Appendix:

Technical Supporting Documentation

This section details the complete technical workflow, rationale, and implementation logic behind the visual and analytical components of the memo.

It documents how the raw data was preprocessed, aggregated, and post-processed for interpretation. Each decision from scaling functions to visualization styling is explained with its analytical justification.

1. Dataset Overview and Cleaning

The dataset `Coffe_sales_with_menu_price` consists of **3,547 transactional records** with columns such as:

- `money` : revenue per transaction (numeric, continuous)
- `hour_of_day` : 24-hour timestamp of purchase
- `Weekday` : day of week (categorical)
- `coffee_name` : product purchased
- `Time_of_Day` : derived categorical variable
- `Weekdaysort` : numeric column for weekday sorting

The dataset contained no missing values in the key variables used for analysis. Currency values were formatted as floats, rounded for readability in visualization. Outliers (very high single-transaction amounts) were retained since they likely correspond to bulk orders operationally relevant for sales volume planning.

2. Feature Engineering

Two key engineered features were created:

- **Time_of_Day** derived from `hour_of_day` to represent broad customer behavior periods (Morning, Afternoon, Evening).
- **Weekdaysort** assigns numeric order to weekdays for consistent plotting.

The cutoffs for `Time_of_Day` were defined as:

- Morning: 0 ≤ hour < 11
- Afternoon: 11 ≤ hour < 17
- Evening: 17 ≤ hour < 24

These bins align with typical coffee consumption and operational patterns (morning commute, midday office breaks, evening social visits).

```
day_bins = [0, 11, 17, 24]
day_labels = ["Morning", "Afternoon", "Evening"]
```

```
df["Time_of_Day"] = pd.cut(df["hour_of_day"], bins=day_bins,
labels=day_labels, right=False)

weekday_order = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]
df["Weekdaysort"] = df["Weekday"].apply(lambda x:
weekday_order.index(x))
```

3. Aggregation Logic

Several aggregation layers were created to support different visual analyses.
Each aggregation corresponds to a managerial insight discussed in the memo:

Aggregation	Purpose	Analytical Rationale
`sales_by_hour`	Total sales by hour	Identifies intra-day peaks and operational "rush hours."
`sales_by_weekday`	Total sales by day	Distinguishes weekday vs. weekend trends.
`sales_by_coffee`	Total sales by product	Ranks products by contribution to revenue.
`pivot_sales`	2D pivot (Weekday × Time_of_Day)	Enables heatmap of sales by time and day.
`coffee_heatmap`	2D pivot (Coffee Type × Time_of_Day)	Visualizes product popularity across dayparts.

```
sales_by_hour = df.groupby("hour_of_day", as_index=False)
["money"].sum()
sales_by_weekday = (
    df.groupby(["Weekday", "Weekdaysort"], as_index=False)
["money"].sum().sort_values("Weekdaysort")
)
sales_by_coffee = (
    df.groupby("coffee_name", as_index=False)
["money"].sum().sort_values("money", ascending=False)
)
pivot_sales = df.pivot_table(index="Weekday", columns="Time_of_Day",
values="money", aggfunc="sum", fill_value=0)
coffee_heatmap = df.pivot_table(index="coffee_name",
columns="Time_of_Day", values="money", aggfunc="sum", fill_value=0)
```

4. Post-Processing and Derived Metrics

4.1 Normalization and Staffing Function

After calculating total sales per hour (`sales_by_hour`), a heuristic staffing model was applied to convert hourly revenue into a recommended number of staff members.
This step bridges raw financial data with operational guidance allowing sales intensity to be interpreted as workload intensity.

```
import math

# Extract hourly sales as a Series indexed by hour
hour_sales = sales_by_hour.set_index("hour_of_day")["money"]

# Normalize hourly sales between 0 and 1
normalized = hour_sales / hour_sales.max()

# Scale and convert normalized values into discrete staff counts
recommended_staff = (normalized * 5).apply(math.ceil) + 1

# Create final DataFrame with hour and staff recommendations
rec_hours = (
    pd.DataFrame({"hour_of_day": range(0, 24)})
    .merge(recommended_staff.rename("recommended_staff"),
on="hour_of_day", how="left")
    .fillna(1)
)
rec_hours["recommended_staff"] =
rec_hours["recommended_staff"].astype(int)
rec_hours.head(10)
```

Explanation of the Transformation

The staffing calculation proceeds in five key steps:

1. `hour_sales` — represents total hourly revenue, e.g.:

hour_of_day	money ($)
6	120
7	250
8	430
9	670
10	620
11	540

Here, 9–10 AM is clearly the high-demand period.

2. **Normalization:**
   Dividing by `hour_sales.max()` scales all hourly sales to a 0–1 range:
   $[\text{normalized}_i = \frac{\text{sales}_i}{\max(\text{sales})}]$
   For example, if 9 AM = $670 and 6AM =$120, then:
   $[\text{normalized}(6\text{AM}) = 120/670 \approx 0.18]$ This allows comparison of relative sales intensity across hours.

3. **Scaling:**
   Multiplying by 5 maps the normalized sales into a theoretical range of 0–5.
   This constant (5) represents the **maximum number of employees needed during**

peak demand for a small- to mid-sized coffee shop.
It's a tunable parameter that can be adapted for larger stores.

4. **Ceiling Function (`math.ceil()`):**
   Rounds each scaled value **up** to the nearest integer, ensuring that fractional workloads are represented by whole staff members.
   For example:
   - 9 AM (normalized 1.00 × 5 = 5.00 → ceil = 5)
   - 6 AM (normalized 0.18 × 5 = 0.9 → ceil = 1)
5. **Baseline Adjustment (+1):**
   Adds a **minimum coverage of one staff member**, ensuring that even during very low traffic hours (late evenings or early mornings), at least one barista is on duty for safety and customer service.

The final function therefore transforms revenue into operationally interpretable staff counts:

$$[\text{Recommended Staff}_i = \lceil (\text{Sales}_i / \text{Max Sales}) \times 5 \rceil + 1]$$

This is a **nonlinear heuristic**, meaning small increases in sales at lower hours may not proportionally increase staffing, but high-volume hours rapidly reach peak staffing levels.

Example: Inspect calculated recommendations for peak and off-peak hours

```
rec_hours.loc[rec_hours["hour_of_day"].isin([6, 9, 14, 20])]
```

Example Interpretation

Hour	Sales ($)	Normalized	Scaled	Ceil	+1	Recommended Staff
6 AM	120	0.18	0.9	1	+1	2
9 AM	670	1.00	5.0	5	+1	6
2 PM	350	0.52	2.6	3	+1	4
8 PM	190	0.28	1.4	2	+1	3

Hence, the model recommends **6 staff members at peak (9 AM)** and **2–3 during slower hours (6 AM, 8 PM)** consistent with observed transaction volume patterns.

Analytical Rationale

- **Normalization:** Enables comparison across stores and days by removing scale bias.
- **Scaling factor (×5):** Reflects realistic peak staffing capacity; can be recalibrated for store size or regional norms.
- **Ceiling and baseline adjustment:** Prevents fractional staffing and ensures continuous coverage.
- **Interpretability:** The resulting staffing chart aligns visually with transaction and revenue peaks, making it actionable for operations teams.

This transformation is **not a predictive model**, but an **empirical operational heuristic** derived from proportional scaling ideal for translating historical demand patterns into scheduling decisions without requiring regression-based forecasting.

5. Reproducibility and Analytical Assumptions

- **Environment:**
  Python 3.10.
  Libraries: `pandas 2.x`, `numpy 1.26+`, `matplotlib 3.8+`, `seaborn 0.13+`.
- **Assumptions:**
  1. Sales volume correlates linearly with staffing demand (sufficient for aggregate-level scheduling).
  2. Customer arrival patterns are consistent across stores in the region.
  3. No external seasonality or promotional data were included; patterns are purely temporal.
- **Reproducibility:**
  Each cell is modular; parameters such as scaling factor (`5` in the staffing heuristic) or top product filter (`K` in product analysis) can be modified to test alternative operational scenarios.