Context Statement

In this project, we analyzed coffee sales data to uncover insights that can guide better day to day decisions across multiple store locations. Our intended audience is a **Regional Coffee Shop Manager** who oversees several stores while managing employee schedules, inventory, and customer service under tight time pressure. While experienced in operations behavior, most managers have limited experience interpreting detailed data analyses.

Our analysis focuses on two main questions:

1. What are the busiest times of day and days of the week that generate the highest sales volumes, and how can staffing or store hours be adjusted to match customer demand?
2. Which types of coffee are most in demand at different times of day, and how should inventory levels and promotions be adjusted accordingly?

We address these questions through clear, intuitive visuals that highlight patterns and support quick, actionable decisions on staffing, inventory, and promotions. Since a regional manager oversees multiple stores and teams, they need insights that are quick to grasp and ready to apply. Our goal is to keep the analysis direct, easy to read, and immediately useful for daily operations.

Executive Summary

Running a busy coffee shop means constantly balancing staff, supplies, and customer experience. Managers often rely on intuition for these decisions, but sales data can make those choices clearer and more precise. This report analyzes daily transaction patterns to reveal when customers visit most, what they tend to buy, and how operations can be aligned for smoother service and stronger sales.

The data shows consistent daily and weekly rhythms. Sales surge twice a day around 10 AM and again near 4 PM with the morning rush generating nearly half of daily revenue before 2 PM. Weekday mornings and evenings are the busiest periods, while weekends attract a steadier afternoon crowd that lingers longer over drinks.

These insights point to clear operational improvements. Staffing should peak during morning and afternoon rushes (about six employees on the floor) and scale down to three or four during quieter hours. Core supplies such as espresso beans, milk, and takeaway cups should be fully stocked before opening, while syrups and specialty milks can be replenished later for afternoon drinks like lattes. Promotions can follow the same rhythm commuter bundles in the morning and flavored latte or loyalty specials later in the day.

By aligning staffing, inventory, and promotions with customer behavior, managers can reduce waste, shorten lines, and deliver consistently great service turning daily data into everyday efficiency.

Translating Sales Data into Operational Strategy

A coffee shop's success depends not only on the quality of its drinks but also on how well it aligns its operations with customer routines. Each hour brings its own rhythm of arrivals and orders, revealing how people move through their day. Customer activity captures this rhythm clearly, offering insight into when foot traffic peaks and how staffing and preparation should adjust.

Hourly transaction data, a direct indicator of customer flow, shows two consistent surges,one around 10 AM and another near 4 PM, corresponding to the morning commute and afternoon break. These peaks rise well above the daily average of 209 transactions, signaling predictable high-demand windows that can guide staffing and service planning. Recognizing this rhythm helps managers anticipate demand, deploy staff efficiently, and maintain smooth, timely service throughout the day.

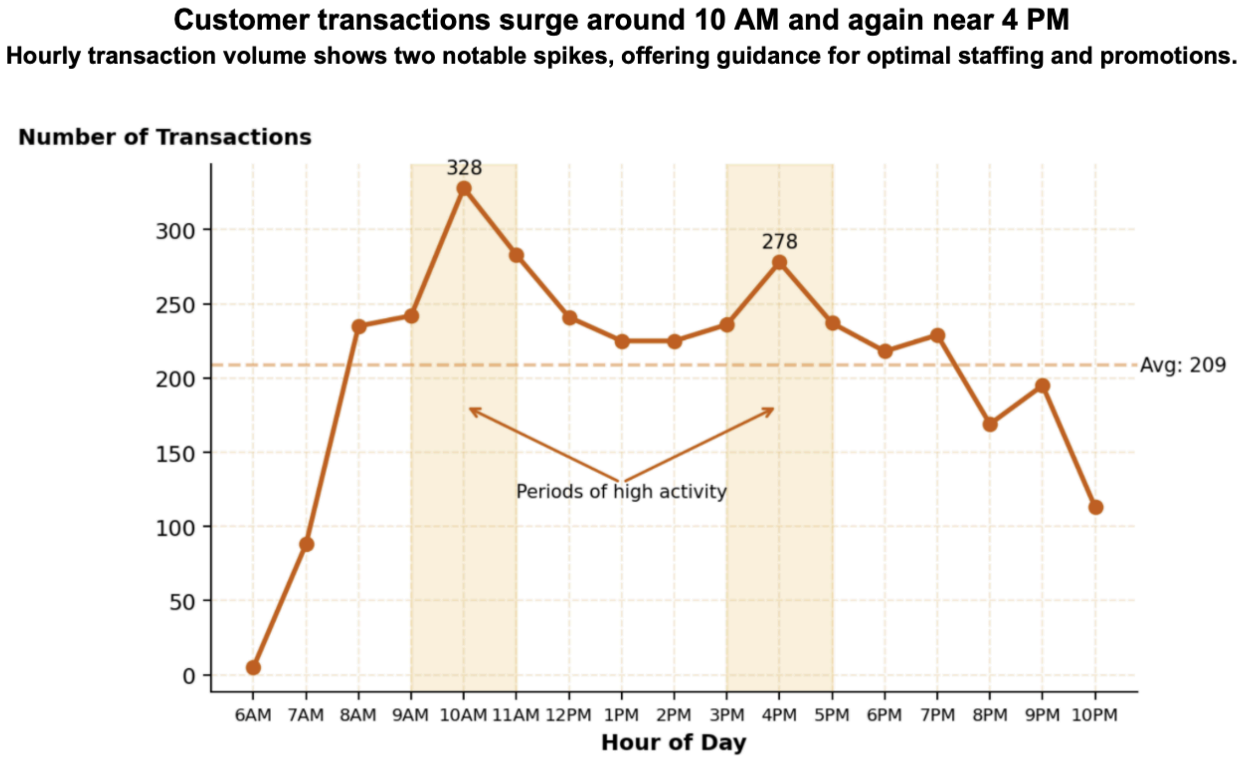


Figure 1: Hourly Transactions

Building on this pattern of customer flow, comparing revenue by both day and time (Figure 2) reveals how spending habits evolve throughout the week. While transaction counts show when customers visit, revenue highlights how much they spend and when. Weekday mornings and evenings generate the highest earnings, reflecting commuter routines and quick, on-the-go purchases before and after work. On weekends, the rhythm softens—afternoons become more dominant, shaped by customers who stay longer, socialize, and spend more per visit.

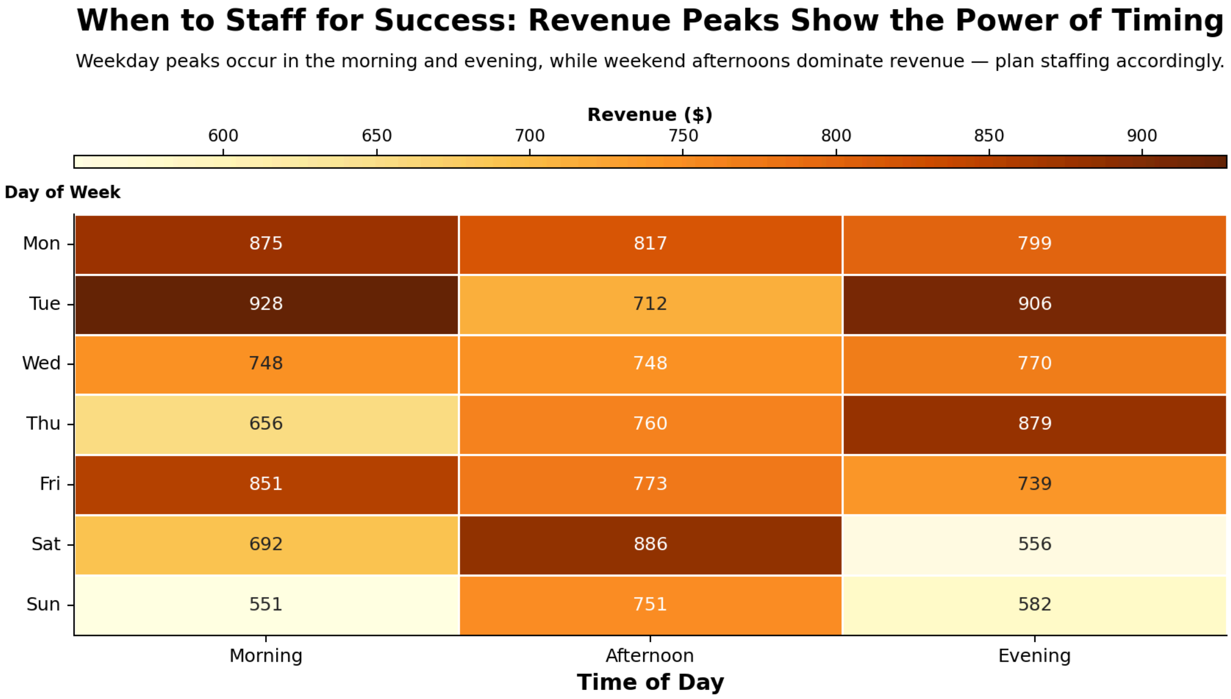


Figure 2: Revenue by Day and Time of Day

Within these patterns, Tuesday stands out as the strongest weekday, especially in the morning and evening rush, while Saturday afternoons dominate weekend sales, when guests linger longer and spend more per visit. These differences indicate that weekday schedules should handle short, intense surges, while weekend staffing should maintain steady coverage and emphasize hospitality. Tailoring labor, prep, and promotions around these rhythms keeps service smooth and costs efficient across all store types.

Translating these insights into operations, the recommended staffing plan in Figure 3 increases coverage to six employees during the 10 AM and 4 PM peaks, when service speed is most critical and lines are longest. During steadier midday hours, maintaining five employees provides sufficient coverage without inflating labor costs, while early mornings and late evenings can taper to three or four staff members. The staffing levels are derived from hourly sales patterns using a normalized scaling approach, outlined in the appendix 4.1. This data-driven scheduling ensures efficiency without compromising service quality, keeping operations responsive to the natural rhythm of customer demand.

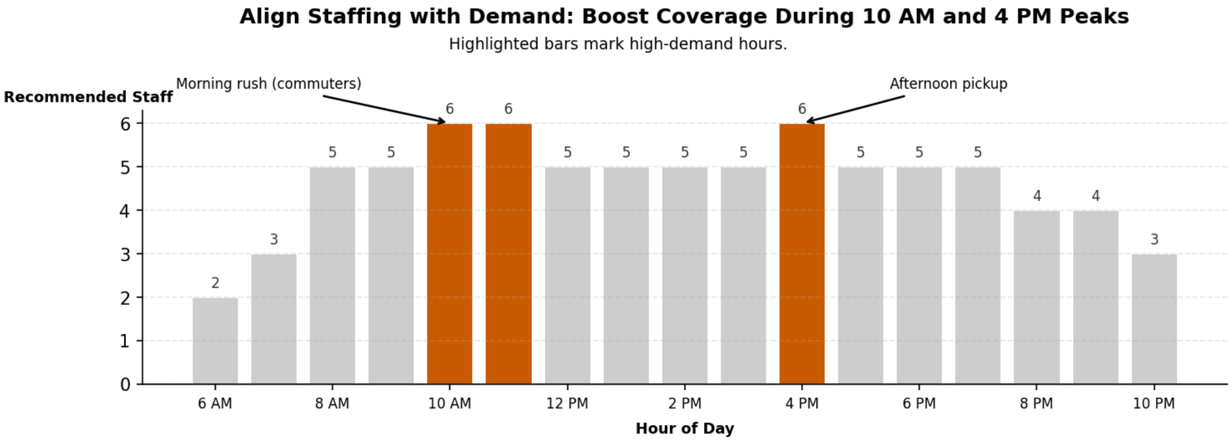


Figure 3: Recommended Staffing Plan

Previously, in Figure 1, we saw sharp transaction spikes around 10 AM and 4 PM, two distinct moments of heightened customer activity. Yet when we look at cumulative revenue across the day (Figure 4), the picture tells a more balanced story: by 2 PM, the shop has earned roughly half of its total daily revenue, with the remaining half coming afterward. This near-even split suggests that while the intensity of customer flow changes throughout the day, sales volume remains strong and sustained. For inventory planning, that means supply should not be front-loaded solely toward the morning rush. Instead, stock levels should be managed dynamically, ensuring full readiness for the early surge while maintaining sufficient ingredients and products for steady afternoon and evening demand.

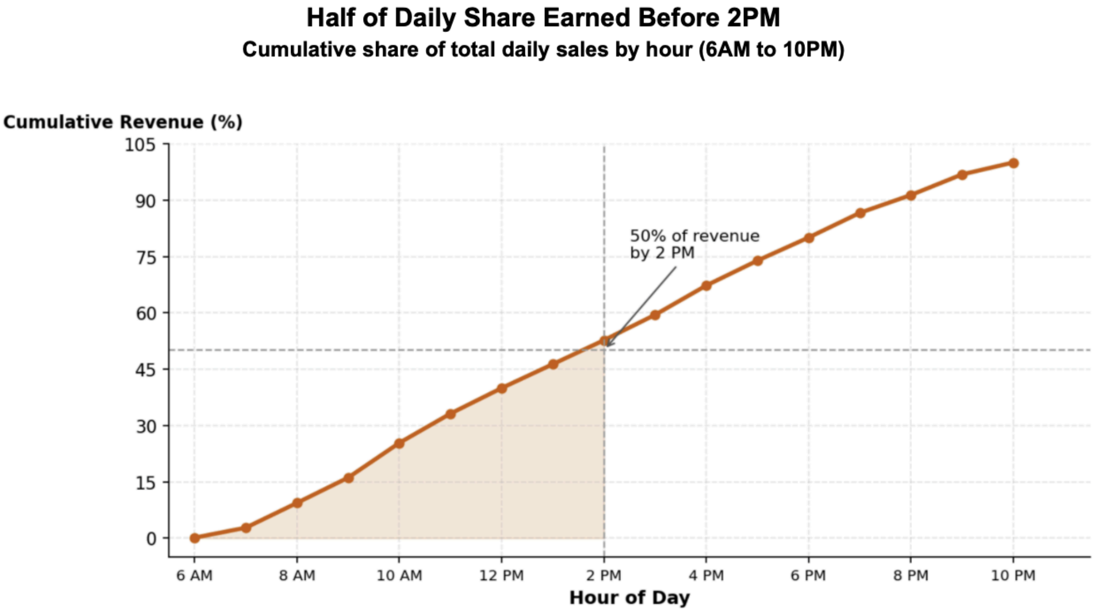


Figure 4: Cumulative Revenue by Hour Shows a More Balanced Story

That being said, the mix of products sold varies noticeably across these periods. As shown in Figure 5, Lattes and Americanos with Milk consistently drive revenue, but their popularity shifts with time of day. Americanos dominate the morning hours, reflecting the need for a strong caffeine boost, while Lattes rise in the afternoon and evening as customers slow down and enjoy a more relaxed, milk-based drink.

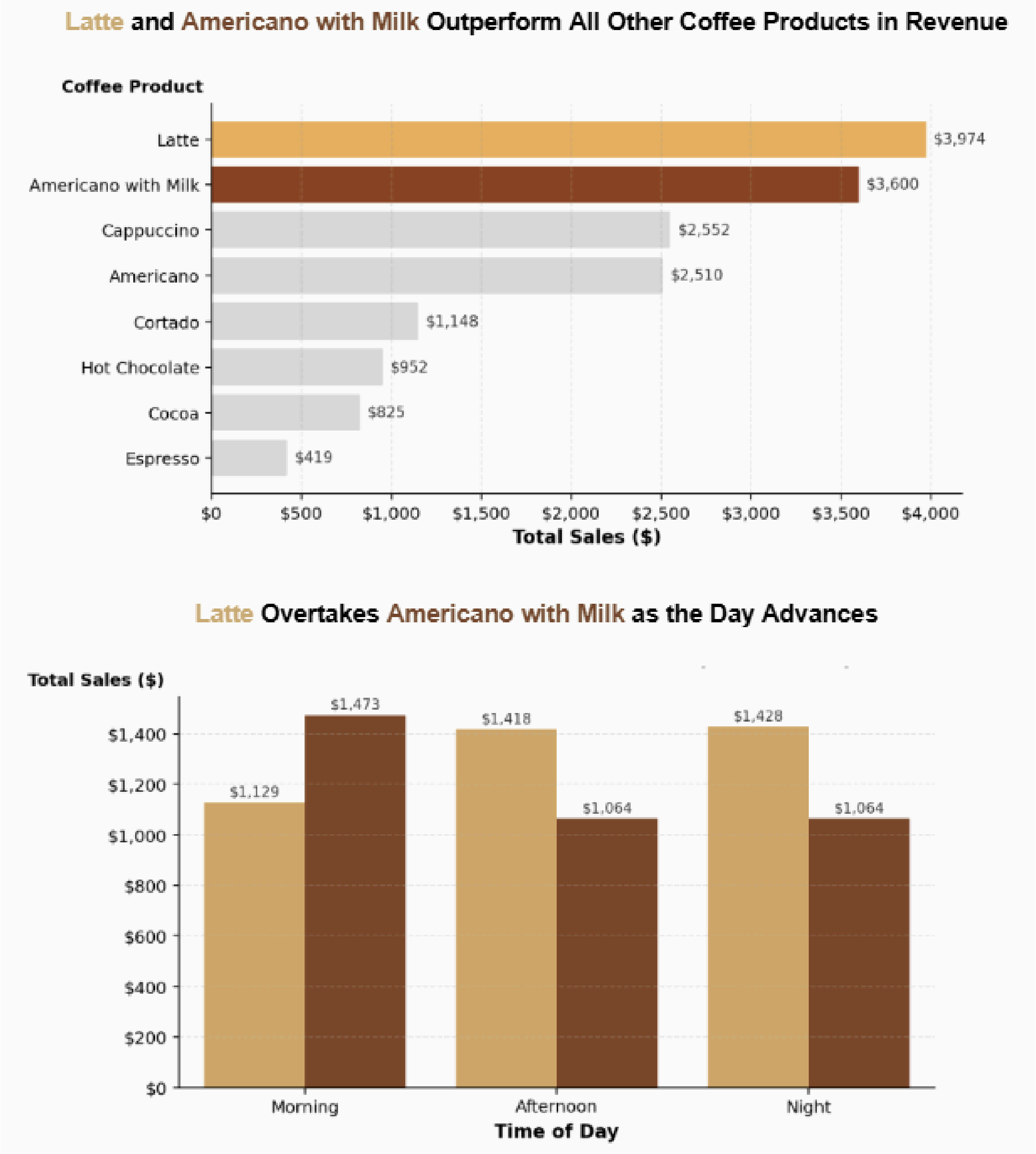


Figure 5: Latte vs. Americano with Milk by Time of Day – “Latte Overtakes as the Day Advances”

Together, these findings point to one clear strategy: match staffing, inventory, and promotions to the natural rhythm of demand. Morning and early afternoon hours drive most revenue, so resources should peak then. Stock core items : espresso beans, milk, and takeaway cups before opening, ensuring stations are ready for the first rush. As the day slows, restock syrups and specialty milks to support afternoon Latte sales and shift focus toward a relaxed, social customer base.

Marketing can mirror these patterns: quick-service deals or commuter bundles in the morning, followed by flavored-Latte promotions or loyalty specials later in the day. By linking preparation, staffing, and marketing to when and what customers buy, managers can reduce waste, improve service flow, and sustain strong sales from open to close. The takeaway is simple , understanding customer timing and taste turns raw transaction data into actionable, everyday decisions that keep stores efficient and customers satisfied.

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 6 AM =$120, then:
[$\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.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MaxNLocator, FuncFormatter
import math
```

In [2]:

```
df = pd.read_csv("Coffe_sales_with_menu_price.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	hour_of_day	cash_type	coffee_name	Time_of_Day	Weekday	Month_name
0	10	card	Latte	Morning	Fri	Mar
1	12	card	Hot Chocolate	Afternoon	Fri	Mar
2	12	card	Hot Chocolate	Afternoon	Fri	Mar
3	13	card	Americano	Afternoon	Fri	Mar
4	13	card	Latte	Afternoon	Fri	Mar

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3547 entries, 0 to 3546
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   hour_of_day  3547 non-null   int64  
 1   cash_type    3547 non-null   object 
 2   coffee_name  3547 non-null   object 
 3   Time_of_Day  3547 non-null   object 
 4   Weekday      3547 non-null   object 
 5   Month_name   3547 non-null   object 
 6   Weekdaysort  3547 non-null   int64  
 7   Monthsort    3547 non-null   int64  
 8   Date         3547 non-null   object 
 9   Time         3547 non-null   object 
 10  money        3547 non-null   float64
dtypes: float64(1), int64(3), object(7)
memory usage: 304.9+ KB
```

In [5]:

```
# convert to datetime
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
```

```
df["Time"] = pd.to_datetime(df["Time"], errors="coerce")
```

```
/var/folders/mc/2wjfdchj6vsffbrpfbfgqw4w0000gn/T/ipykernel_70053/309375
0455.py:3: UserWarning: Could not infer format, so each element will be
parsed individually, falling back to `dateutil`. To ensure parsing is c
onsistent and as-expected, please specify a format.
  df["Time"] = pd.to_datetime(df["Time"], errors="coerce")
```

In [6]:
```
df
```

Out[6]:

	hour_of_day	cash_type	coffee_name	Time_of_Day	Weekday	Month_n
0	10	card	Latte	Morning	Fri	
1	12	card	Hot Chocolate	Afternoon	Fri	
2	12	card	Hot Chocolate	Afternoon	Fri	
3	13	card	Americano	Afternoon	Fri	
4	13	card	Latte	Afternoon	Fri	
...	...	...	...	...	...	
3542	10	card	Cappuccino	Morning	Sun	
3543	14	card	Cocoa	Afternoon	Sun	
3544	14	card	Cocoa	Afternoon	Sun	
3545	15	card	Americano	Afternoon	Sun	
3546	18	card	Latte	Night	Sun	

3547 rows × 11 columns

In [7]:
```
df[df.duplicated()]
```

Out[7]:

hour_of_day	cash_type	coffee_name	Time_of_Day	Weekday	Month_name

In [8]:

```
df.isnull().any()
```

Out[8]:

```
hour_of_day    False
cash_type      False
coffee_name    False
Time_of_Day    False
Weekday        False
Month_name     False
Weekdaysort    False
Monthsort      False
Date           False
Time           False
money          False
dtype: bool
```

In [9]:

```
df["money"] = pd.to_numeric(df["money"], errors="coerce")
```

In [10]:

```
weekday_order = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
month_order = [
    "Jan",
    "Feb",
    "Mar",
    "Apr",
    "May",
    "Jun",
    "Jul",
    "Aug",
    "Sep",
    "Oct",
    "Nov",
    "Dec",
]

df["Weekday"] = pd.Categorical(df["Weekday"], categories=weekday_order
df["Month_name"] = pd.Categorical(
    df["Month_name"], categories=month_order, ordered=True
)
```

In [11]:

```
df.head()
```

Out[11]:

	hour_of_day	cash_type	coffee_name	Time_of_Day	Weekday	Month_name
0	10	card	Latte	Morning	Fri	Mar
1	12	card	Hot Chocolate	Afternoon	Fri	Mar
2	12	card	Hot Chocolate	Afternoon	Fri	Mar
3	13	card	Americano	Afternoon	Fri	Mar
4	13	card	Latte	Afternoon	Fri	Mar

Question 1

What times of day and days of the week generate the highest sales volume, and how can staffing or store hours be optimized to match customer demand?

Purpose: Helps identify peak operational periods to guide shift scheduling and labor cost efficiency.

In [12]:

```
# general aggregations
sales_by_hour = (
    df.groupby("hour_of_day", as_index=False)["money"].sum().sort_valu
)
count_by_hour = (
    df.groupby("hour_of_day", as_index=False)
    .size()
    .rename(columns={"size": "transactions"})
)
sales_by_timeofday = (
    df.groupby("Time_of_Day", as_index=False)["money"]
    .sum()
    .sort_values("money", ascending=False)
)
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)
)
```

```
/var/folders/mc/2wjfdchj6vsffbrpfbfgqw4w0000gn/T/ipykernel_70053/106961
8463.py:16: FutureWarning: The default of observed=False is deprecated
and will be changed to True in a future version of pandas. Pass observe
d=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  df.groupby(["Weekday", "Weekdaysort"], as_index=False)["money"]
```

In [13]:

```
pivot_df = df.pivot_table(
    index="hour_of_day", columns="Weekday", values="money", aggfunc="s
)
```

```
/var/folders/mc/2wjfdchj6vsffbrpfbfgqw4w0000gn/T/ipykernel_70053/177229
2445.py:1: FutureWarning: The default value of observed=False is deprec
ated and will change to observed=True in a future version of pandas. Sp
ecify observed=False to silence this warning and retain the current beh
avior
  pivot_df = df.pivot_table(
```

In [14]:

```
pivot_df
```

Out[14]:

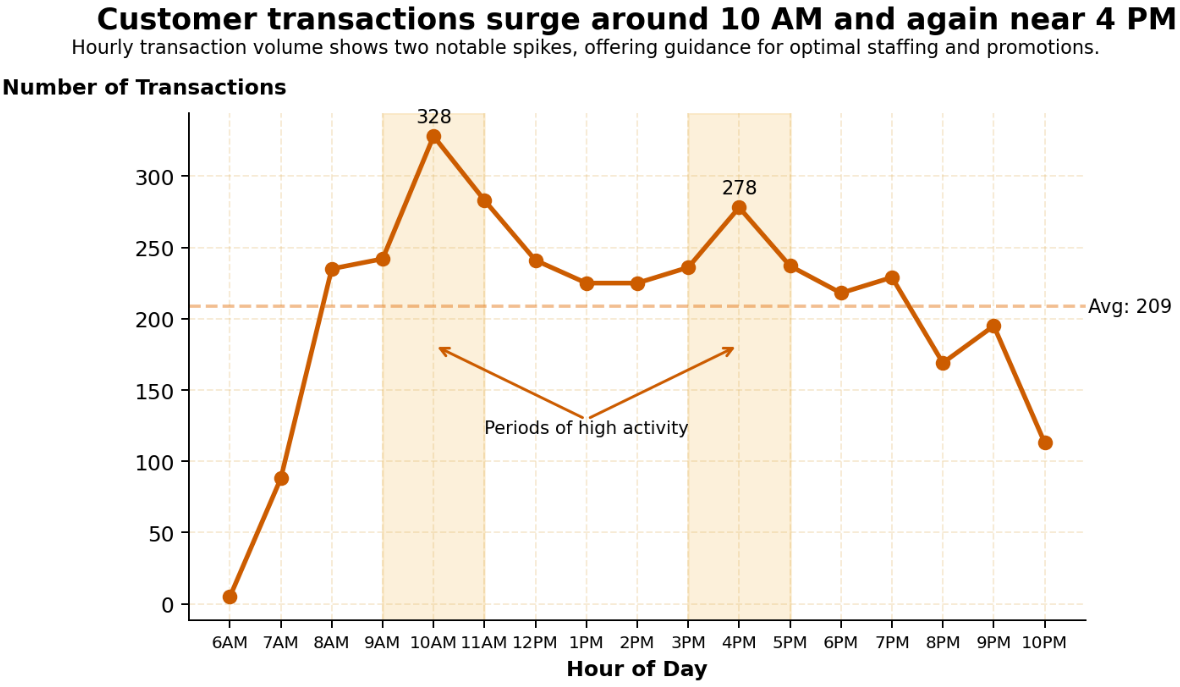
Weekday	Mon	Tue	Wed	Thu	Fri	Sat	Sun
hour_of_day							
6	9.70	0.00	0.00	0.00	13.35	0.00	0.00
7	105.45	69.50	84.45	63.45	82.85	9.70	8.90
8	188.25	220.85	131.65	102.20	216.55	107.10	88.70
9	165.80	175.75	117.25	146.70	225.15	149.15	95.05
10	232.90	213.95	211.10	217.55	189.35	183.65	228.05
11	173.25	247.50	203.50	125.65	124.10	242.35	130.50
12	112.15	134.75	149.05	129.25	163.60	207.30	188.75
13	124.95	99.45	195.35	138.75	163.70	147.45	148.20
14	190.90	147.60	68.45	150.85	148.35	169.45	142.85
15	170.15	119.45	141.05	163.90	158.45	168.30	159.35
16	218.65	211.10	194.30	177.55	139.05	193.65	111.55
17	160.55	157.25	156.70	162.25	184.50	129.50	108.70
18	162.70	147.70	155.10	138.25	138.80	85.05	153.20
19	203.10	236.95	164.05	193.05	116.25	89.30	46.45
20	94.40	165.40	108.10	135.05	70.55	103.65	81.50
21	117.70	144.55	140.05	192.80	76.45	52.15	153.50
22	60.65	54.25	45.50	57.95	152.70	95.95	38.80

In [15]:

```
tx_hour = count_by_hour.sort_values("hour_of_day")

fig, ax_tx = plt.subplots(figsize=(8, 5), dpi=180)

main_color = "#cc5c00"
light_color = "#f5c76e"
highlight_color = "#e67e22"

ax_tx.plot(
    tx_hour["hour_of_day"],
    tx_hour["transactions"],
    marker="o",
    linewidth=2.2,
    color=main_color,
)
ax_tx.grid(True, linestyle="--", alpha=0.25, color="#e0b35c")

spacing = tx_hour["transactions"].max() * 0.02
```

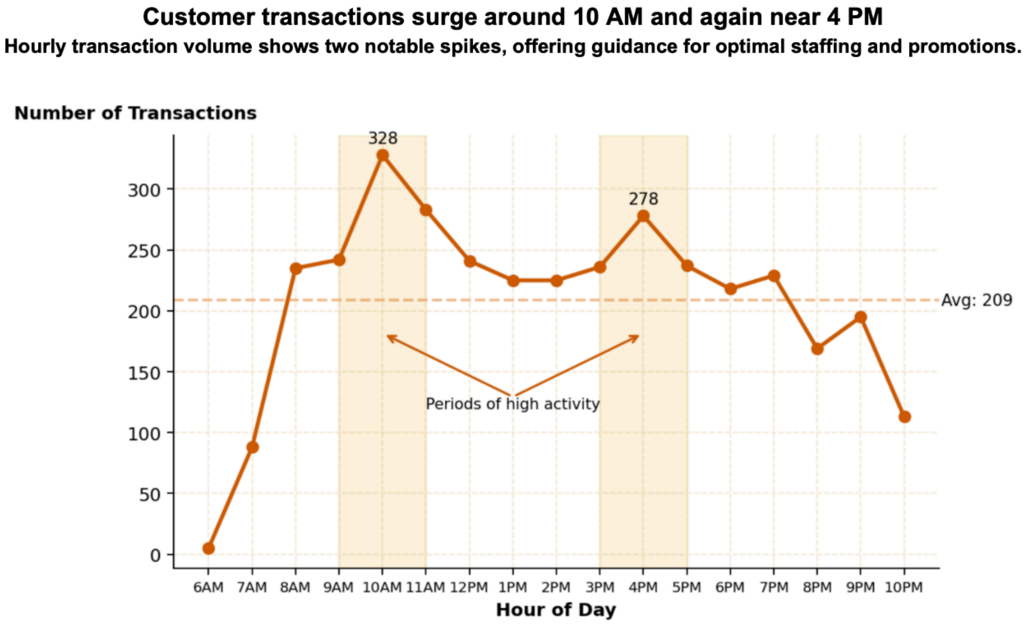
```
peak_hours = [10, 16]
for x, y in zip(tx_hour["hour_of_day"], tx_hour["transactions"]):
    if x in peak_hours:
        ax_tx.text(
            x,
            y + spacing,
            f"{y:,.0f}",
            ha="center",
            va="bottom",
            fontsize=9,
            fontweight="medium",
            color="black",
        )

plt.suptitle(
    "Hourly transaction volume shows two notable spikes, offering guid
    fontsize=9,
    fontweight="medium",
    y=0.872,
    x=0.5,
    ha="center",
    color="black",
)

plt.title(
    "Customer transactions surge around 10 AM and again near 4 PM",
    fontsize=14,
    fontweight="bold",
    pad=40,
    loc="center",
    color="black",
)

plt.xlabel("Hour of Day", fontsize=10, fontweight="bold", color="black
plt.ylabel("", fontsize=10, fontweight="bold")

ax_tx.text(
    -0.05,
    1.03,
    "Number of Transactions",
    transform=ax_tx.transAxes,
    ha="center",
    va="bottom",
    fontweight="bold",
    fontsize=10,
    color="black",
)

ax_tx.set_xticks(range(6, 23))
ax_tx.set_xticklabels(
    [f"{h%12 or 12}{'AM' if h < 12 else 'PM'}" for h in range(6, 23)],
    fontsize=8,
    fontweight="medium",
    color="black",
    family="sans-serif",
)

# get the avg line
avg_tx = tx_hour["transactions"].mean()
ax_tx.axhline(avg_tx, color=highlight_color, linestyle="--", alpha=0.5
ax_tx.text(
    x=tx_hour["hour_of_day"].max() + 0.85,
    y=avg_tx,
    s=f"Avg: {avg_tx:,.0f}",
    va="center",
    ha="left",
    fontsize=9,
    color="black",
    fontweight="medium",
)

# highlight windows
ax_tx.axvspan(9, 11, color=light_color, alpha=0.25)
ax_tx.axvspan(15, 17, color=light_color, alpha=0.25)

label_y = avg_tx * 0.62
arrow_y = avg_tx * 0.87

ax_tx.text(
    13,
    label_y,
    "Periods of high activity",
    ha="center",
    va="top",
    fontsize=8.5,
    fontweight="medium",
    color="black",
)

ax_tx.annotate(
    "",
    xy=(10, arrow_y),
    xytext=(13, label_y),
    arrowprops=dict(arrowstyle="->", lw=1.3, color=main_color),
)
ax_tx.annotate(
    "",
    xy=(16, arrow_y),
    xytext=(13, label_y),
    arrowprops=dict(arrowstyle="->", lw=1.3, color=main_color),
)

sns.despine()
plt.tight_layout()
```

```python
plt.show()
```



We need to post processes this for better labeling for which we move our plot above to a word document and then work on the labels

The is our plot post processing



In [16]:
```python
weekday_order = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
pivot_df = pivot_df[weekday_order]

left_data = pivot_df.copy()
```

```
def get_time_of_day(hour):
    if 6 <= hour < 12:
        return "Morning"
    elif 12 <= hour < 17:
        return "Afternoon"
    elif 17 <= hour <= 22:
        return "Evening"
    else:
        return "Other"


df_long = pivot_df.reset_index().melt(
    id_vars="hour_of_day", var_name="Weekday", value_name="Revenue"
)
df_long["TimeOfDay"] = df_long["hour_of_day"].apply(get_time_of_day)
agg = df_long.groupby(["Weekday", "TimeOfDay"])["Revenue"].sum().reset
time_order = ["Morning", "Afternoon", "Evening"]
right_data = (
    agg.pivot(index="Weekday", columns="TimeOfDay", values="Revenue")
    .reindex(index=weekday_order, columns=time_order)
    .fillna(0)
)
```

In [17]:

```
# prepare heatmap
td_map = {"Night": "Evening"}
df_for_heatmap = df.assign(Time_of_Day_plot=df["Time_of_Day"].replace(

right_data = (
    df_for_heatmap.pivot_table(
        index="Weekday",
        columns="Time_of_Day_plot",
        values="money",
        aggfunc="sum",
        fill_value=0,
    )
    .reindex(index=weekday_order)
    .reindex(columns=time_order, fill_value=0)
    .astype(float)
)
```

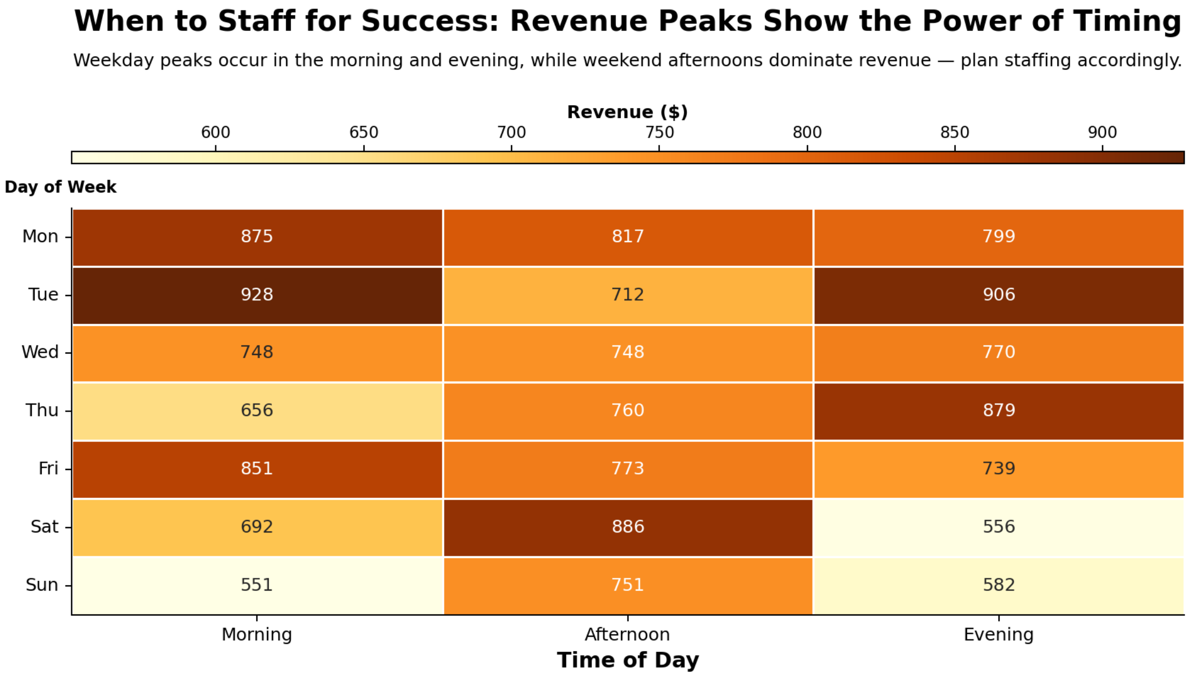
```
/var/folders/mc/2wjfdchj6vsffbrpfbfgqw4w0000gn/T/ipykernel_70053/185310
7675.py:6: FutureWarning: The default value of observed=False is deprec
ated and will change to observed=True in a future version of pandas. Sp
ecify observed=False to silence this warning and retain the current beh
avior
  df_for_heatmap.pivot_table(
```

In [18]:

```
weekday_order = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
time_order = ["Morning", "Afternoon", "Evening"]

fig, ax = plt.subplots(figsize=(10, 6), dpi=180)
```

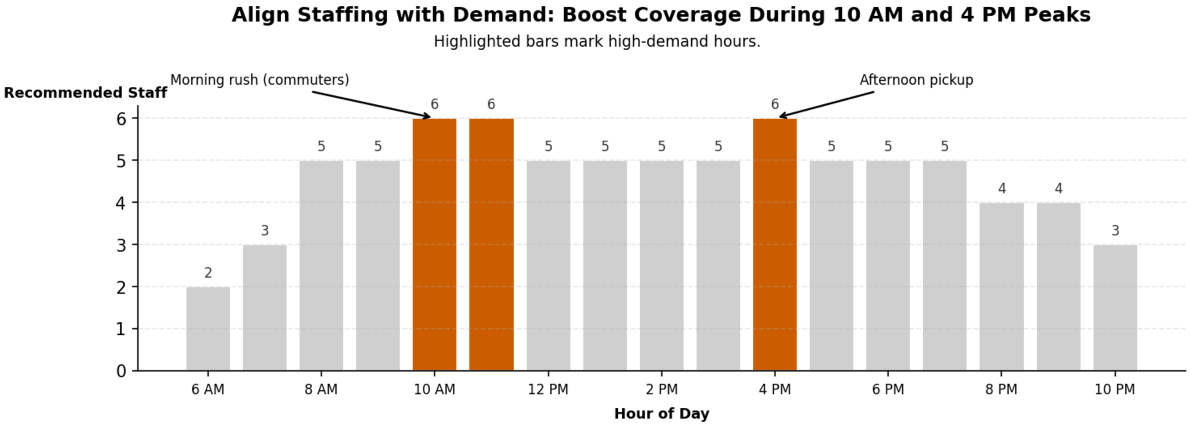
```
heatmap = sns.heatmap(
    right_data,
    ax=ax,
    cmap="YlOrBr",
    annot=True,
    fmt=".0f",
    linewidths=0.5,
    linecolor="white",
    cbar=False,
)

# horizontal colorbar axis just above the heatmap
from mpl_toolkits.axes_grid1 import make_axes_locatable

divider = make_axes_locatable(ax)
cax = divider.append_axes("top", size="3%", pad=0.35)
norm = plt.Normalize(vmin=right_data.values.min(), vmax=right_data.val
sm = plt.cm.ScalarMappable(cmap="YlOrBr", norm=norm)
cbar = fig.colorbar(sm, cax=cax, orientation="horizontal")
cbar.set_label("Revenue ($)", fontsize=10, fontweight="bold", labelpad
cbar.ax.tick_params(labelsize=9, pad=2)
cbar.ax.xaxis.set_ticks_position("top")
cbar.ax.xaxis.set_label_position("top")

ax.set_title(
    "When to Staff for Success: Revenue Peaks Show the Power of Timing
    fontsize=16,
    fontweight="bold",
    pad=100,
)
ax.text(
    0.5,
    1.35,
    "Weekday peaks occur in the morning and evening, while weekend aft
    transform=ax.transAxes,
    ha="center",
    fontsize=10,
)

ax.set_xlabel("Time of Day", fontsize=12, fontweight="bold")
ax.set_ylabel("", fontsize=10, fontweight="bold")
ax.text(
    -0.01,
    1.03,
    "Day of Week",
    transform=ax.transAxes,
    ha="center",
    va="bottom",
    fontweight="bold",
    fontsize=9,
)
```

```
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
ax.set_yticklabels(ax.get_yticklabels(), rotation=0)

sns.despine()
plt.tight_layout(rect=[0, 0.05, 1, 0.96])
plt.show()
```



In [19]:
```
hour_sales = sales_by_hour.set_index("hour_of_day")["money"]
normalized = hour_sales / hour_sales.max() if hour_sales.max() > 0 els
recommended_staff = (normalized * 5).apply(math.ceil) + 1

recommended_df = recommended_staff.reset_index().rename(
    columns={"money": "recommended_staff"}
)

rec_hours = (
    pd.DataFrame({"hour_of_day": range(6, 23)})
    .merge(recommended_df, on="hour_of_day", how="left")
    .fillna(1)
)
rec_hours["recommended_staff"] = rec_hours["recommended_staff"].astype
```

In [20]:
```
plt.figure(figsize=(10, 4), dpi=150)

non_peak_color = "#cfcfcf"
peak_color = "#cc5c00"

bars = plt.bar(
    rec_hours["hour_of_day"],
    rec_hours["recommended_staff"],
    color=non_peak_color,
    edgecolor="white",
)

for idx, val in enumerate(rec_hours["recommended_staff"]):
    plt.text(
        rec_hours["hour_of_day"].iloc[idx],
        val + 0.15,
        str(val),
        ha="center",
        va="bottom",
        fontsize=8,
        color="#333",
    )

# highlight peak bars
peak_hours = [10, 11, 16]
plt.bar(
    rec_hours.loc[rec_hours["hour_of_day"].isin(peak_hours), "hour_of_
    rec_hours.loc[rec_hours["hour_of_day"].isin(peak_hours), "recommen
    color=peak_color,
    edgecolor="white",
)

plt.title(
    "Align Staffing with Demand: Boost Coverage During 10 AM and 4 PM
    fontsize=12,
    fontweight="bold",
    pad=50,
)
plt.suptitle(
    "Highlighted bars mark high-demand hours.",
    fontsize=9,
    y=0.835,
)

plt.xlabel("Hour of Day", fontsize=8.5, fontweight="bold", labelpad=6)
plt.ylabel("")
plt.text(
    -0.05,
    1.02,
    "Recommended Staff",
    transform=plt.gca().transAxes,
    ha="center",
    va="bottom",
    fontweight="bold",
    fontsize=8.5,
)

ax = plt.gca()
ax.set_xticks(range(6, 23))
ax.set_xticklabels(
    [f"{h%12 or 12} {'AM' if h < 12 else 'PM'}" for h in range(6, 23)]
    rotation=0,
    fontsize=8,
    fontweight="medium",
    family="sans-serif",
)

plt.annotate(
    "Morning rush (commuters)",
    xy=(10, 6),
    xytext=(8.5, 6.8),
    arrowprops=dict(arrowstyle="->", color="black", lw=1.2),
    fontsize=8,
    color="black",
    ha="right",
)

plt.annotate(
    "Afternoon pickup",
    xy=(16, 6),
    xytext=(17.5, 6.8),
    arrowprops=dict(arrowstyle="->", color="black", lw=1.2),
    fontsize=8,
    color="black",
    ha="left",
)

ax.xaxis.set_major_locator(MaxNLocator(integer=True))
ax.grid(axis="y", linestyle="--", alpha=0.3)
ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)

plt.tight_layout()
plt.show()
```



In [21]:

```
rec_open_hours = rec_hours[rec_hours["recommended_staff"] > 1][
    ["hour_of_day", "recommended_staff"]
]
print("Recommended Staff by Hour (hours with >1 staff):")
print(rec_open_hours)
```

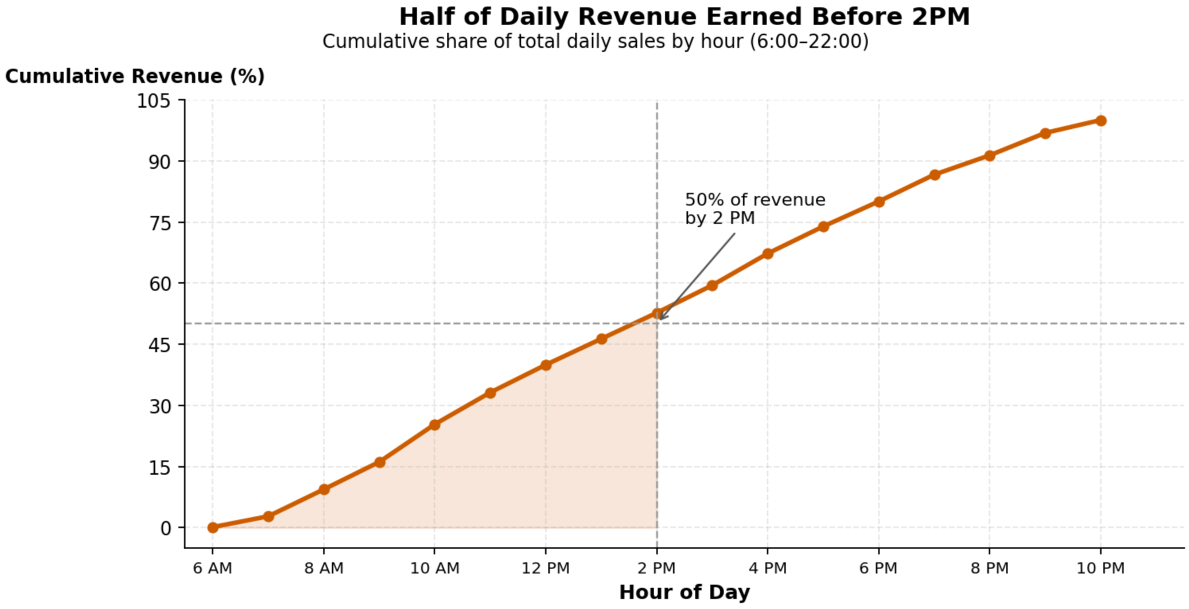
```
Recommended Staff by Hour (hours with >1 staff):
    hour_of_day  recommended_staff
0             6                  2
1             7                  3
2             8                  5
3             9                  5
4            10                  6
5            11                  6
6            12                  5
7            13                  5
8            14                  5
9            15                  5
10           16                  6
11           17                  5
12           18                  5
13           19                  5
14           20                  4
15           21                  4
16           22                  3
```

Question 2

Which types of coffee are most popular during different times of the day, and how should inventory and promotions adjust accordingly? Purpose: Helps managers plan inventory and marketing by matching coffee types to customer habits (e.g., cappuccinos in the morning, iced drinks in the afternoon).

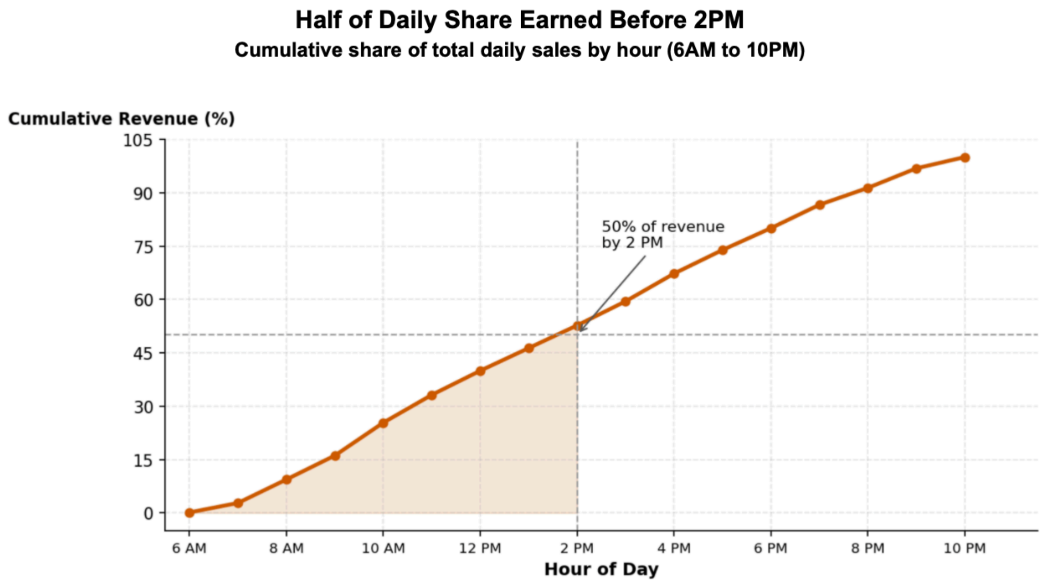
In [22]:

```
assert {"hour_of_day", "money"} <= set(df.columns), "Missing columns."

OPEN_HOUR = 6
CLOSE_HOUR = 23

h = (
    df.assign(hour_of_day=pd.to_numeric(df["hour_of_day"], errors="coe
    .dropna(subset=["hour_of_day", "money"])
    .query("@OPEN_HOUR <= hour_of_day <= @CLOSE_HOUR")
    .groupby("hour_of_day", dropna=True)["money"]
    .sum()
    .sort_index()
)
if h.empty:
    print("No hourly data to plot within open hours.")
else:
    cum_pct = h.cumsum() / h.sum() * 100
    half_idx = (cum_pct >= 50).idxmax()

    plt.figure(figsize=(9, 5), dpi=170)
    ax = plt.gca()
    plt.plot(
        cum_pct.index,
        cum_pct.values,
        color="#cc5c00",
        lw=2.4,
        marker="o",
        markersize=5,
    )

    plt.fill_between(
        cum_pct.index,
        0,
        cum_pct.values,
        where=cum_pct.index <= half_idx,
        color="#cc5c00",
        alpha=0.15,
    )

    plt.axhline(50, ls="--", c="#999", lw=1)
    plt.axvline(half_idx, ls="--", c="#999", lw=1)

    hour_12 = half_idx % 12
    hour_12 = 12 if hour_12 == 0 else hour_12
    period = "AM" if half_idx < 12 else "PM"

    plt.annotate(
        f"50% of revenue\nby {hour_12} {period}",
        xy=(half_idx, 50),
        xytext=(half_idx + 0.5, 78),
        arrowprops=dict(arrowstyle="->", color="#555", lw=1),
        fontsize=9.5,
        ha="left",
        va="center",
    )

    plt.title(
        "Half of Daily Revenue Earned Before 2PM",
        fontsize=13,
        fontweight="bold",
        pad=40,
    )
    plt.suptitle(
        f"Cumulative share of total daily sales by hour ({OPEN_HOUR}:0
        fontsize=10,
        y=0.873,
    )

    ax.set_xlabel("Hour of Day", fontsize=10.5, fontweight="bold")

    ax.text(
        -0.05,
        1.03,
        "Cumulative Revenue (%)",
        transform=ax.transAxes,
        ha="center",
        va="bottom",
        fontweight="bold",
        fontsize=10,
    )

    ax.set_xlim(OPEN_HOUR - 0.5, CLOSE_HOUR + 0.5)
    ax.set_xticks(range(OPEN_HOUR, CLOSE_HOUR + 1, 2))
    ax.set_xticklabels(
        [
            f"{h%12 or 12} {'AM' if h < 12 else 'PM'}"
            for h in range(OPEN_HOUR, CLOSE_HOUR + 1, 2)
        ],
        fontsize=8.5,
    )

    ax.yaxis.set_major_locator(MaxNLocator(integer=True))
    ax.grid(axis="both", linestyle="--", alpha=0.3)
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)

    plt.tight_layout()
    plt.show()
```



We need to post processes this for better labeling for which we move our plot above to a word document and then work on the labels

The is our plot post processing



In [23]:

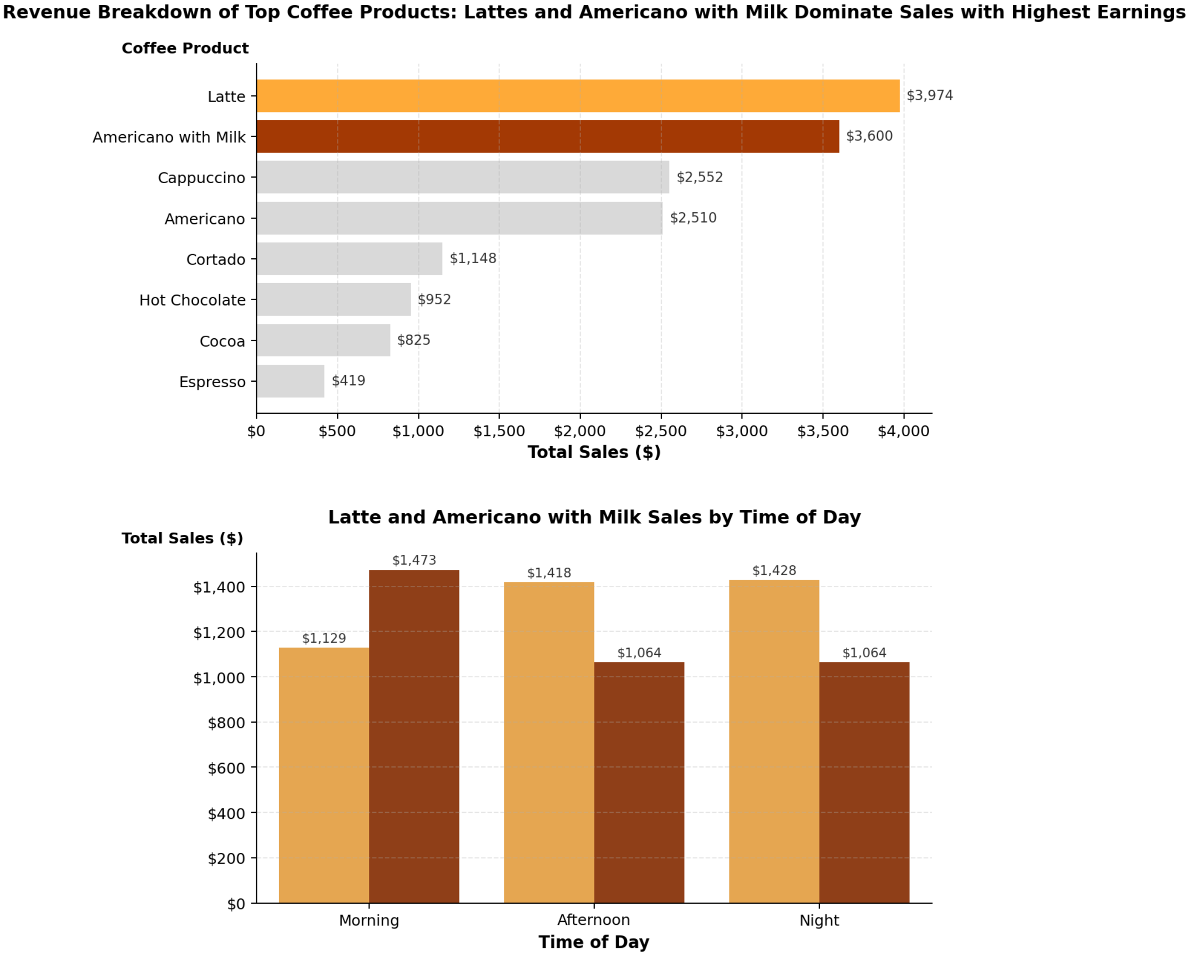
```
DAYPART_ORDER = ["Morning", "Afternoon", "Night"]
DAYPART_DESC = {"Morning": "Open–10am", "Afternoon": "10am–2pm", "Nigh
TOP_N = 8
CMAP = "Blues"
CURRENCY = FuncFormatter(lambda x, p: f"${x:,.0f}")

sales_by_coffee_ord = sales_by_coffee.sort_values("money", ascending=T
overall_top = sales_by_coffee_ord.iloc[-1]["coffee_name"]
overall_top_amt = sales_by_coffee_ord.iloc[-1]["money"]

coffee_heatmap = (
    df[df["coffee_name"].isin(sales_by_coffee_ord["coffee_name"])]
    .pivot_table(
        index="coffee_name",
        columns="Time_of_Day",
        values="money",
        aggfunc="sum",
        fill_value=0,
    )
    .reindex(index=sales_by_coffee_ord["coffee_name"].tolist())
    .reindex(columns=[d for d in DAYPART_ORDER if d in df["Time_of_Day
)

daypart_leads = coffee_heatmap.idxmax(axis=0)
lead_morn = daypart_leads.get("Morning", None)
lead_aft = daypart_leads.get("Afternoon", None)
lead_night = daypart_leads.get("Night", None)
```

In [24]:

```
cmap = plt.cm.YlOrBr
latte_color = cmap(0.45)
americano_color = cmap(0.85)

fig, (ax1, ax2) = plt.subplots(
    2, 1, figsize=(8, 10), dpi=180, gridspec_kw={"hspace": 0.4}
)

y = np.arange(len(sales_by_coffee_ord))
colors = []
for name in sales_by_coffee_ord["coffee_name"]:
    if name == "Latte":
        colors.append(latte_color)
    elif name == "Americano with Milk":
        colors.append(americano_color)
    else:
        colors.append("#D9D9D9")

bars = ax1.barh(y, sales_by_coffee_ord["money"].values, color=colors)

xmax = sales_by_coffee_ord["money"].max()
for yi, v in zip(y, sales_by_coffee_ord["money"].values):
    ax1.text(v + xmax * 0.01, yi, f"${v:,.0f}", va="center", fontsize=

ax1.set_yticks(y)
ax1.set_yticklabels(sales_by_coffee_ord["coffee_name"])
ax1.set_xlabel("Total Sales ($)", fontsize=11, fontweight="bold")
ax1.set_ylabel("")
ax1.xaxis.set_major_formatter(CURRENCY)
ax1.grid(axis="x", linestyle="--", alpha=0.3)
ax1.spines["top"].set_visible(False)
ax1.spines["right"].set_visible(False)

ax1.set_title(
    "Revenue Breakdown of Top Coffee Products: Lattes and Americano wi
    fontsize=12,
    fontweight="bold",
    pad=30,
)
ax1.text(
    -0.2,
    1.02,
    "Coffee Product",
    transform=ax1.transAxes,
    fontsize=10,
    ha="left",
    va="bottom",
    fontweight="bold",
)

focus_products = ["Latte", "Americano with Milk"]
focus_df = coffee_heatmap.loc[focus_products]

focus_long = focus_df.reset_index().melt(
    id_vars="coffee_name", var_name="Time of Day", value_name="Sales (
)
focus_long.rename(columns={"coffee_name": "Coffee"}, inplace=True)
```

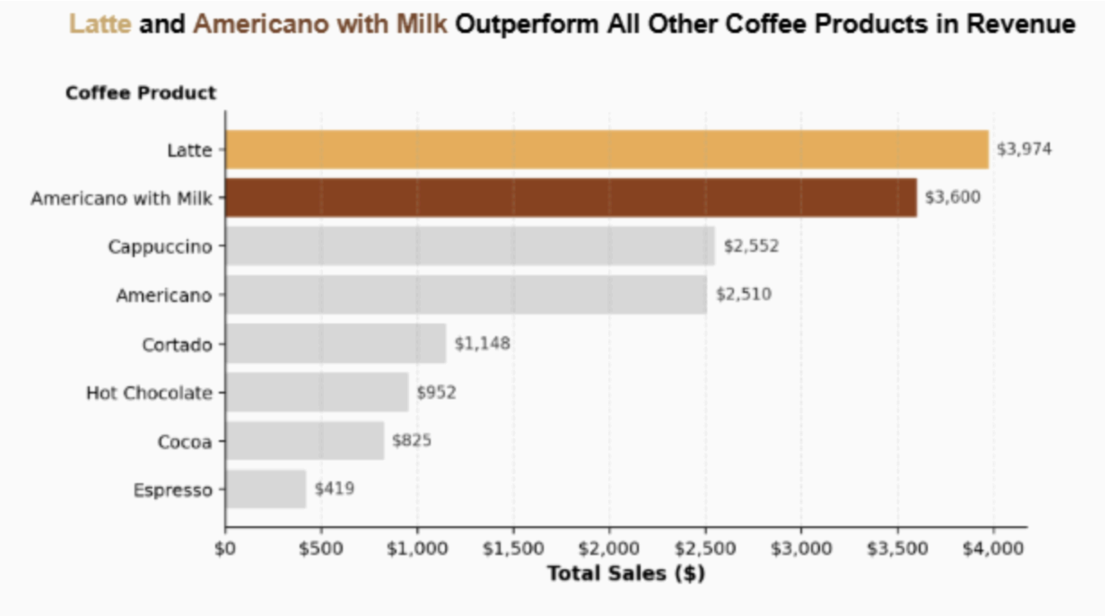
```
sns.barplot(
    data=focus_long,
    x="Time of Day",
    y="Sales ($)",
    hue="Coffee",
    ax=ax2,
    palette=[latte_color, americano_color],
)

for container in ax2.containers:
    ax2.bar_label(
        container,
        labels=[f"${h.get_height():,.0f}" for h in container],
        fmt="%d",
        label_type="edge",
        padding=2,
        fontsize=8.5,
        color="#333",
    )

ax2.set_title(
    "Latte and Americano with Milk Sales by Time of Day",
    fontsize=12,
    fontweight="bold",
    pad=20,
)
ax2.text(
    -0.2,
    1.02,
    "Total Sales ($)",
    transform=ax2.transAxes,
    fontsize=10,
    ha="left",
    va="bottom",
    fontweight="bold",
)

ax2.set_xlabel("Time of Day", fontsize=11, fontweight="bold")
ax2.set_ylabel("")
ax2.yaxis.set_major_formatter(CURRENCY)
ax2.grid(axis="y", linestyle="--", alpha=0.3)
ax2.spines["top"].set_visible(False)
ax2.spines["right"].set_visible(False)

ax2.legend_.remove()

fig.tight_layout(rect=[0, 0, 1, 0.97], pad=2.0)
plt.show()
```

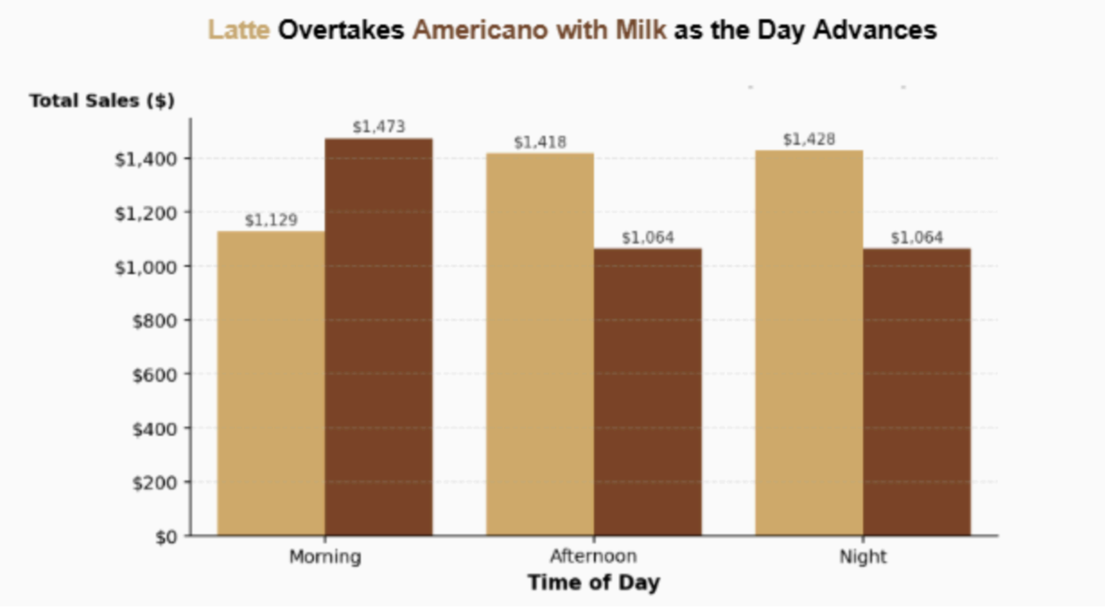
```
/var/folders/mc/2wjfdchj6vsffbrpfbfgqw4w0000gn/T/ipykernel_70053/346913
3764.py:105: UserWarning: This figure includes Axes that are not compat
ible with tight_layout, so results might be incorrect.
  fig.tight_layout(rect=[0, 0, 1, 0.97], pad=2.0)
```



We need to post processes this for better labeling for which we move our facetted plot above to a word document and then work on the labels

The is our plot post processing


Latte and Americano with Milk Outperform All Other Coffee Products in Revenue
Coffee Product
Latte
Americano with Milk
Cappuccino
Americano
Cortado
Hot Chocolate
Cocoa
Espresso
$3,974
$3,600
$2,552
$2,510
$1,148
$952
$825
$419
$0
$500
$1,000
$1,500
$2,000
$2,500
$3,000
$3,500
$4,000
Total Sales ($)


Latte Overtakes Americano with Milk as the Day Advances
Total Sales ($)
$1,400
$1,200
$1,000
$800
$600
$400
$200
$0
$1,129
$1,473
$1,418
$1,064
$1,428
$1,064
Morning
Afternoon
Night
Time of Day