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 `Coffee_sales.csv` 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

A normalization and ceiling function was applied to translate sales intensity into a **recommended staffing level per hour**.

```
import math
hour_sales = sales_by_hour.set_index("hour_of_day")["money"]
```

```
normalized = hour_sales / hour_sales.max()
recommended_staff = (normalized * 5).apply(math.ceil) + 1
```

Explanation:

- The variable `hour_sales` gives absolute sales by hour.
- Dividing by `hour_sales.max()` scales all values between 0–1 (min–max normalization).
- Multiplying by 5 scales the range to 0–5, roughly reflecting relative labor intensity for a small café (where 5 employees represent peak coverage).
- Applying `math.ceil()` ensures discrete staff counts (no fractions).
- Adding +1 ensures a minimum staffing baseline even during off-peak hours.

This heuristic-based transformation mirrors real-world scheduling logic: sales intensity correlates with labor need, but always requires minimum coverage for operations and safety.

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.