🎨 AI Financial Market — Colorful EDA (v3)

This version **fixes the `NameError`** (datetime inference) and cleans up the event/company analysis.

- **Source:** `ai_financial_market_daily_realistic_synthetic.csv`
- Run all cells to generate a large set of charts. Images save under `figures/`.

In [12]:
```
# --- Setup ---
import os, math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12, 5)
plt.rcParams['axes.grid'] = True
plt.rcParams['figure.dpi'] = 140

PALETTE = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
FIGDIR = "figures"
os.makedirs(FIGDIR, exist_ok=True)

CSV_PATH = r"ai_financial_market_daily_realistic_synthetic.csv"

def infer_datetime_col(frame: pd.DataFrame):
    candidates = [c for c in frame.columns if c.lower() in ["date","time","t
    if frame.columns.size > 0:
        candidates += [frame.columns[0]]
    for col in candidates:
        try:
            pd.to_datetime(frame[col])
            return col
        except Exception:
            continue
    return None
```

In [13]:
```
# --- Load data & set index ---
df = pd.read_csv(CSV_PATH)
print("Shape:", df.shape)
print("Columns:", list(df.columns))

date_col = infer_datetime_col(df)
if date_col is not None:
    df[date_col] = pd.to_datetime(df[date_col])
    df = df.sort_values(by=date_col).reset_index(drop=True)
    df = df.set_index(date_col)
    print("Using datetime index:", date_col)
else:
    print("No datetime column detected; using integer index.")
```

```
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
companies = sorted(df["Company"].dropna().unique().tolist()) if "Company" ir
print("Numeric columns:", num_cols)
df.head(5)
```

```
Shape: (10959, 7)
Columns: ['Date', 'Company', 'R&D_Spending_USD_Mn', 'AI_Revenue_USD_Mn', 'AI
_Revenue_Growth_%', 'Event', 'Stock_Impact_%']
Using datetime index: Date
Numeric columns: ['R&D_Spending_USD_Mn', 'AI_Revenue_USD_Mn', 'AI_Revenue_Gr
owth_%', 'Stock_Impact_%']
```

Out[13]:

Date	Company	R&D_Spending_USD_Mn	AI_Revenue_USD_Mn	AI_Revenue_Gr
2015-01-01	OpenAI	5.92	0.63	
2015-01-01	Google	79.89	30.19	
2015-01-01	Meta	50.39	18.95	
2015-01-02	OpenAI	5.41	1.81	
2015-01-02	Google	78.99	30.44	

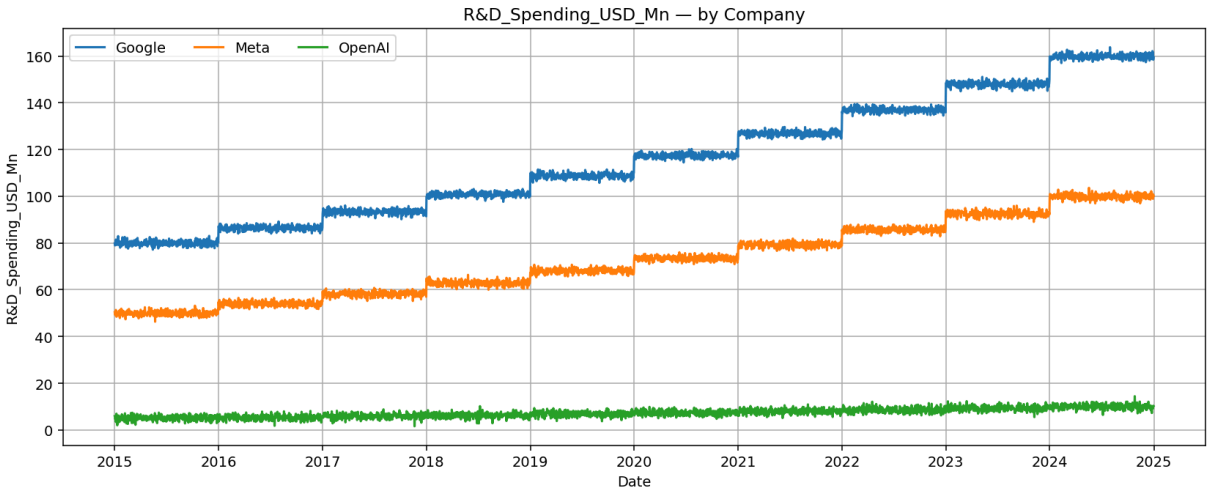
In [14]:

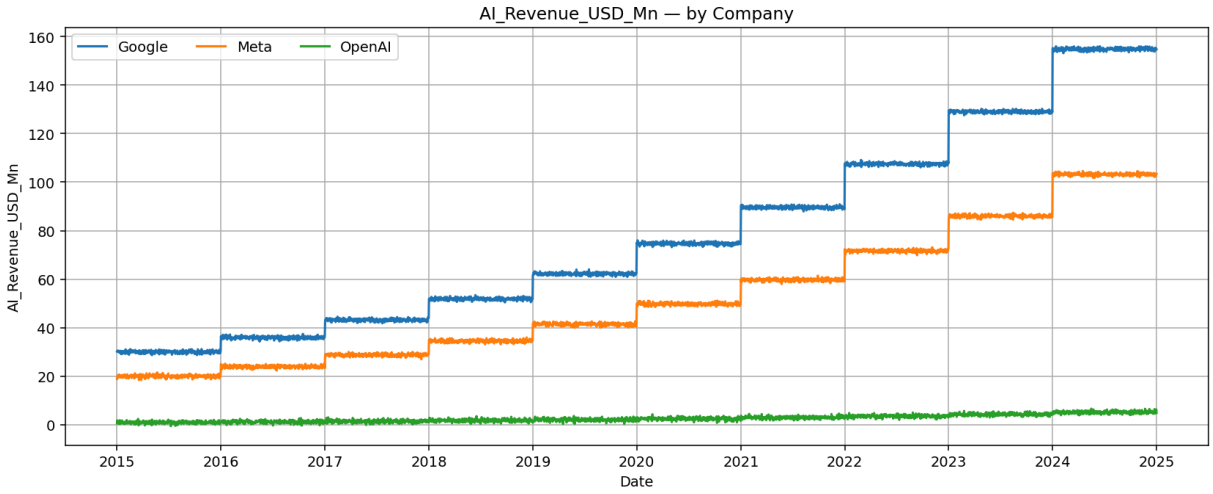
```
# --- Descriptive statistics ---
desc = df.describe(include='all')
display(desc)
desc.to_csv(os.path.join(FIGDIR, "summary_stats.csv"))
print("Saved: figures/summary_stats.csv")
```

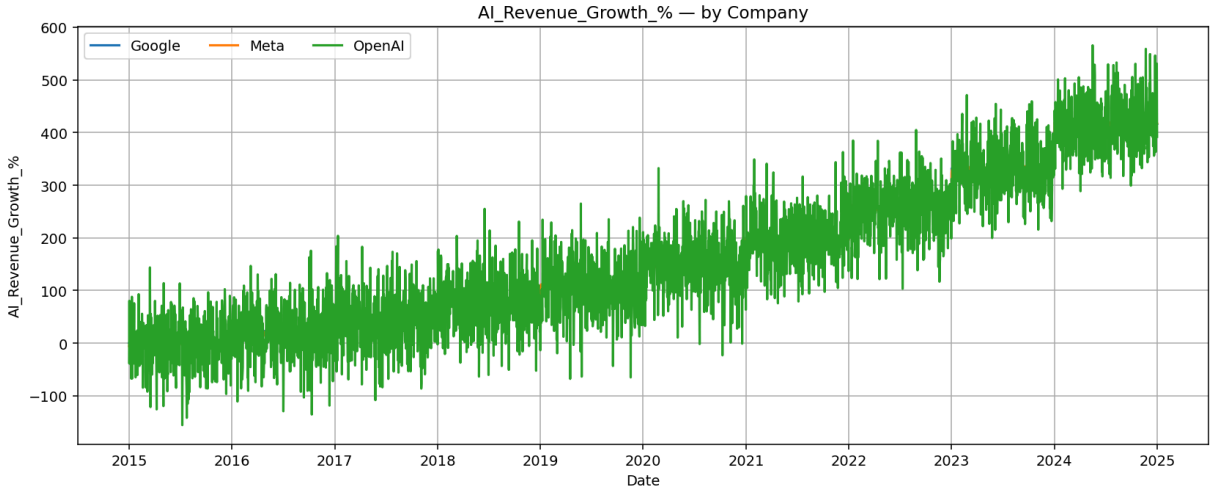
	Company	R&D_Spending_USD_Mn	AI_Revenue_USD_Mn	AI_Revenue_Gr
count	10959	10959.000000	10959.000000	10959
unique	3	NaN	NaN	
top	OpenAI	NaN	NaN	
freq	3653	NaN	NaN	
mean	NaN	65.184504	44.126571	159
std	NaN	47.918247	41.639356	135
min	NaN	1.570000	-0.550000	-155
25%	NaN	8.640000	3.610000	43
50%	NaN	70.960000	35.220000	133
75%	NaN	99.600000	71.680000	258
max	NaN	163.830000	155.960000	565

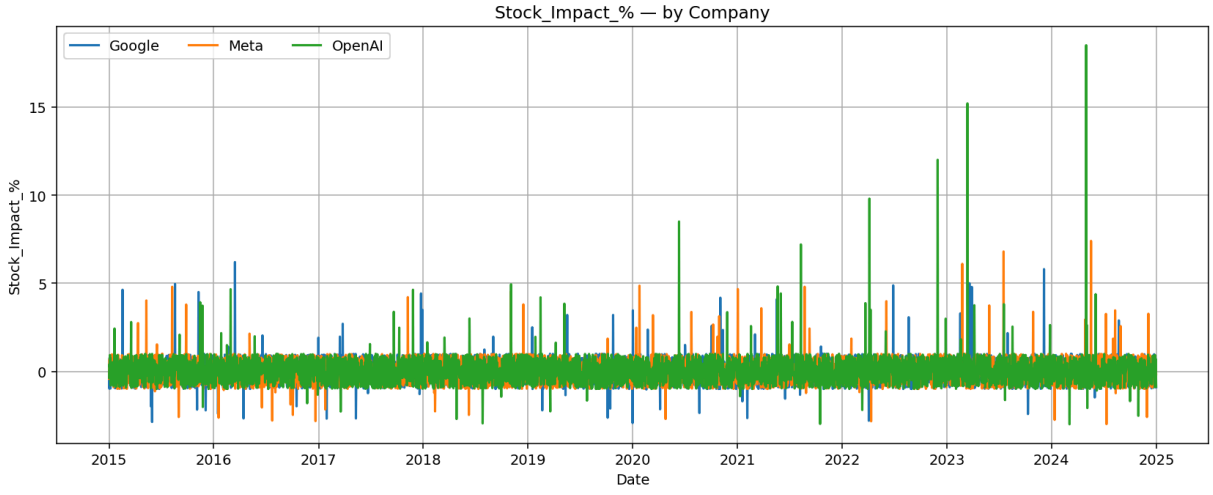
Saved: figures/summary_stats.csv

In [15]:
```
# --- Company-level overlays for numeric metrics ---
for m in num_cols:
    plt.figure()
    for i, comp in enumerate(companies):
        sub = df[df["Company"] == comp]
        if sub.empty:
            continue
        plt.plot(sub.index, sub[m], label=comp, linewidth=1.6, color=PALETTE
    plt.title(f"{m} — by Company")
    plt.xlabel("Date"); plt.ylabel(m)
    if companies: plt.legend(ncol=min(3, len(companies)))
    plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, f"{m}_by_company.pr
```



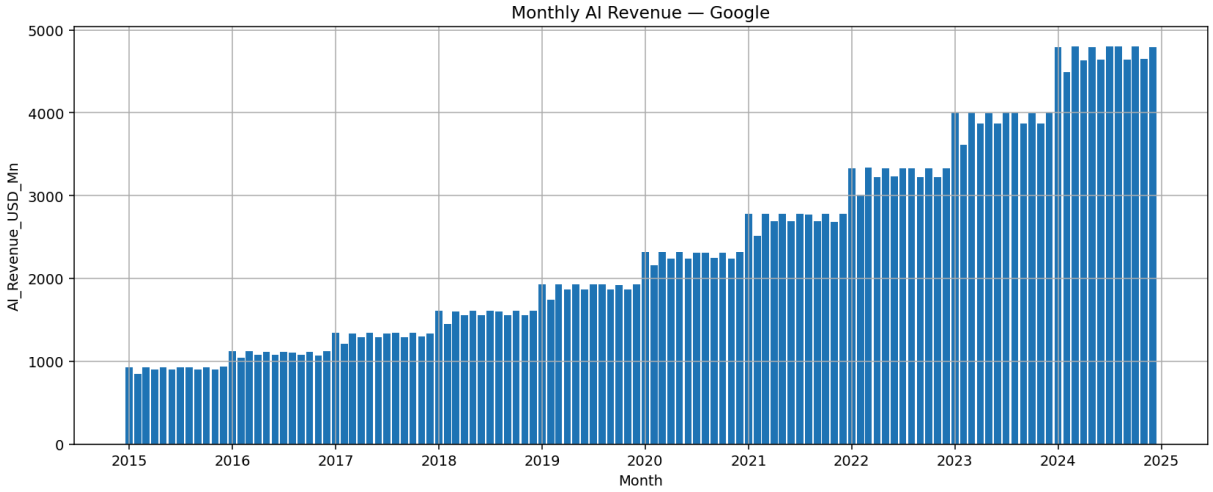


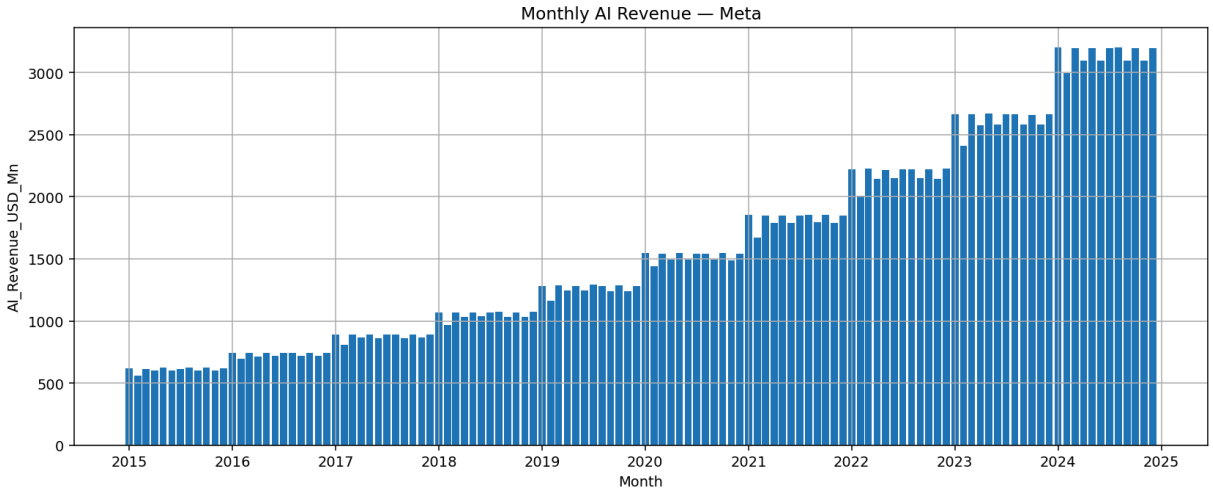


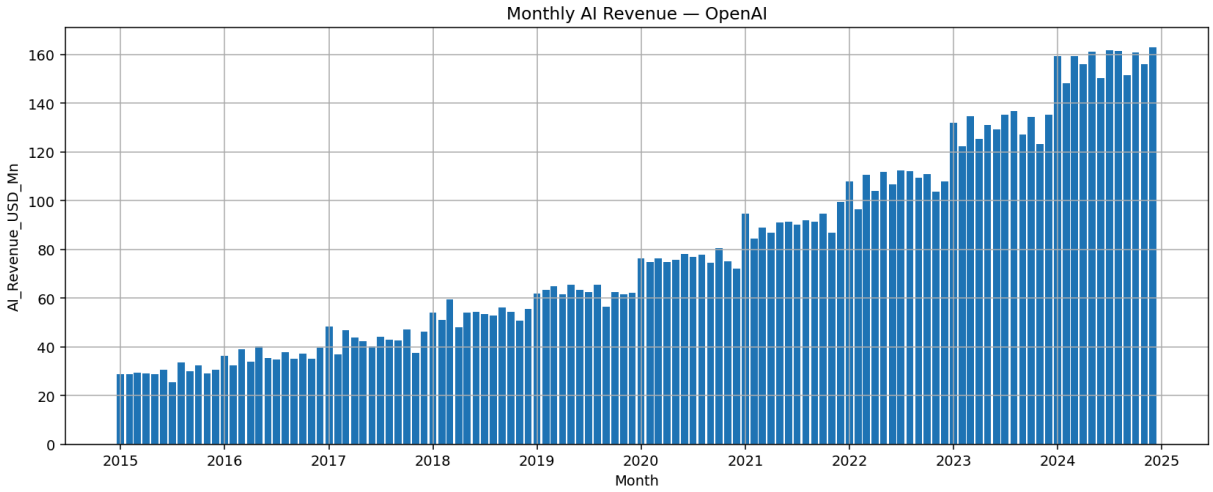


In [16]:
```python
# --- Monthly AI Revenue by company (bar) ---
if "AI_Revenue_USD_Mn" in df.columns and hasattr(df.index, "to_period"):
    monthly = df.copy()
    monthly["month"] = monthly.index.to_period("M").to_timestamp()
    g = monthly.groupby(["month","Company"])["AI_Revenue_USD_Mn"].sum().unst
    for comp in g.columns:
        plt.figure()
        plt.bar(g.index, g[comp], width=25, label=comp)
        plt.title(f"Monthly AI Revenue — {comp}")
```

```
        plt.xlabel("Month"); plt.ylabel("AI_Revenue_USD_Mn")
        plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, f"monthly_rever
```

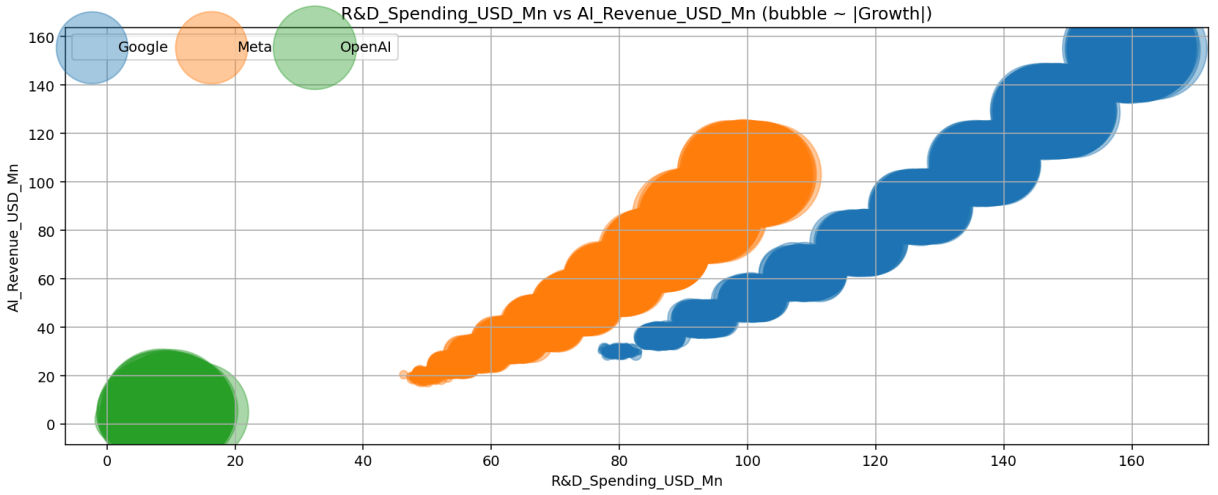






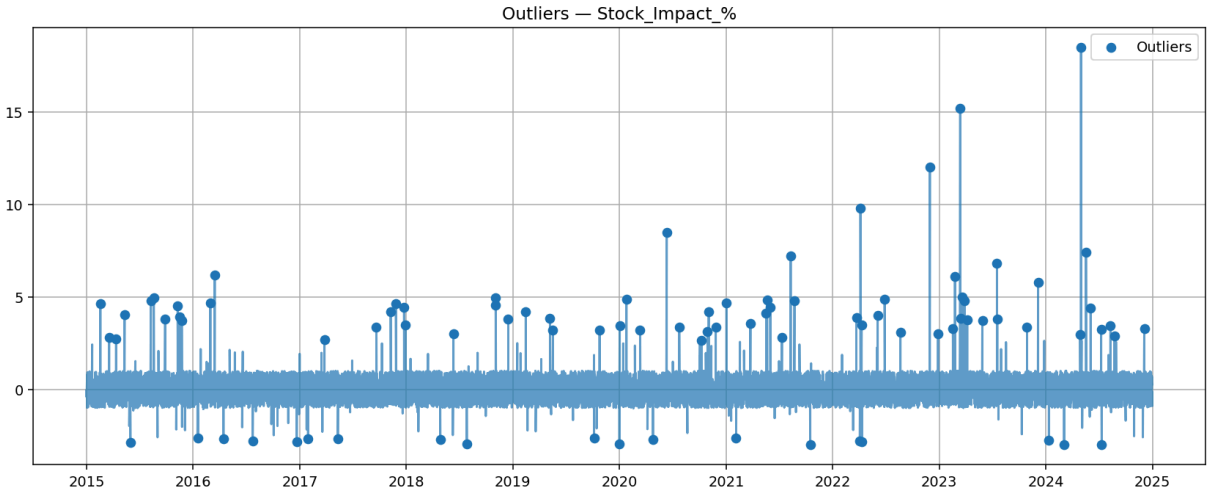
In [17]:
```
# --- R&D vs AI Revenue scatter (bubble ~ |Growth|) ---
xcol, ycol = "R&D_Spending_USD_Mn", "AI_Revenue_USD_Mn"
if xcol in df.columns and ycol in df.columns:
    plt.figure()
    for i, comp in enumerate(companies):
        sub = df[df["Company"] == comp]
        sizes = 12*sub.get("AI_Revenue_Growth_%", pd.Series(0, index=sub.inc
```

```
        plt.scatter(sub[xcol], sub[ycol], alpha=0.4, label=comp, s=sizes, cc
    plt.title(f"{xcol} vs {ycol} (bubble ~ |Growth|)")
    plt.xlabel(xcol); plt.ylabel(ycol)
    if companies: plt.legend(ncol=min(3, len(companies)))
    plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, "scatter_randd_vs_r
```



In [18]:

```
# --- Outlier detection (z>3.5) using positional indices to avoid x/y mismat
from collections import defaultdict
outliers = defaultdict(list)

for c in num_cols:
    x = pd.to_numeric(df[c], errors="coerce").astype(float)
    mu, sd = np.nanmean(x), np.nanstd(x)
    if not np.isfinite(sd) or sd == 0:
        continue
    z = (x - mu) / sd
    mask = np.abs(z) > 3.5
    pos = np.flatnonzero(mask.to_numpy() if hasattr(mask, "to_numpy") else m
    if pos.size > 0:
        outliers[c] = pos.tolist()
        plt.figure()
        x_axis = df.index
        plt.plot(x_axis, x, alpha=0.7)
        plt.scatter(x_axis[pos], x.iloc[pos], label="Outliers", zorder=5)
        plt.title(f"Outliers — {c}")
        plt.legend()
        plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, f"outliers_{c}.

print("Outlier counts by column:", {k: len(v) for k,v in outliers.items()})
```

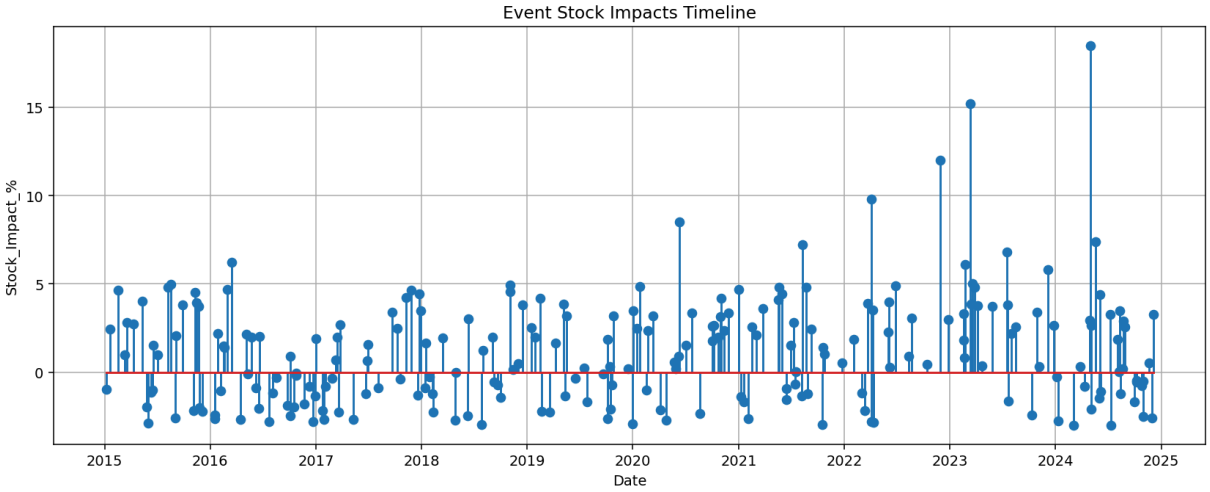


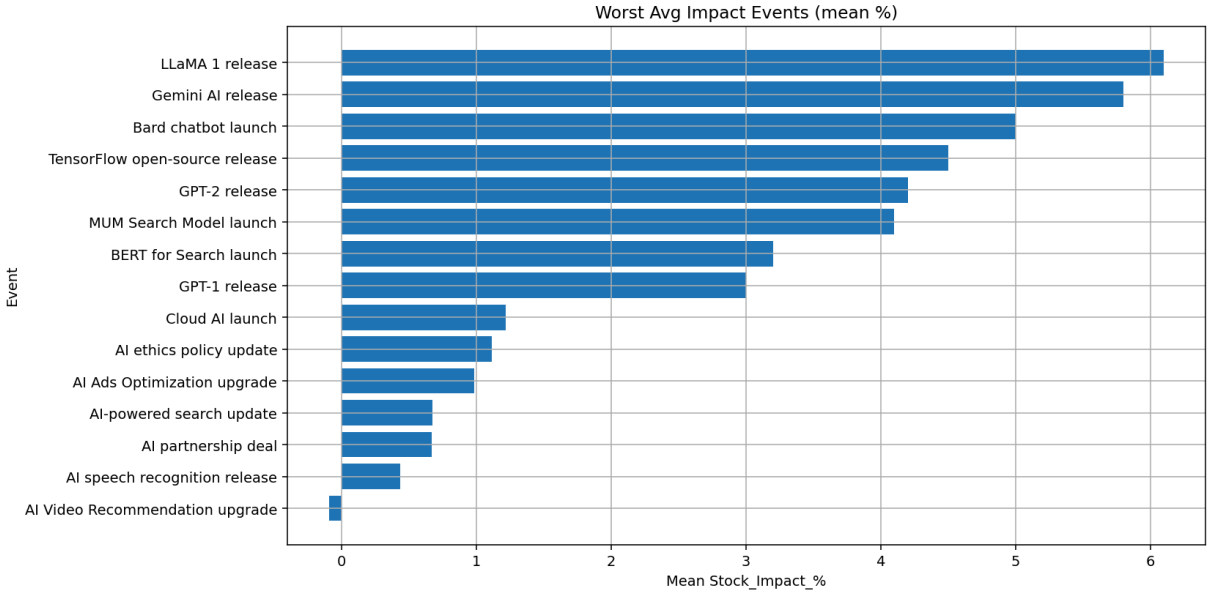

```
Outlier counts by column: {'Stock_Impact_%': 90}
```

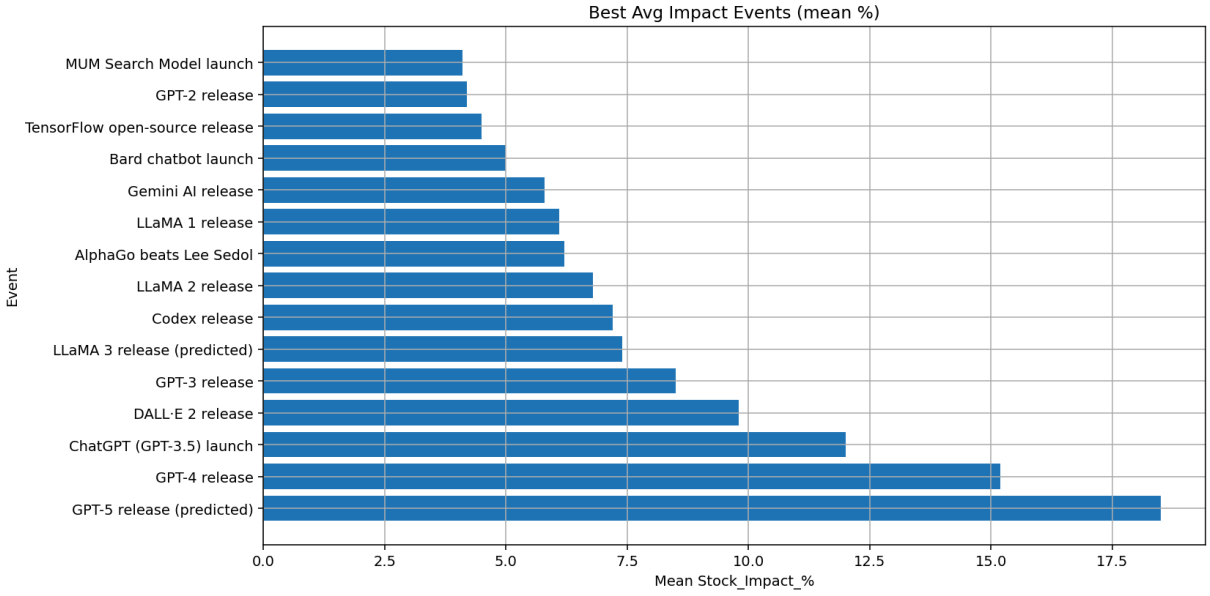
In [19]:

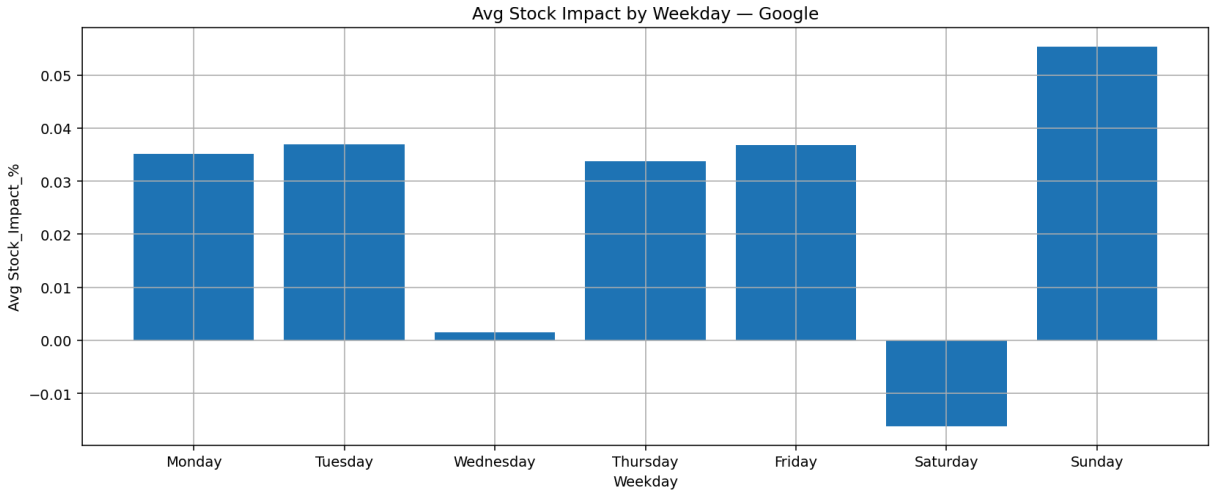
```python
# --- Event analysis (timeline & top events) ---
if "Event" in df.columns and "Stock_Impact_%" in df.columns:
    ev = df.dropna(subset=["Event"]).copy()
    if not ev.empty:
        # Timeline
        plt.figure()
        dates = ev.index
        impacts = pd.to_numeric(ev["Stock_Impact_%"], errors="coerce")
        markerline, stemlines, baseline = plt.stem(dates, impacts)
        plt.setp(markerline, markersize=6)
        plt.title("Event Stock Impacts Timeline")
        plt.xlabel("Date"); plt.ylabel("Stock_Impact_%")
        plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, "events_timelir

        # Top ± events
        agg = ev.groupby("Event")["Stock_Impact_%"].agg(["count","mean"]).sc
        worst = agg.head(15)
        best = agg.tail(15).iloc[::-1]

        for title, block, fname in [("Worst Avg Impact Events", worst, "ever
                                    ("Best Avg Impact Events", best, "events
            plt.figure(figsize=(12, 6))
            plt.barh(block.index, block["mean"])
            plt.title(title + " (mean %)")
            plt.xlabel("Mean Stock_Impact_%"); plt.ylabel("Event")
            plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, fname), bbc

        # Weekday effects by company
        if "Company" in df.columns:
            tmp = df.copy()
            tmp["weekday"] = tmp.index.day_name()
            order = ["Monday","Tuesday","Wednesday","Thursday","Friday","Sat
            for comp in sorted(tmp["Company"].dropna().unique().tolist()):
                sub = tmp[tmp["Company"] == comp]
                g = sub.groupby("weekday")["Stock_Impact_%"].mean().reindex(
                plt.figure()
                plt.bar(g.index, g.values)
                plt.title(f"Avg Stock Impact by Weekday — {comp}")
```

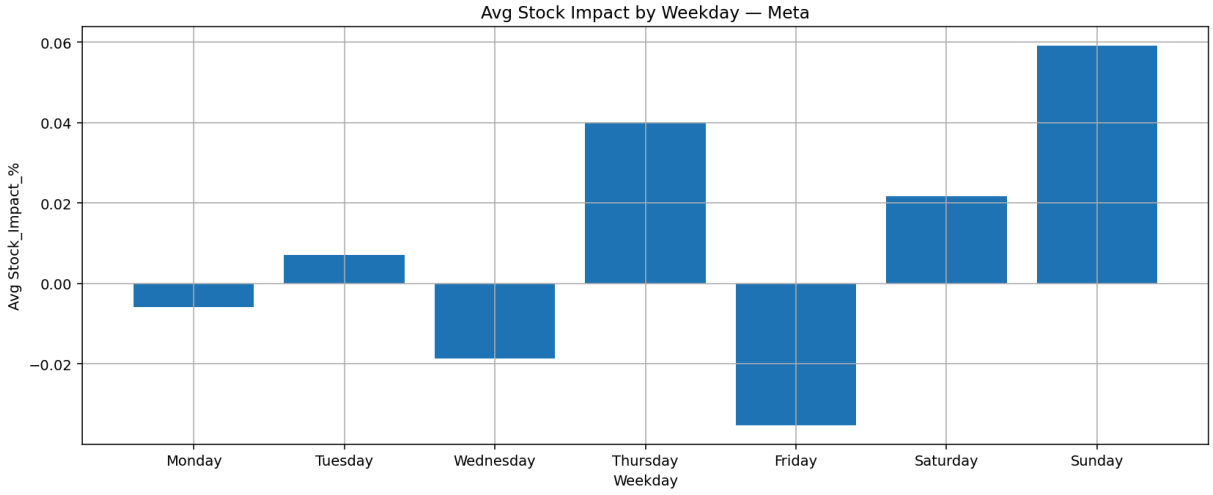
```
                plt.xlabel("Weekday"); plt.ylabel("Avg Stock_Impact_%")
                plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, f"weekd
```

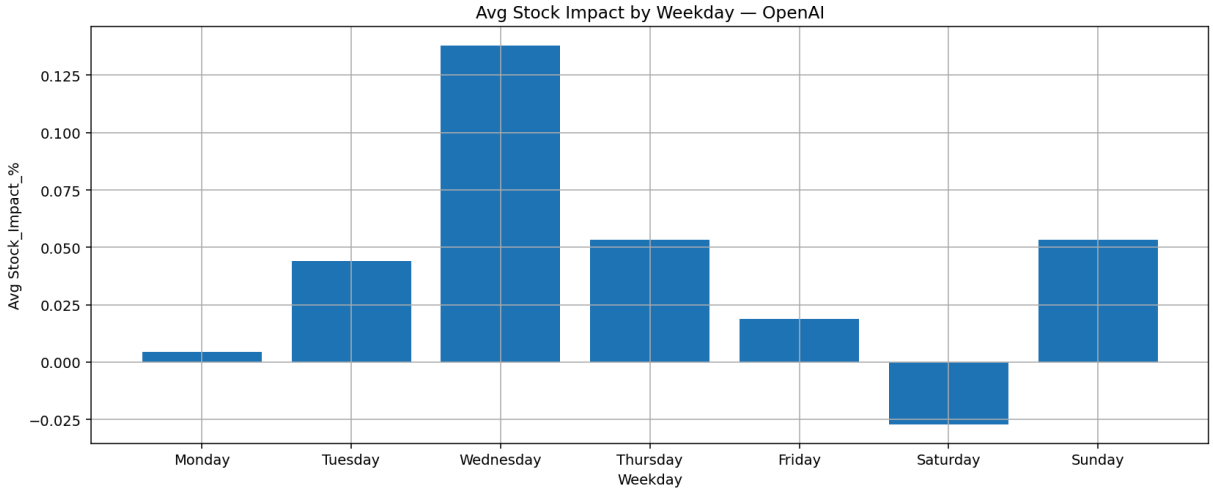






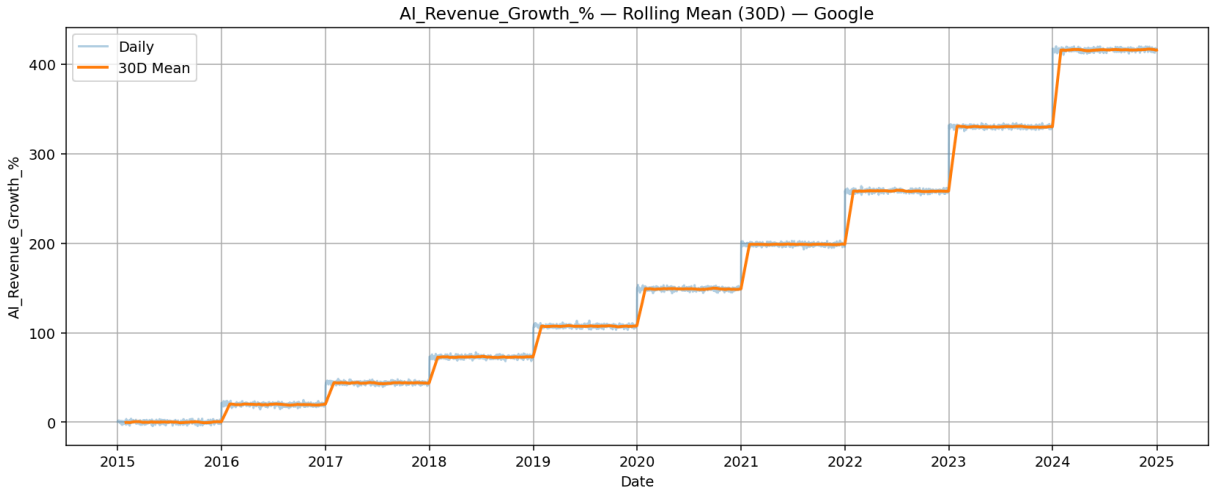


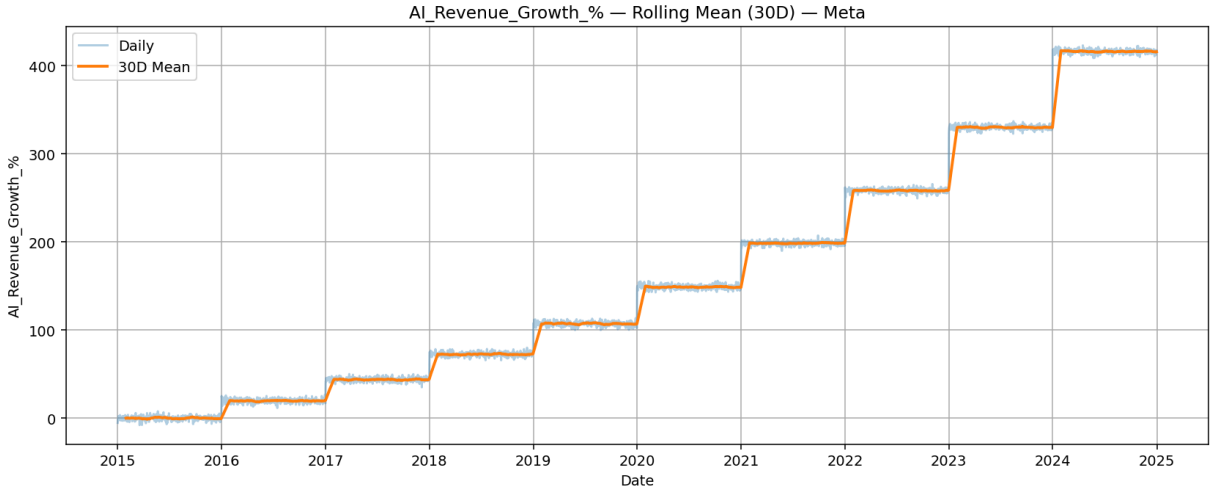


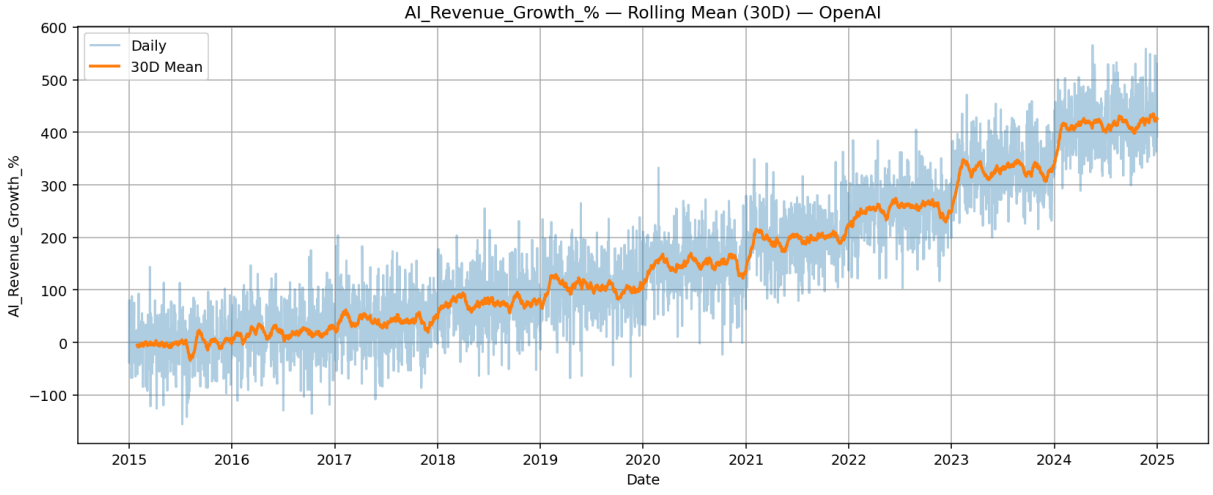


In [20]:
```python
# --- Rolling mean (30D) for AI_Revenue_Growth_% by company ---
col = "AI_Revenue_Growth_%"
if col in df.columns and "Company" in df.columns:
    for comp in sorted(df["Company"].dropna().unique().tolist()):
        sub = df[df["Company"] == comp][[col]].copy()
        if sub.empty:
            continue
        roll = sub[col].rolling(30).mean()
        plt.figure()
        plt.plot(sub.index, sub[col], alpha=0.35, label="Daily")
```

```
        plt.plot(roll.index, roll.values, linewidth=2.0, label="30D Mean")
        plt.title(f"{col} — Rolling Mean (30D) — {comp}")
        plt.xlabel("Date"); plt.ylabel(col)
        plt.legend()
        plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, f"rolling_growt
```

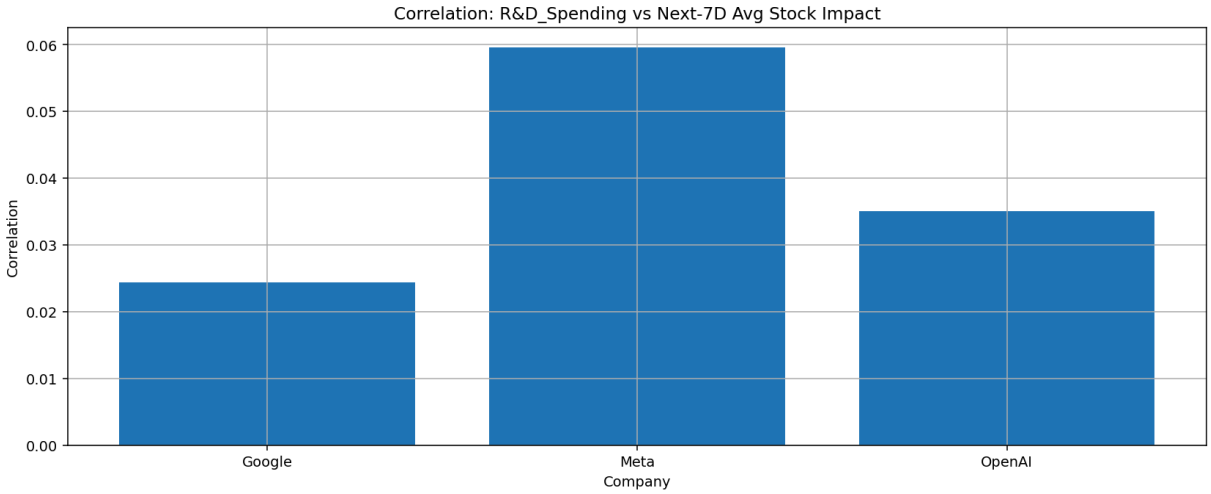






In [21]:
```
# --- Lag correlation: R&D today vs next-7D avg Stock Impact ---
xcol, ycol = "R&D_Spending_USD_Mn", "Stock_Impact_%"
if xcol in df.columns and ycol in df.columns and "Company" in df.columns:
    results = []
```

```
    for comp in sorted(df["Company"].dropna().unique().tolist()):
        sub = df[df["Company"] == comp][[xcol, ycol]].copy()
        if len(sub) < 10:
            continue
        future = sub[ycol].rolling(7).mean().shift(-7)
        xv = pd.to_numeric(sub[xcol], errors="coerce").fillna(0).values
        yv = pd.to_numeric(future, errors="coerce").fillna(0).values
        if xv.size == yv.size and xv.size > 1:
            corr = np.corrcoef(xv, yv)[0,1]
            results.append((comp, corr))
    if results:
        comps, vals = zip(*results)
        plt.figure()
        plt.bar(comps, vals)
        plt.title("Correlation: R&D_Spending vs Next-7D Avg Stock Impact")
        plt.xlabel("Company"); plt.ylabel("Correlation")
        plt.tight_layout(); plt.savefig(os.path.join(FIGDIR, "corr_randd_fut
```



In [22]:
```
# --- Save processed snapshot ---
df.to_csv(os.path.join(FIGDIR, "data_sorted.csv"))
print("Saved processed data to figures/data_sorted.csv")
print("All figures saved under:", FIGDIR)
```

```
Saved processed data to figures/data_sorted.csv
All figures saved under: figures
```