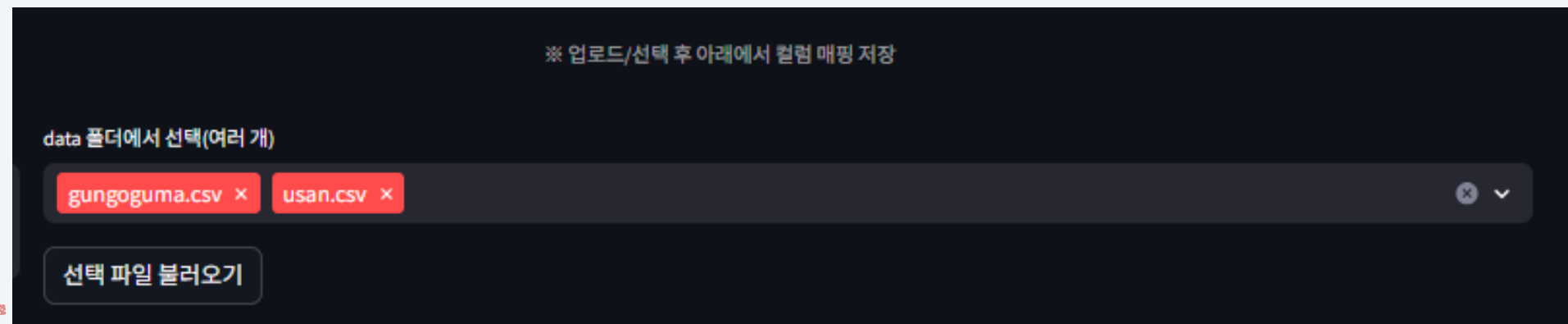
2. source 컬럼 자동 생성

- 옵션 선택 시 각 행에 원본 파일명 기록
- 데이터 출처 추적 가능

3. data 폴더 자동 저장

- 업로드된 파일을 data/에 저장해 재사용 가능

data 폴더 선택 기능(데이터)



```
# --- data 폴더에서 다중 선택 ---
with cols[1]:
    files = list_data_files()
    picks = st.multiselect("data 폴더에서 선택(여러 개)", files)
    if st.button("선택 파일 불러오기", disabled=(len(picks)==0)):
        dfs = []
        for name in picks:
            path = os.path.join(DATA_DIR, name)
            df_i = read_csv_flexible(path)
            if add_source:
                df_i["source"] = name
            dfs.append(df_i)
        df = pd.concat(dfs, axis=0, ignore_index=True, sort=True)
        st.session_state["df"] = df
        st.success(f"불러오기/결합 완료: {df.shape} (파일 {len(dfs)}개)")
        st.dataframe(df.head(20), use_container_width=True)
```

1. 저장된 CSV 불러오기

- data 폴더 파일 목록 자동 로딩
- multiselect로 다중 선택 가능

2. 파일 결합 자동 처리

- 선택된 파일들을 concat하여 하나의 DataFrame 생성

3. source 처리 동일 적용

- 업로드 방식과 동일하게 원본 파일명 기록 가능

자동 컬럼 매핑(데이터)

자동 컬럼 매핑 — 선택 없이 자동 적용됩니다.

현재 자동 매핑 결과:

	역할	컬럼
0	날짜(date)	날짜
1	수요/판매량(target)	일일판매량
2	지역/점포(region)	지역
3	브랜드(선택)	브랜드
4	상품/품목(선택)	상품아이디

```
# --- 자동 컬럼 매핑 + 보정 ---
if "df" in st.session_state:
    st.divider()
    st.caption("자동 컬럼 매핑 — 선택 없이 자동 적용됩니다.")

    df = st.session_state["df"]

    # auto_map_columns 결과 사용
    auto = auto_map_columns(df)
    mapping = [
        "date": auto.get("date"),
        "target": auto.get("target"),
        "region": auto.get("region"),
        "brand": auto.get("brand"),
        "item": auto.get("item"),
    ]

    st.session_state["mapping"] = mapping

    # ★ data 폴더용 보정:
    # seoul_gyeonggi_with_demand.csv / usan.csv / gungoguma.csv 는
    # auto_map_columns가 타겟을 '강수량'으로 잡는 케이스가 있어서,
    # '일일판매량' 컬럼이 있으면 그걸 target으로 강제 교체
    if mapping.get("target") == "강수량" and "일일판매량" in df.columns:
        mapping["target"] = "일일판매량"

    # 확인용으로만 읽기 전용 테이블 표시
    mapping_view = pd.DataFrame(
        {
            "역할": ["날짜(date)", "수요/판매량(target)", "지역/점포(region)", "브랜드(선택)", "상품/품목(선택)"],
            "컬럼": [
                mapping.get("date"),
                mapping.get("target"),
                mapping.get("region"),
                mapping.get("brand"),
                mapping.get("item"),
            ],
        }
    ),
```

1. auto_map_columns 자동 적용

- 날짜(date), 판매량(target), 지역(region) 등 자동 탐지

2. 보정 로직 포함

- target이 '강수량'으로 잡힌 경우 → '일일판매량'으로 자동 교체

3. 매핑 결과 테이블 표시

- 매핑된 컬럼을 표로 보여줘서 확인만 하면 됨

Optuna(학습/모델)

모델 학습

☐ Optuna 하이퍼파라미터 튜닝 사용

Optuna 시도 횟수



```
with tabs[1]:
    st.subheader("모델 학습")

    use_optuna = st.checkbox("Optuna 하이퍼파라미터 튜닝 사용", value=False)
    trials = st.slider("Optuna 시도 횟수", 5, 60, 15, 5)

    if "df" not in st.session_state or "mapping" not in st.session_state:
        st.info("먼저 ☺ 탭에서 데이터와 컬럼 매핑을 지정하세요.")
    else:
```

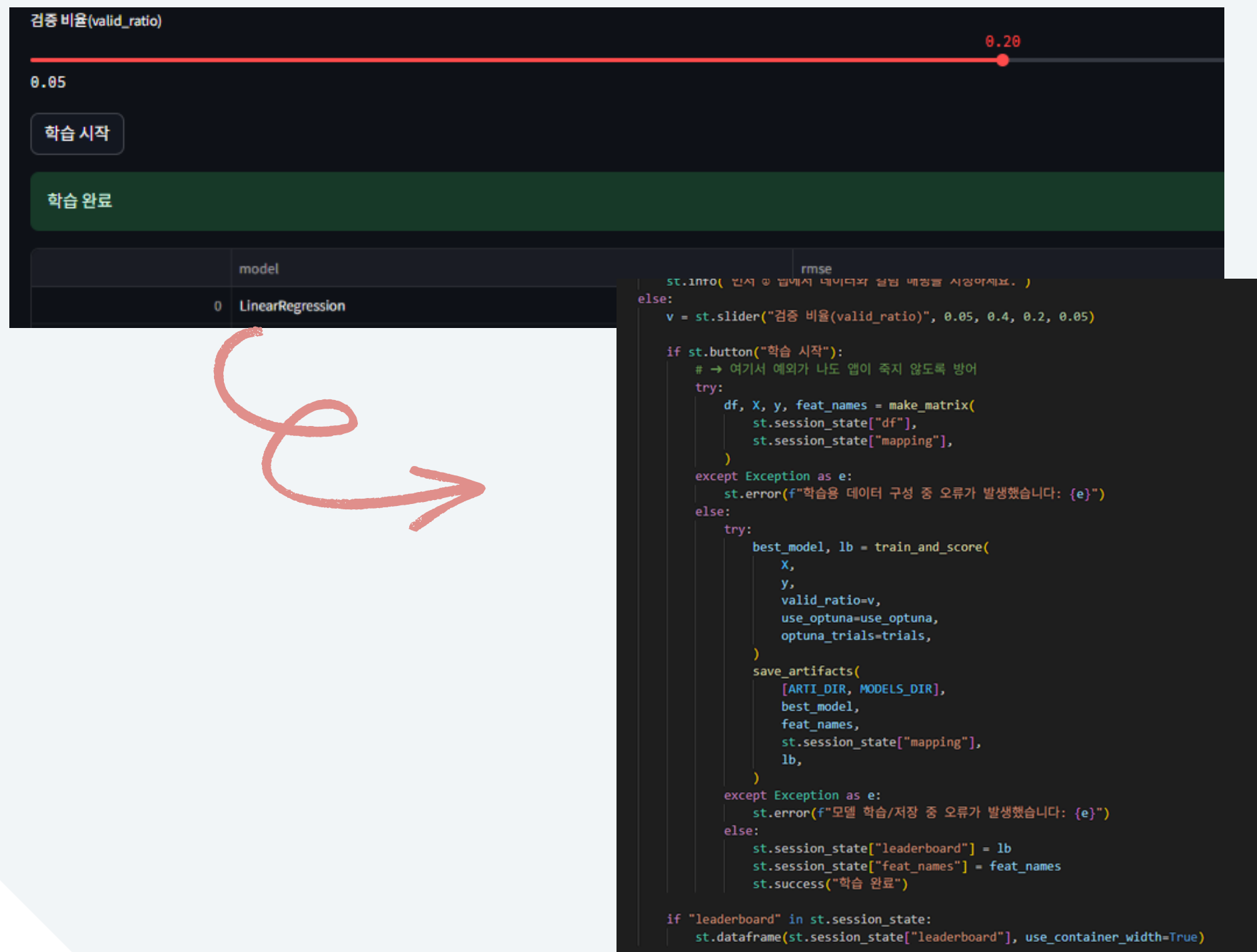
1. Optuna 하이퍼파라미터 튜닝 사용

- 체크하면 Optuna로 자동 튜닝(최적 하이퍼파라미터 탐색)을 수행
- 체크 안 하면 → 기본 설정으로 빠르게 학습
- 초보자/빠른 테스트: OFF
- 성능 최적화: ON

2. Optuna 시도 횟수(trials)

- Optuna가 “몇 번 시도하면서” 최적의 하이퍼파라미터를 찾을지 결정
- 예: 15라면 15개의 조합을 테스트함
- 숫자를 크게 할수록 정확도 ↑, 하지만 시간이 오래 걸림

검증 및 학습 시작(학습/모델)



검증 비율(valid_ratio)

0.05

0.20

학습 시작

학습 완료

	model	rmse
0	LinearRegression	

```
st.info( "분석 및 학습에서 데이터와 실행 매핑을 시각화해보. ")
else:
    v = st.slider("검증 비율(valid_ratio)", 0.05, 0.4, 0.2, 0.05)

if st.button("학습 시작"):
    # → 여기서 예외가 나도 앱이 죽지 않도록 방어
    try:
        df, X, y, feat_names = make_matrix(
            st.session_state["df"],
            st.session_state["mapping"],
        )
    except Exception as e:
        st.error(f"학습용 데이터 구성 중 오류가 발생했습니다: {e}")
    else:
        try:
            best_model, lb = train_and_score(
                X,
                y,
                valid_ratio=v,
                use_optuna=use_optuna,
                optuna_trials=trials,
            )
            save_artifacts(
                [ARTI_DIR, MODELS_DIR],
                best_model,
                feat_names,
                st.session_state["mapping"],
                lb,
            )
        except Exception as e:
            st.error(f"모델 학습/저장 중 오류가 발생했습니다: {e}")
        else:
            st.session_state["leaderboard"] = lb
            st.session_state["feat_names"] = feat_names
            st.success("학습 완료")

if "leaderboard" in st.session_state:
    st.dataframe(st.session_state["leaderboard"], use_container_width=True)
```

3. 검증 비율(valid_ratio)

- 전체 데이터 중 검증 데이터로 사용할 비율
- 모델 성능을 평가하기 위해 일부 데이터를 따로 떼어놓는 과정

4. 학습 시작 버튼

- 클릭하면 내부적으로 다음 수행:
 - a. make_matrix() → X, y 데이터 생성
 - b. train_and_score() → 모델별 학습 & 평가
 - c. (옵션) Optuna 튜닝 실행
 - d. save_artifacts() → 모델·매핑·피쳐 목록 저장
 - e. 리더보드 생성 및 출력

예측/발주

1. 고정: horizon_days = 14

- 데이터의 마지막 날 2025/06/27 을 기준으로 CSV에서 가격·재고를 자동 인식하여 2주(14일) 기준의 예측총량·예상매출·권장발주량을 산출해 대시보드로 표시

2. 세그먼트 선택 (region, brand, item 매핑, 각각 선택 가능)

- 지역과 브랜드 상품 코드를 선택하여 재고 소진 예상 일수와 2주 총 예상매출 예측

※ 예측수량 x 가격 x 정확도 보정 = 금액예측

지역 선택	브랜드 선택	상품아이디 선택
Gyeonggi	7ELEVEN	7ELEVEN_B004
CSV '가격' 컬럼에서 가격 1,800원 자동 인식.		
재고 '재고' 자동 인식 → 47개		
예측 기간(일)	재고 소진 예상일수	2주 총 예상 매출
14	2.0	601,161원
날짜	예측수량	금액예측
2025-06-28 00:00:00	25.0199	45,035.8323
2025-06-29 00:00:00	25.828	46,490.415
2025-06-30 00:00:00	23.4654	42,237.7757
2025-07-01 00:00:00	23.3485	42,027.2214
2025-07-02 00:00:00	22.6124	40,702.3388
2025-07-03 00:00:00	22.8574	41,143.2936
2025-07-04 00:00:00	23.8214	42,878.4744
2025-07-05 00:00:00	25.3748	45,674.5974
2025-07-06 00:00:00	24.9825	44,968.5346
2025-07-07 00:00:00	23.257	41,862.6533

※ 예측수량 x 가격 x 정확도 보정 = 금액예측

가격·재고 자동 인식

```
# 가격 자동 인식
# =====
def guess_price_column(df_seg):
    keys = ["price", "가격", "단가", "판매가", "amount", "금액"]
    for col in df_seg.columns:
        low = col.lower()
        if any(k in low for k in keys):
            return col
    return None
```

<가격 자동 인식>

- 컬럼명에서 키워드(한글/영어)를 포함하면 자동으로 가격/재고 컬럼으로 인식

```
# 재고 자동 인식
# =====
def guess_inventory_onhand(df_seg: pd.DataFrame, mapping):
    candidates = [
        "재고", "재고수", "재고수량",
        "현재재고", "onhand", "on_hand",
        "stock", "inventory",
    ]
    inv_col = None
    for col in df_seg.columns:
        low = col.lower()
        if any(key in low for key in candidates):
            inv_col = col
            break
    if not inv_col:
        return None, None

    series = pd.to_numeric(df_seg[inv_col], errors="coerce").dropna()
    if series.empty:
        return None, None

    return inv_col, float(series.iloc[-1])
```

<재고 자동 인식>

- 재고는 마지막(가장 최신) 관측치를 onhand로 사용
(못 찾으면 사용자 입력을 요구)

예측 계산

```
# 모델 로드
# =====
pkl_path = os.path.join(MODELS_DIR, "best_model.pkl")
if os.path.exists(pkl_path):
    try:
        with open(pkl_path, "rb") as f:
            payload = pickle.load(f)
        model = payload["model"]
        feat_names = payload["feature_names"]
        mapping = payload["mapping"]
```

<모델 로드>

→

```
# **수량 총합**
total_qty_demand = float(fc_df["예측수량"].sum())

# **금액 총합**
fc_df["금액예측"] = (fc_df["예측수량"] * price_val * float(accuracy)).clip(lower=0.0)
total_amt_demand = float(fc_df["금액예측"].sum())
```

<수량/금액 총합 계산>

```
# 3) 재고 자동 인식
# =====
df_seg = st.session_state["df"].copy()
df_seg[dtc] = pd.to_datetime(df_seg[dtc], errors="coerce")
for k, v in seg_vals.items():
    if v and v != "<전체>" and k in df_seg.columns:
        df_seg = df_seg[df_seg[k].astype(str) == str(v)]
df_seg = df_seg.sort_values(dtc)

inv_col, onhand_auto = guess_inventory_onhand(df_seg, mapping)
if onhand_auto is None:
    onhand = st.number_input(
        "현재 재고(직접 입력)",
        min_value=0,
        max_value=100000,
        value=0,
    )
else:
    onhand = onhand_auto
    st.info(f"재고 '{inv_col}' 자동 인식 → {onhand:, .0f}개")

# =====
# 4) 발주량/소진일 계산 (수량 기준)
# =====
avg_daily_qty = total_qty_demand / horizon_days if horizon_days > 0 else 0.0
days_to_out = (onhand / avg_daily_qty) if avg_daily_qty > 0 else float("inf")
rec_qty = max(0.0, total_qty_demand - onhand)
```

<권장발주량 계산>

- 총 수량/금액: 예측된 14일치 수량 합(total_qty_demand)과 가격×보정계수를 곱한 금액 합(total_amt_demand)을 계산.
- 권장 발주량(rec_qty) = 예측 총수요 - 현재 보유재고(onhand) (음수면 0)
- 재고 소진 예상일수: days_to_out = onhand / avg_daily_qty로 계산(무한대 처리 포함)

결과 표시(대시보드 지표)

```
c1, c2, c3 = st.columns(3)
c1.metric("예측 기간(일)", f"{horizon_days}")
c2.metric("재고 소진 예상일수", "∞" if np.isinf(days_to_out) else f"{days_to_out:,.1f}")
c3.metric("2주 총 예상 매출", f"{total_amt_demand:,.0f}원")
```

- 계산된 주요 지표(예측기간, 소진예상일수, 2주 총 매출)를 metric으로 직관적으로 표시하고 테이블 표로 보여줌

강수량에 따른 우산 판매량 분석(그래프)

분석할 연월(YYYY-MM)

2025-03

2024-10

2024-11

2024-12

2025-01

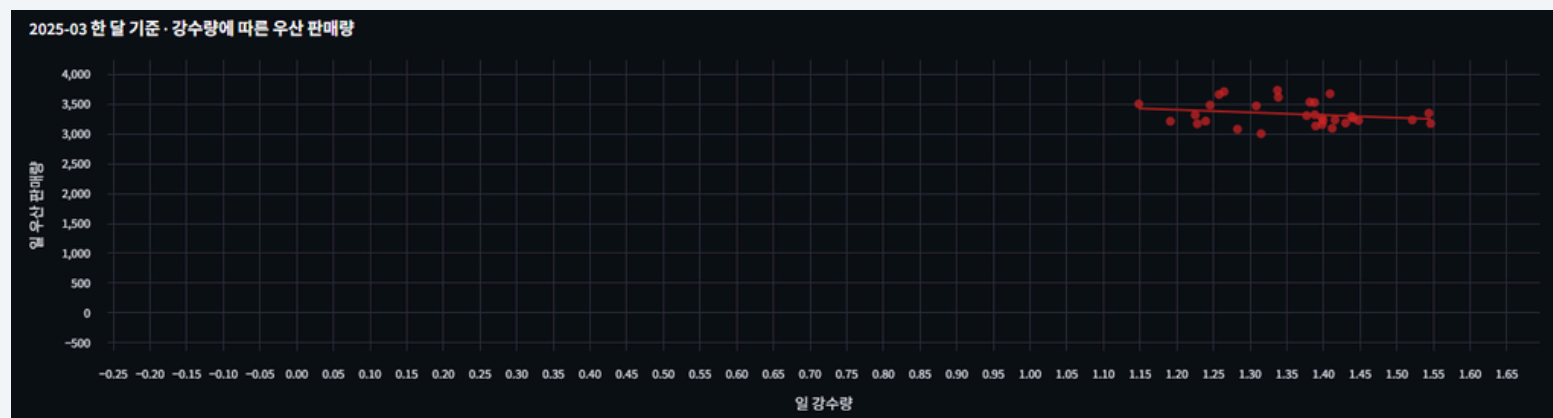
2025-02

2025-03

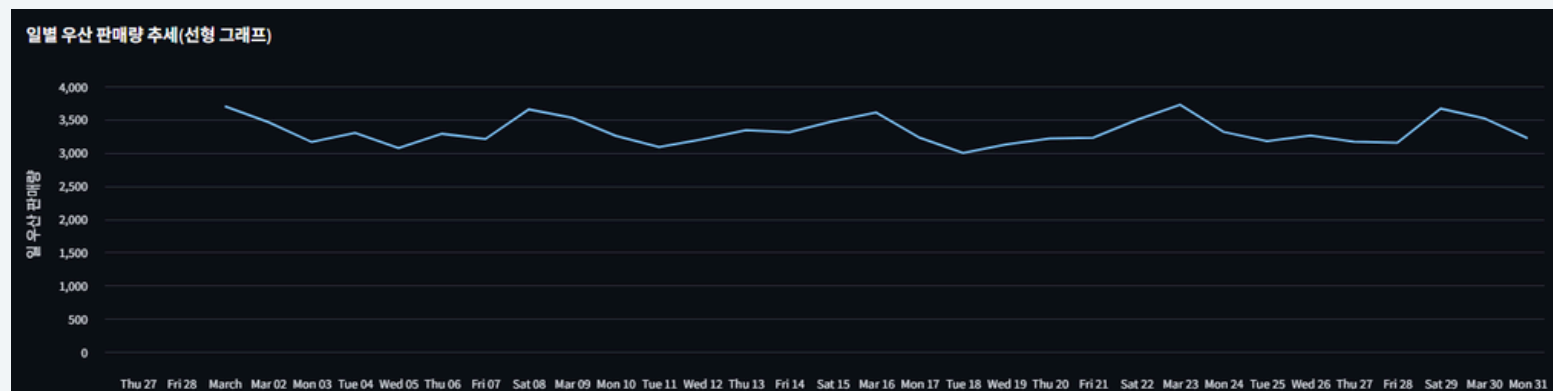
2025-04

2025-05

<분석할 연월(YYYY-MM)>



<2025-03 한 달 기준 · 강수량에 따른 우산 판매량>



<일별 우산 판매량 추세(선형 그래프)>

1. 강수량과 우산 판매량 관계 파악

- 데이터에 있는 강수량과 우산 판매량의 연관성 파악
: 강수량이 높을수록 우산 판매량도 높아진다고 예측

2. 일별 우산 판매량 추세

- 한 달동안 일별로 우산의 판매량 분석
: 많이 팔린 날일수록 그 날의 강수량이 높았을 것으로 예측

기온에 따른 군고구마 판매량 분석(그래프)

분석할 연월(YYYY-MM)

2025-03

2024-10

2024-11

2024-12

2025-01

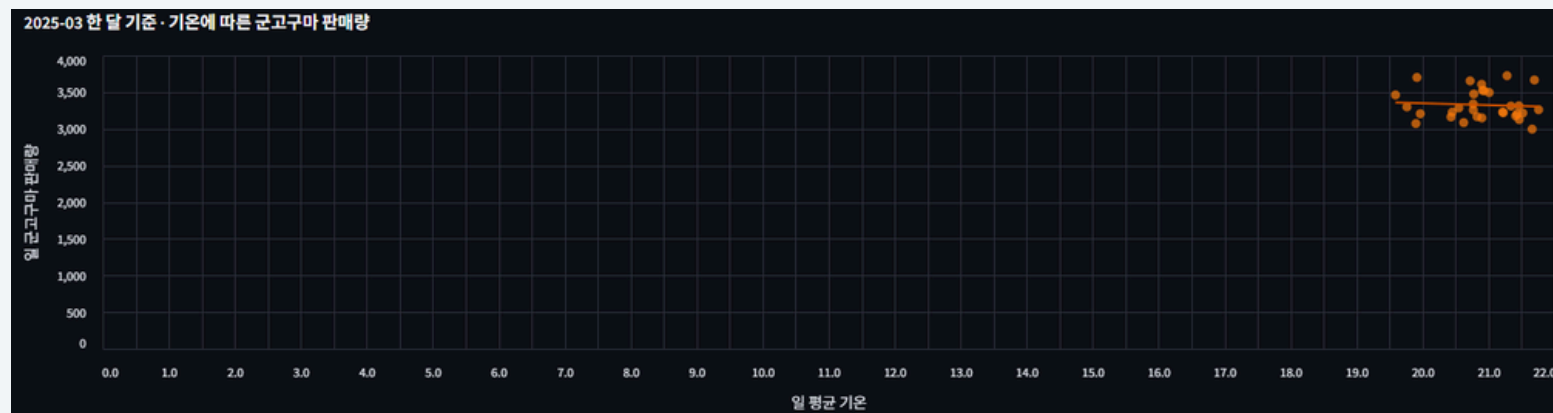
2025-02

2025-03

2025-04

2025-05

<분석할 연월(YYYY-MM)>



<2025-03 한 달 기준 · 기온에 따른 군고구마 판매량>



<일별 군고구마 판매량 추세(선형 그래프)>

1.기온과 군고구마 판매량 관계 파악

- 데이터에 있는 기온과 군고구마 판매량의 연관성 파악
: 기온이 낮을수록 군고구마의 판매량이 높아진다고 예측

2.일별 군고구마 판매량 추세

- 한 달동안 일별로 군고구마의 판매량 분석
: 많이 팔린 날일수록 그 날의 기온이 낮았을 것으로 예측

우산/군고구마 제외 모든 상품 판매량 분석(그래프)

분석할 연월(YYYY-MM)

2025-03

2024-10

2024-11

2024-12

2025-01

2025-02

2025-03

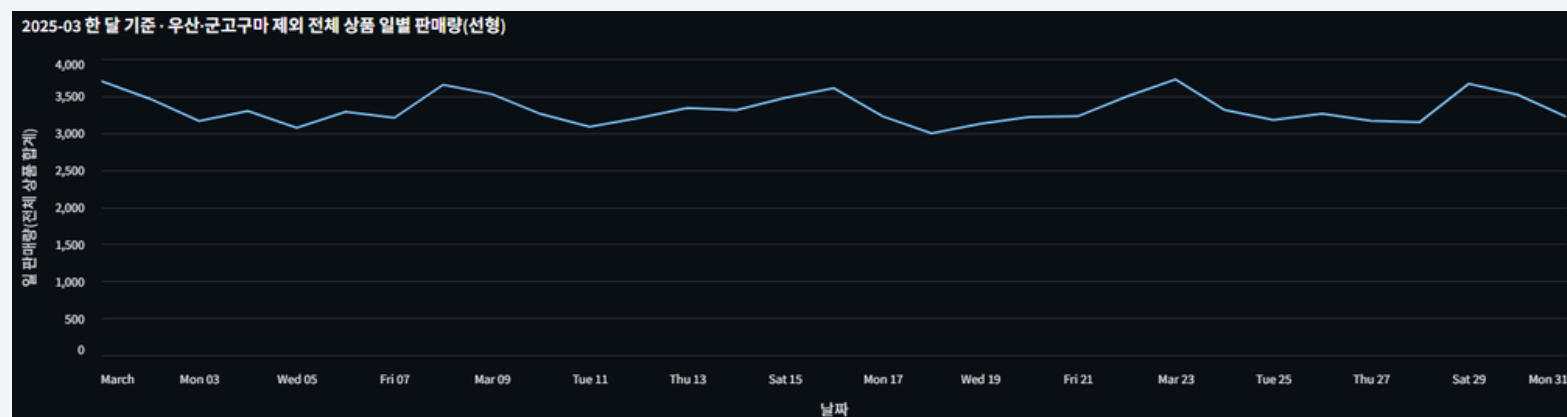
2025-04

2025-05

<분석할 연월(YYYY-MM)>

1. 우산과 군고구마의 데이터를 제외한 나머지 상품의 판매량 분석

- 우산과 군고구마의 데이터를 제외하고 나머지 상품들의 데이터만 이용하여
일별 판매량 분석
: 전체적으로 높았던 수요일과 낮았던 수요일을 확인해 볼 수 있음



<2025-03 한 달 기준 · 우산·군고구마 제외 전체 상품 일별 판매량(선형)>

분석(그래프)

ex) 우산

```
# 연-월 선택 (YYYY-MM 형식만 보여줌)
labels = [lab for lab, _ in ym_options]
default_idx = len(labels) - 1 # 기본값: 가장 최근 월
sel_label = st.selectbox("분석할 연월(YYYY-MM)", labels, index=default_idx, key="ym_umbrella")
sel_period = dict(ym_options)[sel_label]

# 선택한 한 달만 필터
df_month = df_u_raw[df_u_raw["year_month"] == sel_period].copy()
if df_month.empty:
    st.info(f"{sel_label} 에 해당하는 데이터가 없습니다.")
else:
    # 일 단위 집계
    df_month["date_only"] = df_month[date_col].dt.date
    daily = (
        df_month.groupby("date_only", as_index=False)
        .agg({sales_col: "sum", rain_col: "mean"})
        .dropna(subset=[sales_col, rain_col])
    )
    daily = daily.rename(
        columns={"date_only": "date", sales_col: "sales", rain_col: "rain"}
    )

    if daily.empty:
        st.info("해당 연월에서 일별로 집계할 수 있는 데이터가 없습니다.")
    else:
        st.markdown(f"*{sel_label} 한 달 기준 · 강수량에 따른 우산 판매량*")

        base = alt.Chart(daily).encode(
            x=alt.X("rain:Q", title="일 강수량"),
            y=alt.Y("sales:Q", title="일 우산 판매량"),
        )
```

<날짜 선택>

```
# 붉은색 산점도 + 선형 회귀선
points = base.mark_circle(size=70, color="#d62728").encode(
    tooltip=[
        alt.Tooltip("date:T", title="날짜"),
        alt.Tooltip("rain:Q", title="강수량"),
        alt.Tooltip("sales:Q", title="우산 판매량"),
    ]
)
reg_line = base.transform_regression("rain", "sales").mark_line(color="#b22222")

st.altair_chart((points + reg_line).interactive(), use_container_width=True)

# ★ 추가: 일별 우산 판매량 선형 그래프
st.markdown("**일별 우산 판매량 추세(선형 그래프)**")
line_umbrella = (
    alt.Chart(daily)
    .mark_line()
    .encode(
        x=alt.X("date:T", title="날짜"),
        y=alt.Y("sales:Q", title="일 우산 판매량"),
        tooltip=[
            alt.Tooltip("date:T", title="날짜"),
            alt.Tooltip("sales:Q", title="우산 판매량"),
            alt.Tooltip("rain:Q", title="강수량"),
        ],
    )
)
st.altair_chart(line_umbrella.interactive(), use_container_width=True)
```

<산점도, 선형 그래프>

- 월 단위로 일별 집계를 만들어 강수량 vs 판매량(산점도 + 회귀) 형태로 상관관계를 시각화함
- 군고구마, 전체 상품들도 동일한 패턴으로 판매량의 산점도 + 회귀선, 일형 선형 그래프를 그림

진단 및 로그

1. 로그 기록 기능

- 모델 학습 과정(Log)
- 전처리 단계에서의 컬럼 매핑/결측 처리 기록
- 예측 수행 시 생성된 특징(feature) 정보
- 예측값, 보정계수, 발주 산출 과정 메시지
- 오류(error) / 경고(warning) 자동 감지

2. 진단 화면에서 제공하는 정보

- 데이터셋 불러오기 여부
- 학습된 모델 존재 여부(best_model.pkl 체크)
- 전처리된 데이터 샘플(표 형태)
- 최근 예측 결과 요약
- 경고 또는 오류 메시지 표시

3. 사용 목적

- 문제 발생 시 빠른 원인 파악
- 더 신뢰할 수 있는 예측 시스템 운영
- 모델 개선 포인트 발굴

감사합니다.

thank you!