Customer Performance Analysis via Regression

You can find the full code, data, and documentation for this project on GitHub: CSCA-5622-Supervised-Learning-Final-Project (github.com/treinart) https://github.com/treinart/CSCA-5622-Supervised-Learning-Final-Project.git

This project uses supervised regression modeling to analyze dealership customer performance at the customer level. The goal is to predict each customer's average total sales, combining labor and parts, and to understand what characteristics make some customers more valuable than others.

Data Source and Description

The dataset was generated using a custom Python script (`generate_invoice_data.py`). It simulates dealership invoice activity using business-driven logic. All customer names, invoice numbers, and sales totals are synthetic to protect confidentiality.

The file `anonymized_invoice_data.csv` contains 45,514 invoices with 16 columns, including job type, location, labor and parts sales, and customer identifiers.

Project Dependencies and Environment

This notebook requires the following Python packages:

- pandas
- numpy
- matplotlib
- seaborn
- scikit-learn
- scipy
- xgboost

If you are running this notebook for the first time, please ensure all packages are installed in your Jupyter environment.
You can install any missing packages using pip (run these commands in a Jupyter code cell or your terminal):

!pip install pandas numpy matplotlib seaborn scikit-learn scipy xgboost

If you see any import errors, check that your notebook kernel matches your Python environment.
To verify, run:

import sys print(sys.executable)

This tells you which Python Jupyter is using.

If all else fails From command prompt, run:

<full_path_from_above> -m pip install xgboost

This was the only way I was successful installing XGBoost. This will guarantee that XGBoost is installed in the environment Jupyter is using. Restart the Jupyter kernel after installing. In a code cell, try:

from xgboost import XGBRegressor

If you do not get an error, it worked.

In [1]:
```
# Import core packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Set style for plots
sns.set_theme(style="darkgrid", palette="bright", context="talk")
```

Loading Invoice Data for Analysis

We now load the dealership invoice data into a pandas DataFrame for inspection and downstream processing.

In [2]:
```
# Load the invoice-level dataset from CSV using pandas
df = pd.read_csv("anonymized_invoice_data.csv")   # This loads the entire file into a DataFrame called df

# Show the first five rows to confirm successful loading and review column names and sample data
df.head()
```

Out[2]:

	Location	Whse	InvoiceNo	InvDate	CustNo	CustName	ROtype	Dept	HoursWorked	HoursBilled	LaborBilled$	Labc
0	Dallas	DL2	100000	05/12/2025	JJQ9K	Goldstar Associates	COUNTER	30	0.00	0.00	0.00	
1	Green Bay	GB1	100001	08/25/2022	JJQ9K	Goldstar Associates	TRAILER	40	2.44	1.94	604.28	
2	Dallas	DL1	100002	09/21/2022	JJQ9K	Goldstar Associates	RESALE	40	6.27	7.01	1870.92	
3	Chicago	CH1	100003	04/29/2025	JJQ9K	Goldstar Associates	RESALE	20	5.40	7.40	1940.86	
4	Green Bay	GB4	100004	10/26/2023	JJQ9K	Goldstar Associates	TRAILER	50	4.68	5.23	1270.51	

Data Structure and Cleaning Checks

We now check the structure of the data, look for missing values, and confirm column types. These steps ensure that the data is ready for aggregation and modeling.

The generator script was designed to avoid missing or illogical values, but we confirm data integrity here. We check for missing values, review data types, and plot key fields for outliers or oddities.

In [3]:
```
# Display the number of rows and columns in the dataset
df.shape    # Output: (number of rows, number of columns)
print("Rows, columns:", df.shape)
```

```
Rows, columns: (45514, 15)
```

In [4]:
```
# Print the number of unique customers in the dataset.
# This counts how many distinct customer IDs (CustNo) appear in the invoice-level data.
print("Number of unique customers:", df["CustNo"].nunique())
```

```
Number of unique customers: 387
```

In [5]:
```
# Check for missing values in each column to confirm data integrity
df.isnull().sum()     # Expect all columns to show 0 missing values
print("\nMissing values:\n", df.isnull().sum())
```

```
Missing values:
 Location        0
Whse           0
InvoiceNo      0
InvDate        0
CustNo         0
CustName       0
ROtype         0
Dept           0
HoursWorked    0
HoursBilled    0
LaborBilled$   0
LaborWorked$   0
PartsSales$    0
PartsCost$     0
InvCycleDays   0
dtype: int64
```

In [6]:

```python
# Review the data types (e.g., float, int, object) for each column
df.dtypes    # This helps identify if any columns are mis-typed
print("\nColumn types:\n", df.dtypes)
```

```
Column types:
 Location         object
Whse            object
InvoiceNo        int64
InvDate         object
CustNo          object
CustName        object
ROtype          object
Dept             int64
HoursWorked    float64
HoursBilled    float64
LaborBilled$   float64
LaborWorked$   float64
PartsSales$    float64
PartsCost$     float64
InvCycleDays   float64
dtype: object
```

In [7]:

```python
# Show summary statistics (mean, std, min, max, quartiles) for all numeric columns
df.describe()
```

Out[7]:

	InvoiceNo	Dept	HoursWorked	HoursBilled	LaborBilled$	LaborWorked$	PartsSales$	PartsCost$	Inv
count	45514.000000	45514.000000	45514.000000	45514.000000	45514.000000	45514.000000	45514.000000	45514.000000	455
mean	122756.500000	30.054489	1.943942	2.473220	902.099548	308.744873	828.727662	534.102356	
std	13138.904413	14.095500	2.400496	3.195262	1342.479969	388.871755	520.255687	298.860757	
min	100000.000000	10.000000	0.000000	0.000000	0.000000	0.000000	-299.360000	-127.570000	
25%	111378.250000	20.000000	0.000000	0.000000	0.000000	0.000000	415.047500	281.980000	
50%	122756.500000	30.000000	0.000000	0.000000	0.000000	0.000000	776.615000	527.845000	
75%	134134.750000	40.000000	4.110000	4.940000	1614.620000	631.750000	1158.635000	776.700000	
max	145513.000000	50.000000	6.900000	14.490000	13222.220000	1207.500000	2975.650000	1251.410000	

Visualizing Labor Billed $ Distribution

To better understand the distribution and possible outliers in the Labor Billed $ field, we use three complementary visualizations:

- a zoomed-in box plot,
- a standard histogram,
- and a histogram with a logarithmic x-axis.

Main goal:

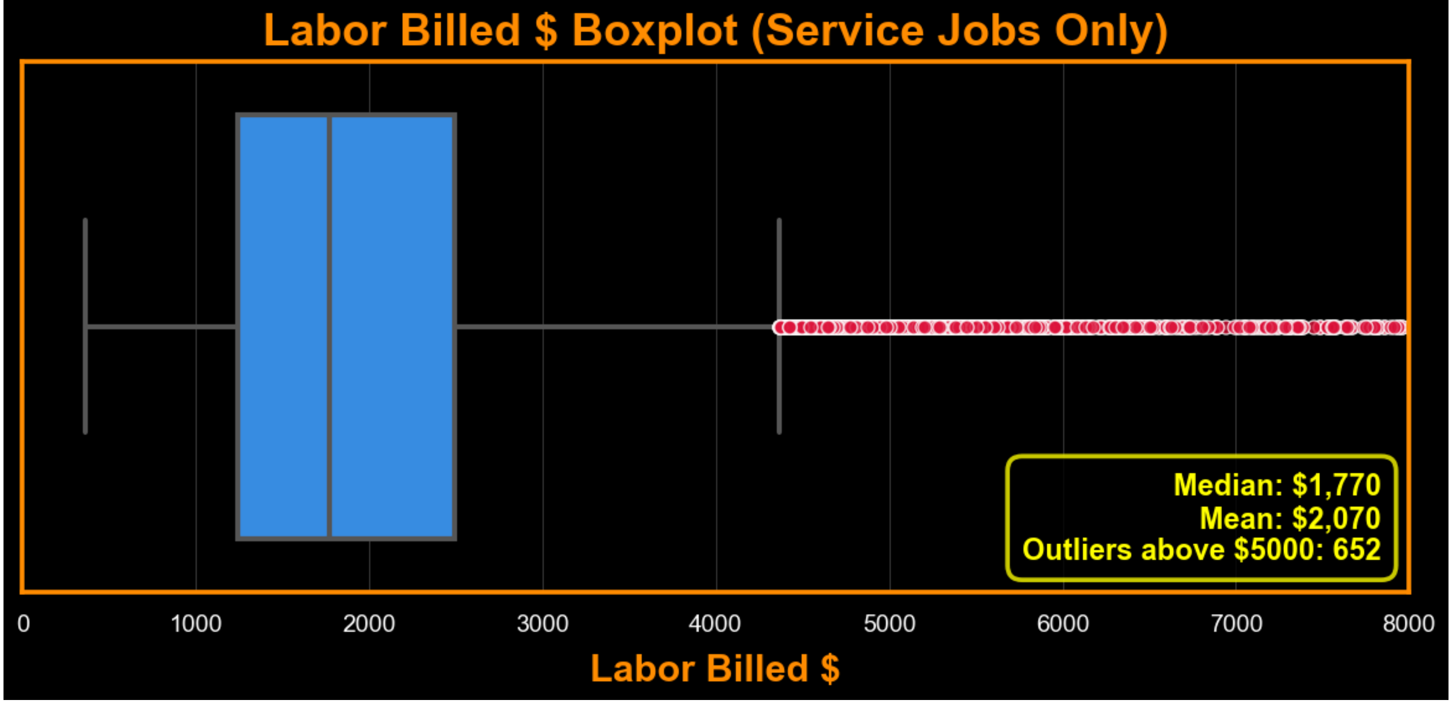
Quickly see where most jobs fall, whether there are lots of outliers, and whether the distribution is skewed (which can affect modeling and feature engineering).

Secondary goal:

Identify whether any data cleaning, scaling, or transformation might be required before modeling.

In [8]:
```
# Filter for service jobs only
service_jobs = df[df['ROtype'].isin(['RESALE', 'TRUCK', 'TRAILER'])].copy()

# Compute quartiles for IQR
q1 = service_jobs["LaborBilled$"].quantile(0.25)
q3 = service_jobs["LaborBilled$"].quantile(0.75)
iqr = q3 - q1

median_val = service_jobs["LaborBilled$"].median()
mean_val = service_jobs["LaborBilled$"].mean()
outlier_count = (service_jobs["LaborBilled$"] > 5000).sum()
total_count = service_jobs["LaborBilled$"].count()

print(f"Labor Billed $ stats for RESALE, TRUCK, and TRAILER jobs only:")
print(f"  Median: ${median_val:,.0f}")
print(f"  Mean:   ${mean_val:,.0f}")
print(f"  Outliers above $5000: {outlier_count} out of {total_count}")

plt.figure(figsize=(12, 6), facecolor='black')
flierprops = dict(marker='o', markerfacecolor='crimson', markeredgecolor='white', markersize=8, linestyle='none', alp
ax = sns.boxplot(
    x=service_jobs["LaborBilled$"],
    color='dodgerblue',
    fliersize=6, linewidth=3,
    flierprops=flierprops
)
plt.title("Labor Billed $ Boxplot (Service Jobs Only)", fontsize=26, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("Labor Billed $", fontsize=22, color='darkorange', fontweight='bold', labelpad=10)
plt.xlim(0, 8000)
plt.xticks(fontsize=14, color='white')
plt.yticks([])
ax.set_facecolor('black')

# Add horizontal grid for readability
ax.xaxis.grid(True, color='gray', linestyle='-', linewidth=0.75, alpha=0.45)

for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.7)

bbox_props = dict(boxstyle="round,pad=0.5", fc="black", ec="yellow", lw=2.7, alpha=0.80)
plt.text(0.98, 0.05,
         f"Median: ${median_val:,.0f}\nMean: ${mean_val:,.0f}\nOutliers above $5000: {outlier_count}",
         ha='right', va='bottom', fontsize=17, color='yellow', fontweight='bold',
         transform=ax.transAxes, bbox=bbox_props)

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

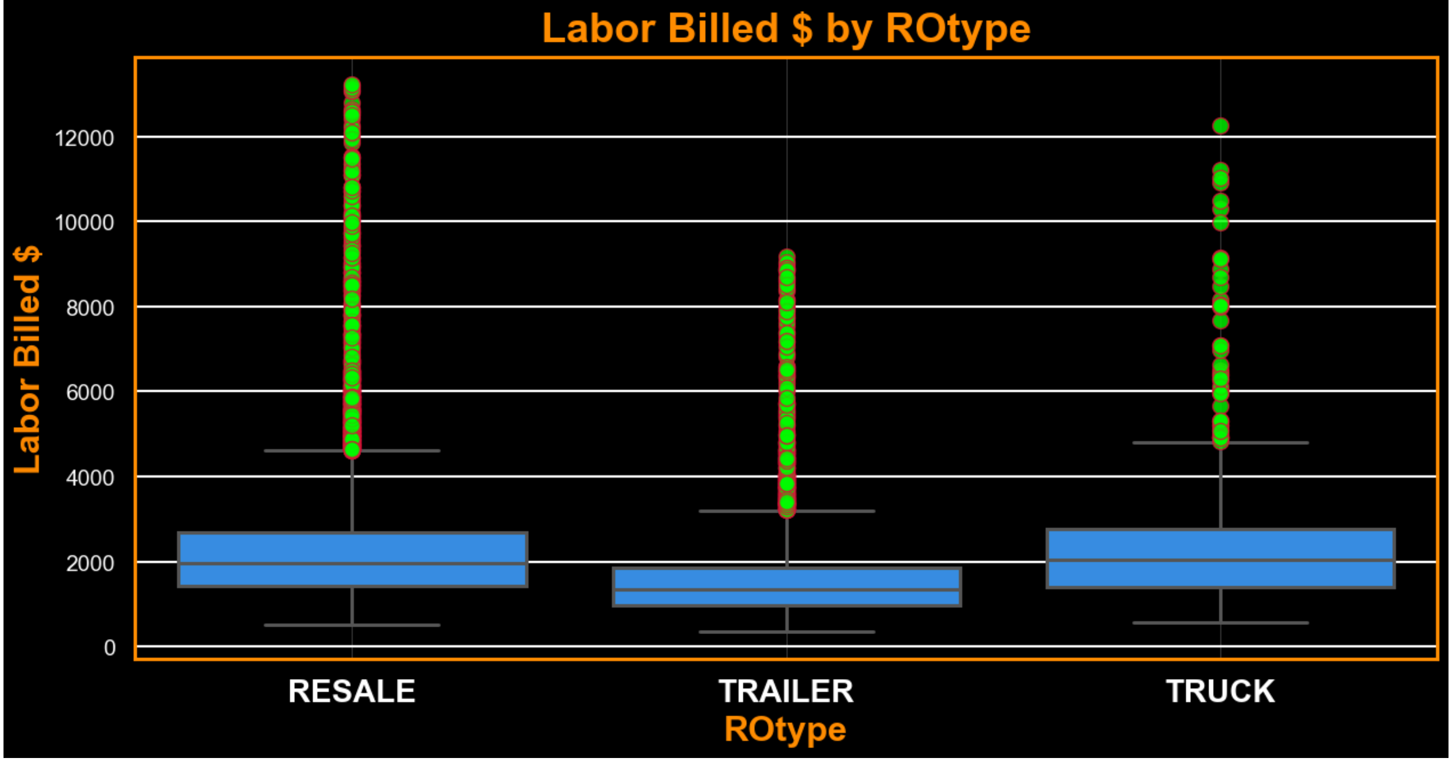
```
Labor Billed $ stats for RESALE, TRUCK, and TRAILER jobs only:
  Median: $1,770
  Mean:   $2,070
  Outliers above $5000: 652 out of 19839
```



In [9]:

```
plt.figure(figsize=(13, 7), facecolor='black')
flierprops = dict(marker='o', markerfacecolor='lime', markeredgecolor='crimson', markersize=10, linestyle='none', alp
ax = sns.boxplot(
    x="ROtype", y="LaborBilled$",
    data=service_jobs,
    order=['RESALE', 'TRAILER', 'TRUCK'],
    color='dodgerblue',
    fliersize=6, linewidth=2.2,
    flierprops=flierprops
)
plt.title("Labor Billed $ by ROtype", fontsize=26, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("ROtype", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Labor Billed $", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=20, color='white', fontweight='bold')
plt.yticks(fontsize=14, color='white')
ax.set_facecolor('black')
ax.xaxis.grid(True, color='gray', linestyle='-', linewidth=0.75, alpha=0.45)
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.2)

# Print summary stats for LaborBilled$ by ROtype
group_stats = service_jobs.groupby("ROtype")["LaborBilled$"].agg(["count", "median", "mean", "min", "max"])
group_stats = group_stats.loc[["RESALE", "TRAILER", "TRUCK"]]  # Enforce order

print("Labor Billed $ summary by ROtype:")
for ro in group_stats.index:
    row = group_stats.loc[ro]
    print(f"  {ro:8s} | Count: {int(row['count']):5d}  Median: ${row['median']:7,.0f}  Mean: ${row['mean']:7,.0f}  M:

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

```
Labor Billed $ summary by ROtype:
  RESALE   | Count: 13041  Median: $  1,974  Mean: $  2,272  Min: $533  Max: $13,222
  TRAILER  | Count:  5815  Median: $  1,362  Mean: $  1,572  Min: $358  Max: $9,187
  TRUCK    | Count:   983  Median: $  2,041  Mean: $  2,336  Min: $585  Max: $12,250
```

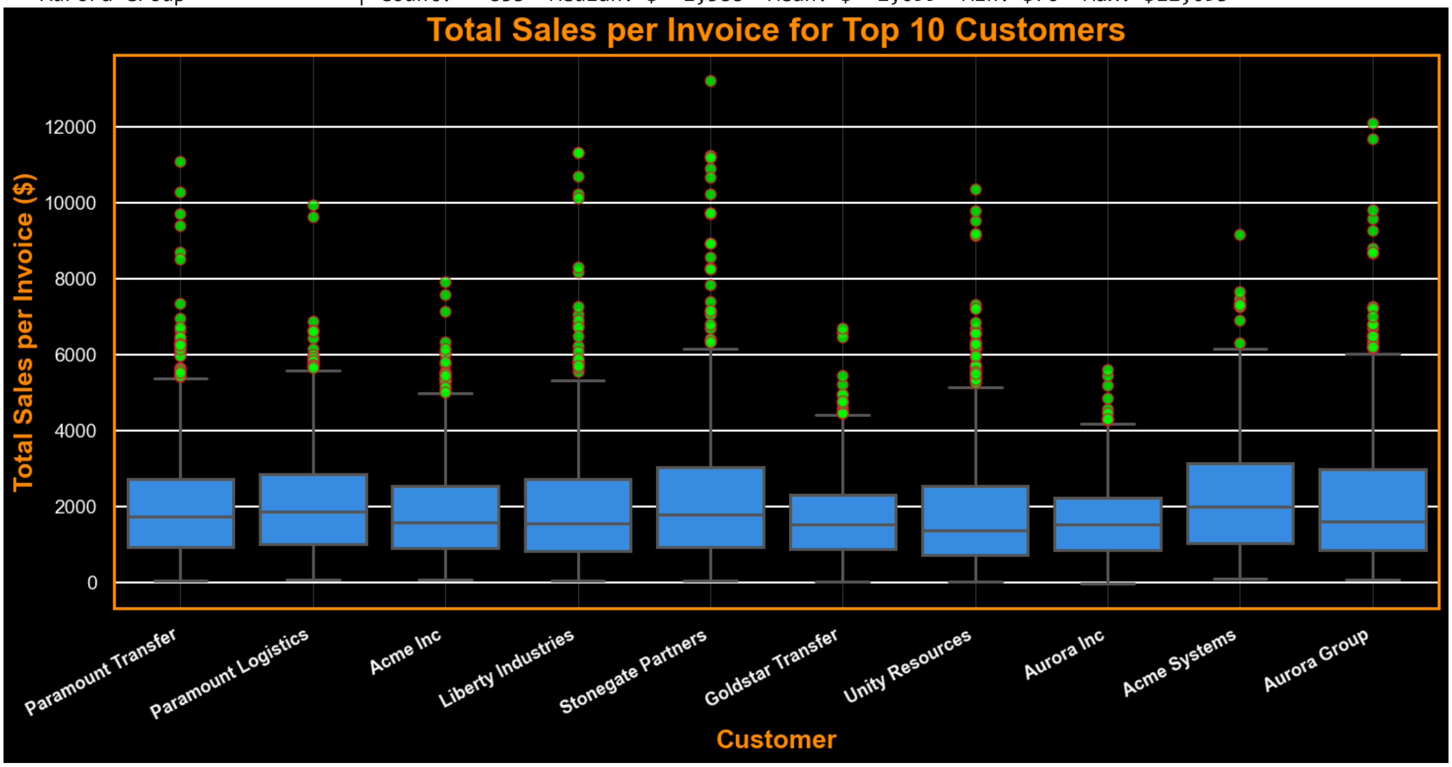


In [10]:
```python
# Calculate medians to order locations by typical labor billed
medians = service_jobs.groupby("Location")["LaborBilled$"].median().sort_values(ascending=False)
ordered_locations = medians.index.tolist()

plt.figure(figsize=(16, 8), facecolor='black')
flierprops = dict(marker='o', markerfacecolor='lime', markeredgecolor='crimson', markersize=8, linestyle='none', alph
ax = sns.boxplot(
    x="Location", y="LaborBilled$",
    data=service_jobs,
    order=ordered_locations,
    color='dodgerblue',
    fliersize=5, linewidth=2.1,
    flierprops=flierprops
)
plt.title("Labor Billed $ by Location", fontsize=26, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("Location", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Labor Billed $", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=15, color='white', fontweight='bold', rotation=30)
plt.yticks(fontsize=14, color='white')
ax.set_facecolor('black')
ax.xaxis.grid(True, color='gray', linestyle='-', linewidth=0.75, alpha=0.45)
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.2)

# Print summary stats for LaborBilled$ by Location (top 5 by median)
location_stats = service_jobs.groupby("Location")["LaborBilled$"].agg(["count", "median", "mean", "min", "max"])
top_locations = medians.index[:5]  # Show top 5 locations by median labor billed

print("Labor Billed $ summary for top 5 Locations (by median):")
for loc in top_locations:
    row = location_stats.loc[loc]
    print(f"  {loc:10s} | Count: {int(row['count']):5d}  Median: ${row['median']:7,.0f}  Mean: ${row['mean']:7,.0f}

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

```
Labor Billed $ summary for top 5 Locations (by median):
  Chicago    | Count:  3203  Median: $  2,196  Mean: $  2,612  Min: $439  Max: $13,222
  Dallas     | Count:  3288  Median: $  2,006  Mean: $  2,470  Min: $365  Max: $13,183
  Green Bay  | Count:  3264  Median: $  1,876  Mean: $  2,151  Min: $403  Max: $12,250
  New York   | Count:  3367  Median: $  1,630  Mean: $  1,780  Min: $404  Max: $8,690
  Los Angeles | Count:  3341  Median: $  1,586  Mean: $  1,743  Min: $358  Max: $8,258
```

In [11]:
```
# Compute total sales per invoice
df['TotalSales'] = df['LaborBilled$'] + df['PartsSales$']

# Aggregate total sales by customer
customer_sales = df.groupby('CustName')['TotalSales'].sum().sort_values(ascending=False)
top_customers = customer_sales.head(10).index.tolist()

# Filter for invoices from top 10 customers
top_cust_df = df[df['CustName'].isin(top_customers)].copy()

# Enforce order (so boxplot is sorted by total sales, not alphabetically)
top_cust_df['CustName'] = pd.Categorical(top_cust_df['CustName'], categories=top_customers, ordered=True)

# Print summary stats for the top 10 customers
cust_stats = top_cust_df.groupby("CustName", observed=True)["TotalSales"].agg(["count", "median", "mean", "min", "ma
print("Total Sales summary for Top 10 Customers:")
for cust in top_customers:
    row = cust_stats.loc[cust]
    print(f"  {cust:25s} | Count: {int(row['count']):5d}  Median: ${row['median']:7,.0f}  Mean: ${row['mean']:7,.0f}

# Create the boxplot
plt.figure(figsize=(15, 8), facecolor='black')
flierprops = dict(marker='o', markerfacecolor='lime', markeredgecolor='crimson', markersize=8, linestyle='none', alph
ax = sns.boxplot(
    x="CustName", y="TotalSales",
    data=top_cust_df,
    order=top_customers,
    color='dodgerblue',
    fliersize=6, linewidth=2.2,
    flierprops=flierprops
)
plt.title("Total Sales per Invoice for Top 10 Customers", fontsize=24, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("Customer", fontsize=19, color='darkorange', fontweight='bold')
plt.ylabel("Total Sales per Invoice ($)", fontsize=19, color='darkorange', fontweight='bold')
plt.xticks(fontsize=13, color='white', fontweight='bold', rotation=28, ha='right')
plt.yticks(fontsize=14, color='white')
ax.set_facecolor('black')
ax.xaxis.grid(True, color='gray', linestyle='-', linewidth=0.75, alpha=0.45)
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.2)
plt.tight_layout()
plt.show()
```

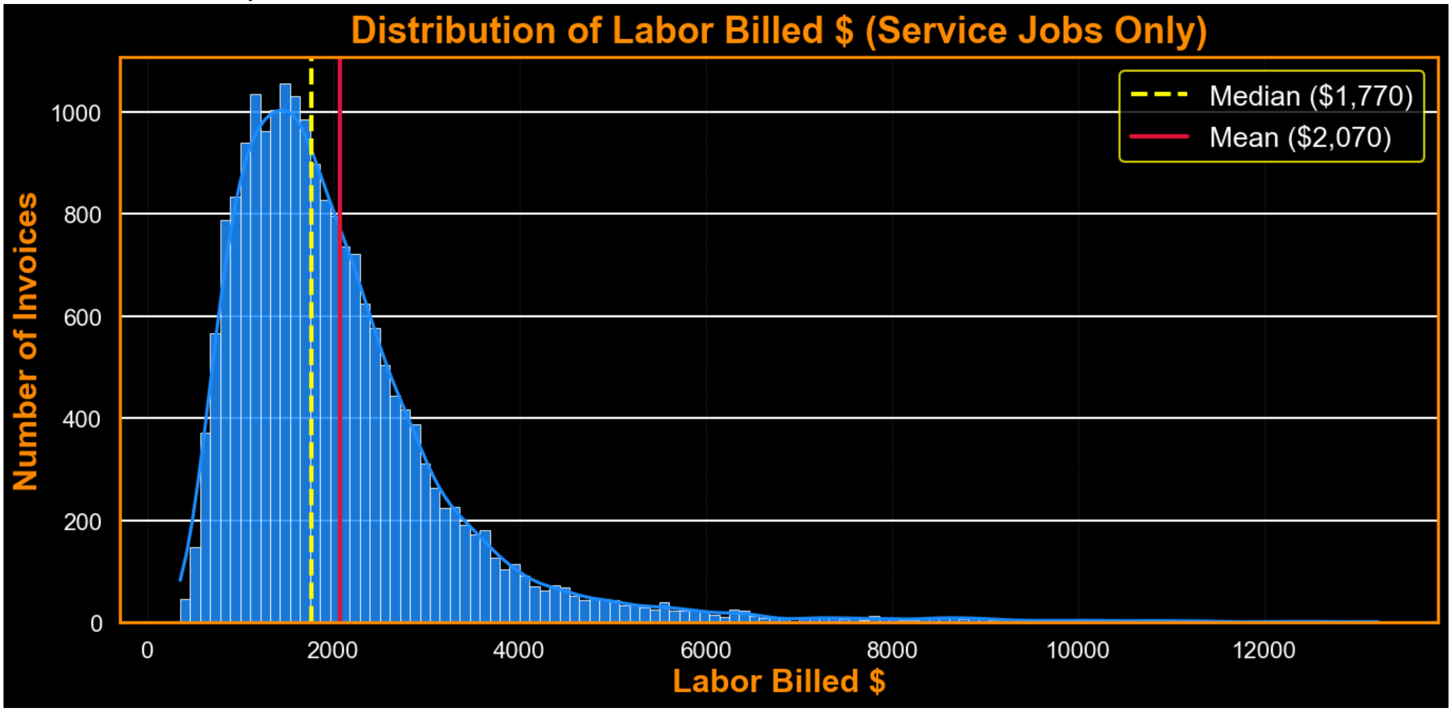
```
Total Sales summary for Top 10 Customers:
  Paramount Transfer        | Count:  2008  Median: $  1,738  Mean: $  1,927  Min: $38  Max: $11,089
  Paramount Logistics       | Count:  1689  Median: $  1,856  Mean: $  2,007  Min: $58  Max: $9,951
  Acme Inc                  | Count:  1747  Median: $  1,587  Mean: $  1,814  Min: $76  Max: $7,922
  Liberty Industries        | Count:  1677  Median: $  1,557  Mean: $  1,867  Min: $48  Max: $11,336
  Stonegate Partners        | Count:  1386  Median: $  1,788  Mean: $  2,113  Min: $36  Max: $13,233
  Goldstar Transfer         | Count:  1522  Median: $  1,534  Mean: $  1,653  Min: $23  Max: $6,708
  Unity Resources           | Count:  1295  Median: $  1,366  Mean: $  1,797  Min: $19  Max: $10,359
  Aurora Inc                | Count:  1245  Median: $  1,511  Mean: $  1,580  Min: $-35  Max: $5,605
  Acme Systems              | Count:   842  Median: $  1,989  Mean: $  2,194  Min: $80  Max: $9,157
  Aurora Group              | Count:   853  Median: $  1,588  Mean: $  2,099  Min: $76  Max: $12,093
```



In [12]:
```python
# Filter to only service jobs for labor sales analysis
service_jobs = df[df['ROtype'].isin(['RESALE', 'TRUCK', 'TRAILER'])].copy()

# Print summary stats for Labor Billed $ (service jobs only)
median_val = service_jobs["LaborBilled$"].median()
mean_val = service_jobs["LaborBilled$"].mean()
min_val = service_jobs["LaborBilled$"].min()
max_val = service_jobs["LaborBilled$"].max()
above_5000 = (service_jobs["LaborBilled$"] > 5000).sum()
total_count = service_jobs["LaborBilled$"].count()

print("Labor Billed $ summary (service jobs only):")
print(f"  Median: ${median_val:,.0f}")
print(f"  Mean:   ${mean_val:,.0f}")
print(f"  Min:    ${min_val:,.0f}")
print(f"  Max:    ${max_val:,.0f}")
print(f"  Invoices above $5,000: {above_5000} out of {total_count}")

# Improved Histogram for Labor Billed $ (Service Jobs)
plt.figure(figsize=(14, 7), facecolor='black')
ax = sns.histplot(
    service_jobs["LaborBilled$"],
    bins=120,
    kde=True,
    color='dodgerblue',
    edgecolor='white',
    alpha=0.86
)

plt.title("Distribution of Labor Billed $ (Service Jobs Only)", fontsize=25, color='darkorange', fontweight='bold', p
plt.xlabel("Labor Billed $", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Number of Invoices", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
```

```python
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.1)
ax.xaxis.grid(True, color='gray', linestyle=':', linewidth=0.7, alpha=0.43)

# Median and mean lines
plt.axvline(median_val, color='yellow', linestyle='--', linewidth=3, label=f"Median (${median_val:,.0f})")
plt.axvline(mean_val, color='crimson', linestyle='-', linewidth=3, label=f"Mean (${mean_val:,.0f})")
leg = plt.legend(fontsize=19, loc='upper right', facecolor='black', edgecolor='yellow', frameon=True)
for text in leg.get_texts():
    text.set_color("white")

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

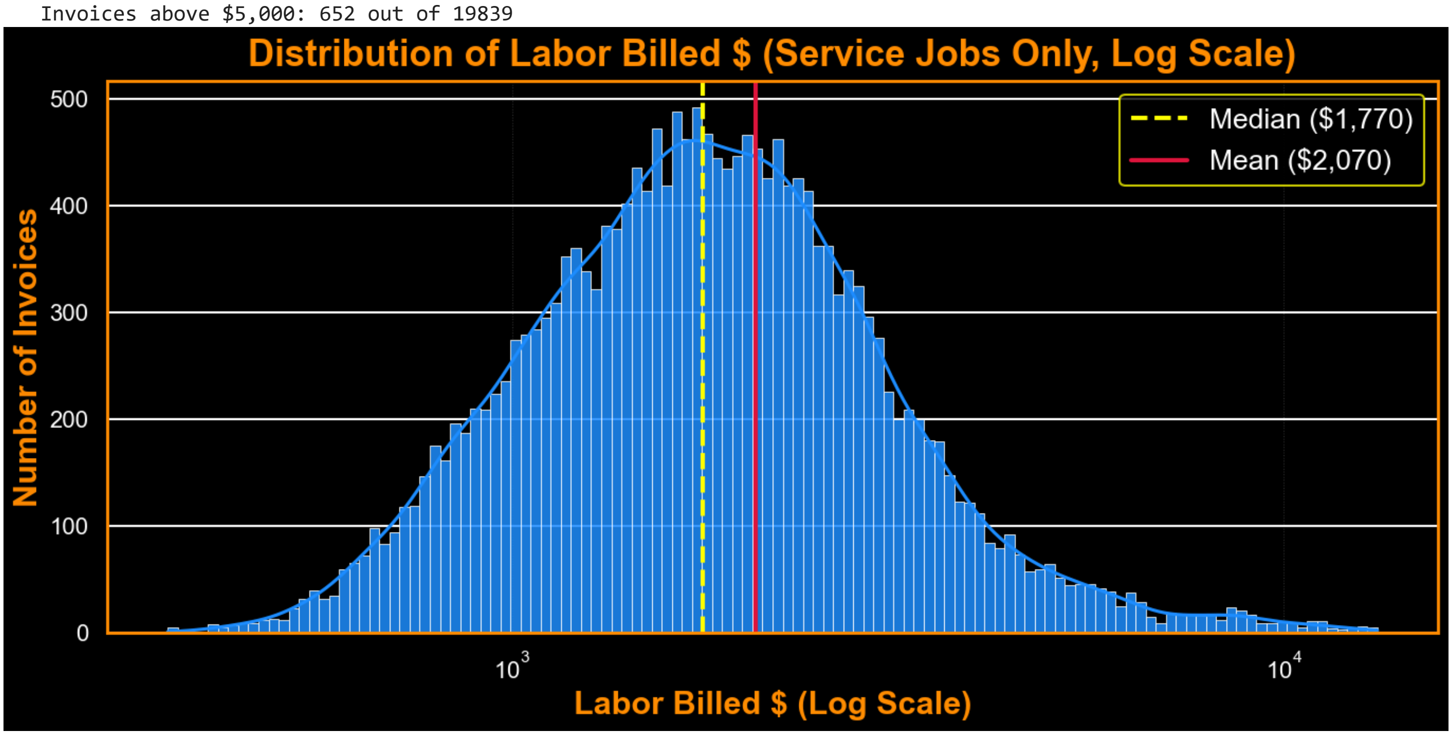
```
Labor Billed $ summary (service jobs only):
  Median: $1,770
  Mean:   $2,070
  Min:    $358
  Max:    $13,222
  Invoices above $5,000: 652 out of 19839
```



In [13]:
```python
# Filter to only service jobs for labor sales analysis
service_jobs = df[df['ROtype'].isin(['RESALE', 'TRUCK', 'TRAILER'])].copy()

# Print summary stats for Labor Billed $ (service jobs only)
median_val = service_jobs["LaborBilled$"].median()
mean_val = service_jobs["LaborBilled$"].mean()
min_val = service_jobs["LaborBilled$"].min()
max_val = service_jobs["LaborBilled$"].max()
above_5000 = (service_jobs["LaborBilled$"] > 5000).sum()
total_count = service_jobs["LaborBilled$"].count()

print("Labor Billed $ summary (service jobs only):")
print(f"  Median: ${median_val:,.0f}")
print(f"  Mean:   ${mean_val:,.0f}")
print(f"  Min:    ${min_val:,.0f}")
print(f"  Max:    ${max_val:,.0f}")
print(f"  Invoices above $5,000: {above_5000} out of {total_count}")

# Histogram with Logarithmic X-Axis
plt.figure(figsize=(14, 7), facecolor='black')
ax = sns.histplot(
    service_jobs["LaborBilled$"],
    bins=120,
    kde=True,
    color='dodgerblue',
```

```python
    edgecolor='white',
    alpha=0.86,
    log_scale=(True, False)  # Logarithmic x-axis, normal y-axis
)

plt.title("Distribution of Labor Billed $ (Service Jobs Only, Log Scale)", fontsize=25, color='darkorange', fontweigh
plt.xlabel("Labor Billed $ (Log Scale)", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Number of Invoices", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.1)
ax.xaxis.grid(True, color='gray', linestyle=':', linewidth=0.7, alpha=0.43)

# Median and mean lines
plt.axvline(median_val, color='yellow', linestyle='--', linewidth=3, label=f"Median (${median_val:,.0f})")
plt.axvline(mean_val, color='crimson', linestyle='-', linewidth=3, label=f"Mean (${mean_val:,.0f})")
leg = plt.legend(fontsize=19, loc='upper right', facecolor='black', edgecolor='yellow', frameon=True)
for text in leg.get_texts():
    text.set_color("white")

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

```
Labor Billed $ summary (service jobs only):
  Median: $1,770
  Mean:   $2,070
  Min:    $358
  Max:    $13,222
  Invoices above $5,000: 652 out of 19839
```

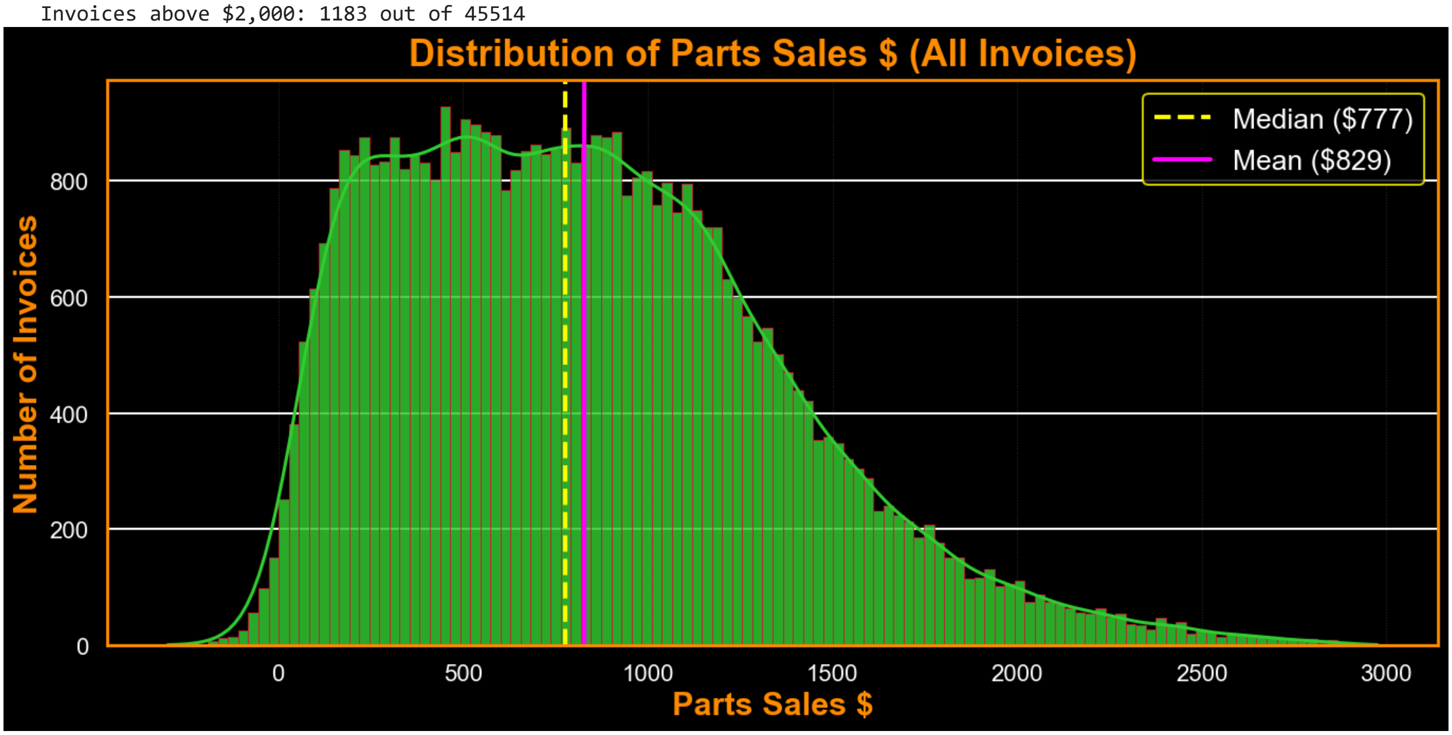


In [14]:

```python
# Filter for only service jobs if you want, or use all jobs:
# parts_jobs = df[df['ROtype'].isin(['RESALE', 'TRUCK', 'TRAILER'])].copy()
parts_jobs = df.copy()

# Print summary stats for Parts Sales $ (all invoices)
median_val = parts_jobs["PartsSales$"].median()
mean_val = parts_jobs["PartsSales$"].mean()
min_val = parts_jobs["PartsSales$"].min()
max_val = parts_jobs["PartsSales$"].max()
above_2000 = (parts_jobs["PartsSales$"] > 2000).sum()
total_count = parts_jobs["PartsSales$"].count()

print("Parts Sales $ summary (all invoices):")
print(f"  Median: ${median_val:,.0f}")
print(f"  Mean:   ${mean_val:,.0f}")
```

```
print(f"  Min:    ${min_val:,.0f}")
print(f"  Max:    ${max_val:,.0f}")
print(f"  Invoices above $2,000: {above_2000} out of {total_count}")

# Improved Histogram for Parts Sales $
plt.figure(figsize=(14, 7), facecolor='black')
ax = sns.histplot(
    parts_jobs["PartsSales$"],
    bins=120,
    kde=True,
    color='limegreen',
    edgecolor='crimson',
    alpha=0.84
)

plt.title("Distribution of Parts Sales $ (All Invoices)", fontsize=25, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("Parts Sales $", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Number of Invoices", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('darkorange')
    spine.set_linewidth(2.1)
ax.xaxis.grid(True, color='gray', linestyle=':', linewidth=0.7, alpha=0.43)

# Median and mean lines
plt.axvline(median_val, color='yellow', linestyle='--', linewidth=3, label=f"Median (${median_val:,.0f})")
plt.axvline(mean_val, color='magenta', linestyle='-', linewidth=3, label=f"Mean (${mean_val:,.0f})")
leg = plt.legend(fontsize=19, loc='upper right', facecolor='black', edgecolor='yellow', frameon=True)
for text in leg.get_texts():
    text.set_color("white")

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

```
Parts Sales $ summary (all invoices):
  Median: $777
  Mean:   $829
  Min:    $-299
  Max:    $2,976
  Invoices above $2,000: 1183 out of 45514
```



Aggregate to Customer Level

We create a customer-level table by summarizing each customer's invoices.

In [15]:
```python
# Calculate total sales for each invoice
df["TotalSales$"] = df["LaborBilled$"] + df["PartsSales$"]

# Calculate labor and parts gross margin percent (as a decimal)
df["LaborGM%"] = 1 - (df["LaborWorked$"] / df["LaborBilled$"])
df["PartsGM%"] = 1 - (df["PartsCost$"] / df["PartsSales$"])

# Calculate efficiency (hours billed divided by hours worked)
df["Efficiency"] = df["HoursBilled"] / df["HoursWorked"]

# Group by customer to create customer-level features
customer_df = df.groupby(["CustNo", "CustName"]).agg(
    AvgTotalSalesPerInvoice=("TotalSales$", "mean"),
    TotalInvoices=("InvoiceNo", "count"),
    AvgLaborGM=("LaborGM%", "mean"),
    AvgPartsGM=("PartsGM%", "mean"),
    AvgEfficiency=("Efficiency", "mean"),
    MostCommonROtype=("ROtype", lambda x: x.mode()[0] if not x.mode().empty else 'UNKNOWN'),
    MostCommonDept=("Dept", lambda x: x.mode()[0] if not x.mode().empty else -1),
    MostCommonLocation=("Location", lambda x: x.mode()[0] if not x.mode().empty else 'UNKNOWN')
).reset_index()

customer_df.head()
```

Out[15]:

	CustNo	CustName	AvgTotalSalesPerInvoice	TotalInvoices	AvgLaborGM	AvgPartsGM	AvgEfficiency	MostCommonROtype
0	016ZZ	Goldstar Solutions	2005.883652	742	0.658874	0.345703	1.370037	COUNTER
1	04CDPQ	Summit Partners	1705.610000	23	0.574704	0.223554	1.024120	COUNTER
2	050E6	Evergreen Systems	2176.141463	82	0.611540	0.232640	1.076672	RESALE
3	0AOBT3	Unity Logistics	1369.023846	13	0.592245	0.323652	1.501356	COUNTER
4	0KB68	Liberty Solutions	1885.920909	11	0.756308	0.335894	1.298439	COUNTER

Customer-Level EDA

We examine the aggregated customer dataset to understand feature distributions, relationships, and trends that might inform modeling.

In [16]:
```python
# Print the number of unique customers in the dataset.
print("Number of unique customers:", df["CustNo"].nunique())
# Show basic stats and check shape (should match your customer count)
print("Customer, Columns", customer_df.shape)
customer_df.describe()
```

```
Number of unique customers: 387
Customer, Columns (387, 10)
```

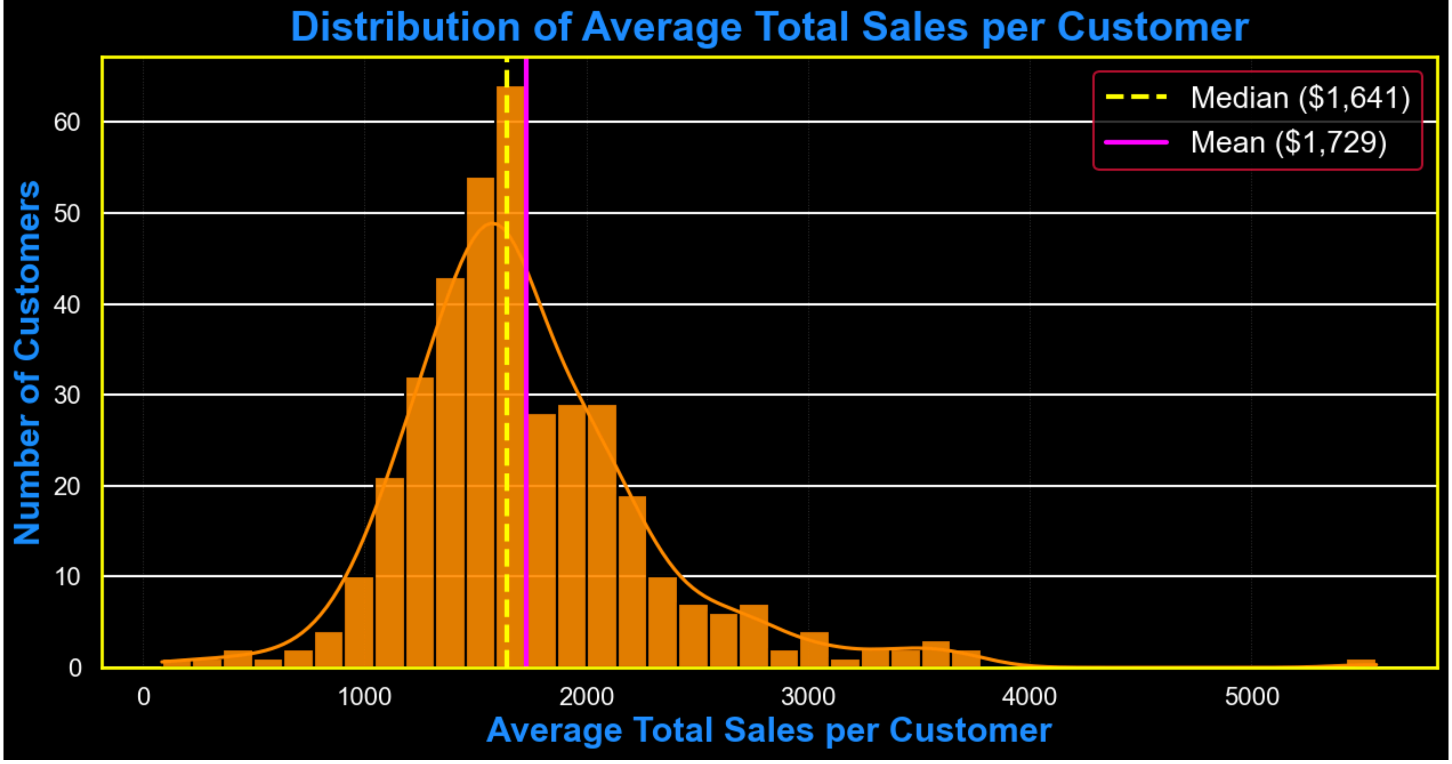
Out[16]:

	AvgTotalSalesPerInvoice	TotalInvoices	AvgLaborGM	AvgPartsGM	AvgEfficiency	MostCommonDept
count	387.000000	387.000000	375.000000	387.000000	375.000000	387.000000
mean	1729.194152	117.607235	0.611134	0.311061	1.236275	27.235142
std	573.425260	276.293905	0.086646	0.065597	0.218668	13.422354
min	86.000000	1.000000	0.330004	0.109550	0.597656	10.000000
25%	1385.931938	13.000000	0.556711	0.275666	1.107004	20.000000
50%	1641.388378	28.000000	0.617406	0.305224	1.236414	30.000000
75%	1996.960852	68.500000	0.662813	0.350300	1.380155	40.000000
max	5563.240000	2008.000000	0.837117	0.610001	1.841010	50.000000

In [17]:

```
plt.figure(figsize=(13, 7), facecolor='black')

# Print summary statistics for AvgTotalSalesPerInvoice at the customer level
median_val = customer_df["AvgTotalSalesPerInvoice"].median()
mean_val = customer_df["AvgTotalSalesPerInvoice"].mean()
min_val = customer_df["AvgTotalSalesPerInvoice"].min()
max_val = customer_df["AvgTotalSalesPerInvoice"].max()
q25 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.25)
q75 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.75)

print("Average Total Sales per Customer summary:")
print(f"  Median: ${median_val:,.0f}")
print(f"  Mean:   ${mean_val:,.0f}")
print(f"  Min:    ${min_val:,.0f}")
print(f"  Max:    ${max_val:,.0f}")
print(f"  25th percentile: ${q25:,.0f}")
print(f"  75th percentile: ${q75:,.0f}")

ax = sns.histplot(customer_df["AvgTotalSalesPerInvoice"], bins=40, kde=True, color='darkorange', edgecolor='black', a
median_val = customer_df["AvgTotalSalesPerInvoice"].median()
mean_val = customer_df["AvgTotalSalesPerInvoice"].mean()
plt.axvline(median_val, color='yellow', linestyle='--', linewidth=3, label=f"Median (${median_val:,.0f})")
plt.axvline(mean_val, color='magenta', linestyle='-', linewidth=3, label=f"Mean (${mean_val:,.0f})")
plt.title("Distribution of Average Total Sales per Customer", fontsize=26, color='dodgerblue', fontweight='bold', pad
plt.xlabel("Average Total Sales per Customer", fontsize=22, color='dodgerblue', fontweight='bold')
plt.ylabel("Number of Customers", fontsize=22, color='dodgerblue', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('yellow')
    spine.set_linewidth(2)
ax.xaxis.grid(True, color='gray', linestyle=':', linewidth=0.7, alpha=0.43)
leg = plt.legend(fontsize=19, loc='upper right', facecolor='black', edgecolor='crimson', frameon=True)
for text in leg.get_texts():
    text.set_color("white")
plt.tight_layout()
plt.show()
```

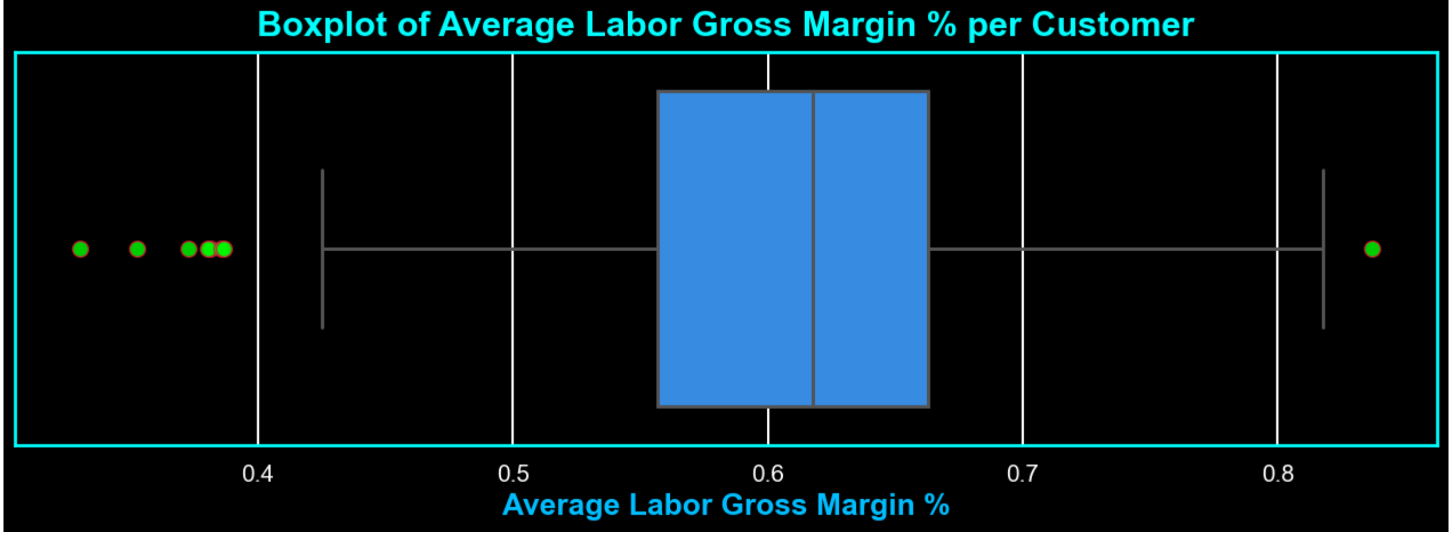
```
Average Total Sales per Customer summary:
  Median: $1,641
  Mean:   $1,729
  Min:    $86
  Max:    $5,563
  25th percentile: $1,386
  75th percentile: $1,997
```



In [18]:

```
plt.figure(figsize=(13, 5), facecolor='black')

# Print summary statistics for average labor gross margin percent per customer
median_gm = customer_df["AvgLaborGM"].median()
mean_gm = customer_df["AvgLaborGM"].mean()
min_gm = customer_df["AvgLaborGM"].min()
max_gm = customer_df["AvgLaborGM"].max()
p25_gm = customer_df["AvgLaborGM"].quantile(0.25)
p75_gm = customer_df["AvgLaborGM"].quantile(0.75)
print("Average Labor Gross Margin % per Customer summary:")
print(f"  Median: {median_gm:.2%}")
print(f"  Mean:   {mean_gm:.2%}")
print(f"  Min:    {min_gm:.2%}")
print(f"  Max:    {max_gm:.2%}")
print(f"  25th percentile: {p25_gm:.2%}")
print(f"  75th percentile: {p75_gm:.2%}")

ax = sns.boxplot(x=customer_df["AvgLaborGM"], color='dodgerblue', fliersize=7, linewidth=2.1,
                 flierprops=dict(marker='o', markerfacecolor='lime', markeredgecolor='crimson', markersize=10, linest
plt.title("Boxplot of Average Labor Gross Margin % per Customer", fontsize=22, color='cyan', fontweight='bold', pad=1
plt.xlabel("Average Labor Gross Margin %", fontsize=19, color='deepskyblue', fontweight='bold')
plt.xticks(fontsize=15, color='white')
plt.yticks([])
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('cyan')
    spine.set_linewidth(2)
plt.tight_layout()
plt.show()
```

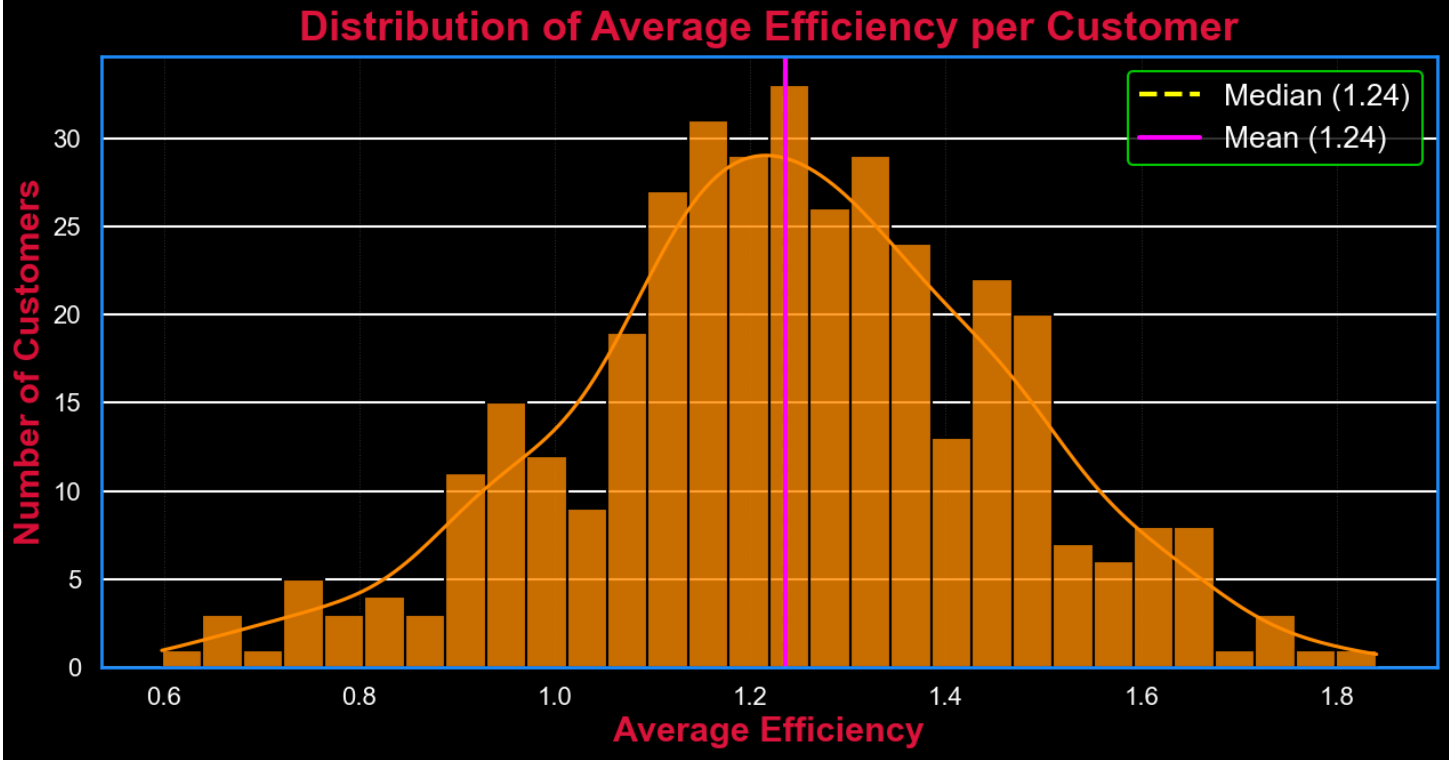
```
Average Labor Gross Margin % per Customer summary:
  Median: 61.74%
  Mean:   61.11%
  Min:    33.00%
  Max:    83.71%
  25th percentile: 55.67%
  75th percentile: 66.28%
```



In [19]:

```
plt.figure(figsize=(13, 7), facecolor='black')

# Print summary statistics for AvgEfficiency per customer
eff_median = customer_df["AvgEfficiency"].median()
eff_mean = customer_df["AvgEfficiency"].mean()
eff_min = customer_df["AvgEfficiency"].min()
eff_max = customer_df["AvgEfficiency"].max()
eff_25 = customer_df["AvgEfficiency"].quantile(0.25)
eff_75 = customer_df["AvgEfficiency"].quantile(0.75)

print("Average Efficiency per Customer summary:")
print(f"  Median: {eff_median:.2f}")
print(f"  Mean:   {eff_mean:.2f}")
print(f"  Min:    {eff_min:.2f}")
print(f"  Max:    {eff_max:.2f}")
print(f"  25th percentile: {eff_25:.2f}")
print(f"  75th percentile: {eff_75:.2f}")

ax = sns.histplot(customer_df["AvgEfficiency"], bins=30, kde=True, color='darkorange', edgecolor='black', alpha=0.8)
median_val = customer_df["AvgEfficiency"].median()
mean_val = customer_df["AvgEfficiency"].mean()
plt.axvline(median_val, color='yellow', linestyle='--', linewidth=3, label=f"Median ({median_val:.2f})")
plt.axvline(mean_val, color='magenta', linestyle='-', linewidth=3, label=f"Mean ({mean_val:.2f})")
plt.title("Distribution of Average Efficiency per Customer", fontsize=26, color='crimson', fontweight='bold', pad=10)
plt.xlabel("Average Efficiency", fontsize=22, color='crimson', fontweight='bold')
plt.ylabel("Number of Customers", fontsize=22, color='crimson', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('dodgerblue')
    spine.set_linewidth(2)
ax.xaxis.grid(True, color='gray', linestyle=':', linewidth=0.7, alpha=0.43)
leg = plt.legend(fontsize=19, loc='upper right', facecolor='black', edgecolor='lime', frameon=True)
for text in leg.get_texts():
    text.set_color("white")
plt.tight_layout()
plt.show()
```

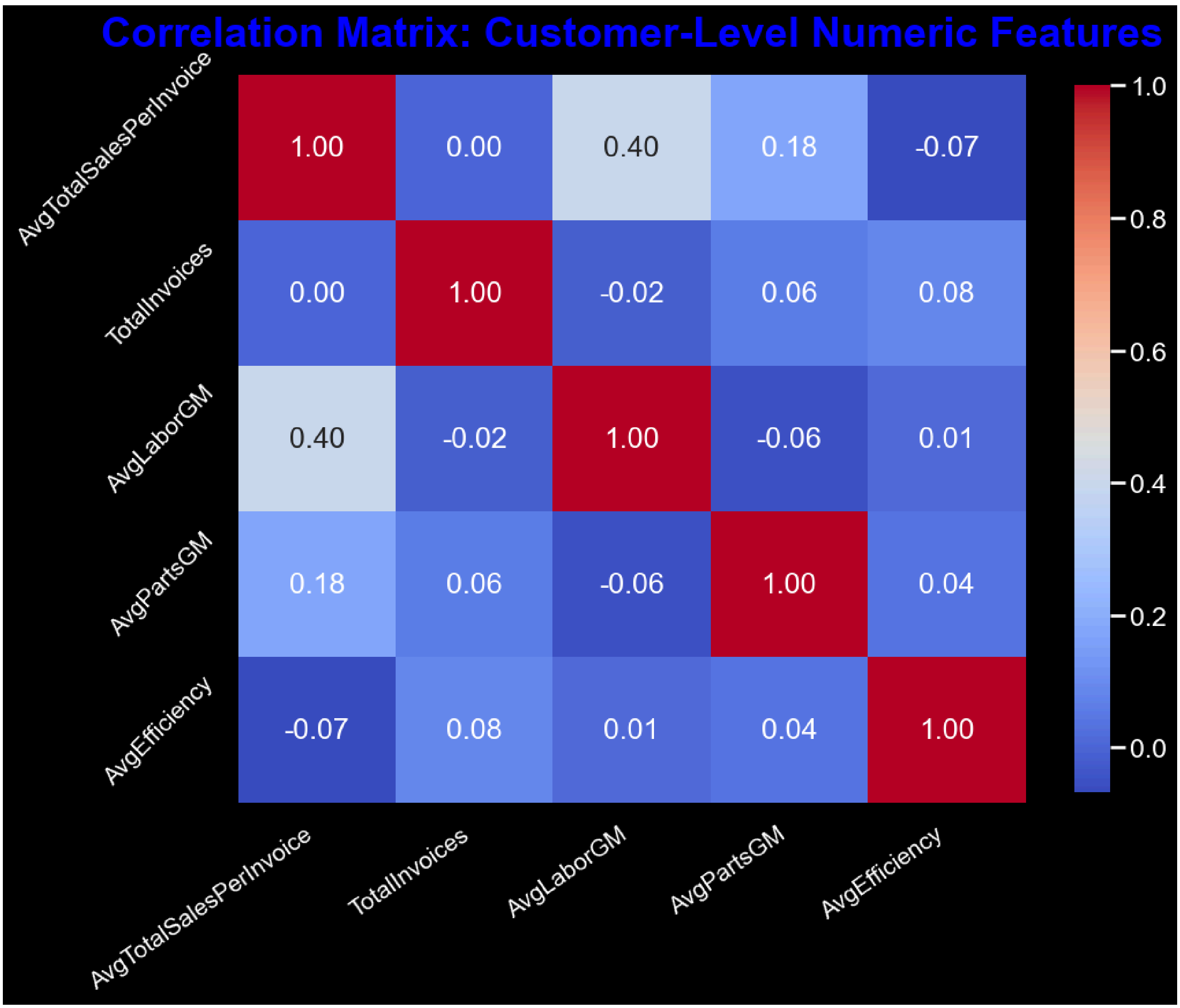
```
Average Efficiency per Customer summary:
  Median: 1.24
  Mean:   1.24
  Min:    0.60
  Max:    1.84
  25th percentile: 1.11
  75th percentile: 1.38
```



In [20]:

```
plt.figure(figsize=(11, 9), facecolor='black')

numeric_cols = [
    "AvgTotalSalesPerInvoice",
    "TotalInvoices",
    "AvgLaborGM",
    "AvgPartsGM",
    "AvgEfficiency"
]
corr_matrix = customer_df[numeric_cols].corr()

print("Correlation Matrix Summary:")
for row in corr_matrix.index:
    for col in corr_matrix.columns:
        if row != col:
            print(f"  {row:22s} vs. {col:22s}: {corr_matrix.loc[row, col]:.2f}")
    print("-" * 50)

corr = customer_df[["AvgTotalSalesPerInvoice", "TotalInvoices", "AvgLaborGM", "AvgPartsGM", "AvgEfficiency"]].corr()
ax = sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", cbar_kws={'shrink': 0.97})

plt.title("Correlation Matrix: Customer-Level Numeric Features", fontsize=26, color='blue', fontweight='bold', pad=18
plt.xticks(fontsize=15, color='white', rotation=35, ha='right')
plt.yticks(fontsize=15, color='white', rotation=45)
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('blue')
    spine.set_linewidth(2.1)

cbar = ax.collections[0].colorbar
cbar.ax.yaxis.set_tick_params(color='white', labelcolor='white')
cbar.outline.set_edgecolor('white')

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

```
Correlation Matrix Summary:
  AvgTotalSalesPerInvoice vs. TotalInvoices         : 0.00
  AvgTotalSalesPerInvoice vs. AvgLaborGM            : 0.40
  AvgTotalSalesPerInvoice vs. AvgPartsGM            : 0.18
  AvgTotalSalesPerInvoice vs. AvgEfficiency         : -0.07
--------------------------------------------------
  TotalInvoices          vs. AvgTotalSalesPerInvoice: 0.00
  TotalInvoices          vs. AvgLaborGM            : -0.02
  TotalInvoices          vs. AvgPartsGM            : 0.06
  TotalInvoices          vs. AvgEfficiency         : 0.08
--------------------------------------------------
  AvgLaborGM             vs. AvgTotalSalesPerInvoice: 0.40
  AvgLaborGM             vs. TotalInvoices         : -0.02
  AvgLaborGM             vs. AvgPartsGM            : -0.06
  AvgLaborGM             vs. AvgEfficiency         : 0.01
--------------------------------------------------
  AvgPartsGM             vs. AvgTotalSalesPerInvoice: 0.18
  AvgPartsGM             vs. TotalInvoices         : 0.06
  AvgPartsGM             vs. AvgLaborGM            : -0.06
  AvgPartsGM             vs. AvgEfficiency         : 0.04
--------------------------------------------------
  AvgEfficiency          vs. AvgTotalSalesPerInvoice: -0.07
  AvgEfficiency          vs. TotalInvoices         : 0.08
  AvgEfficiency          vs. AvgLaborGM            : 0.01
  AvgEfficiency          vs. AvgPartsGM            : 0.04
--------------------------------------------------
```

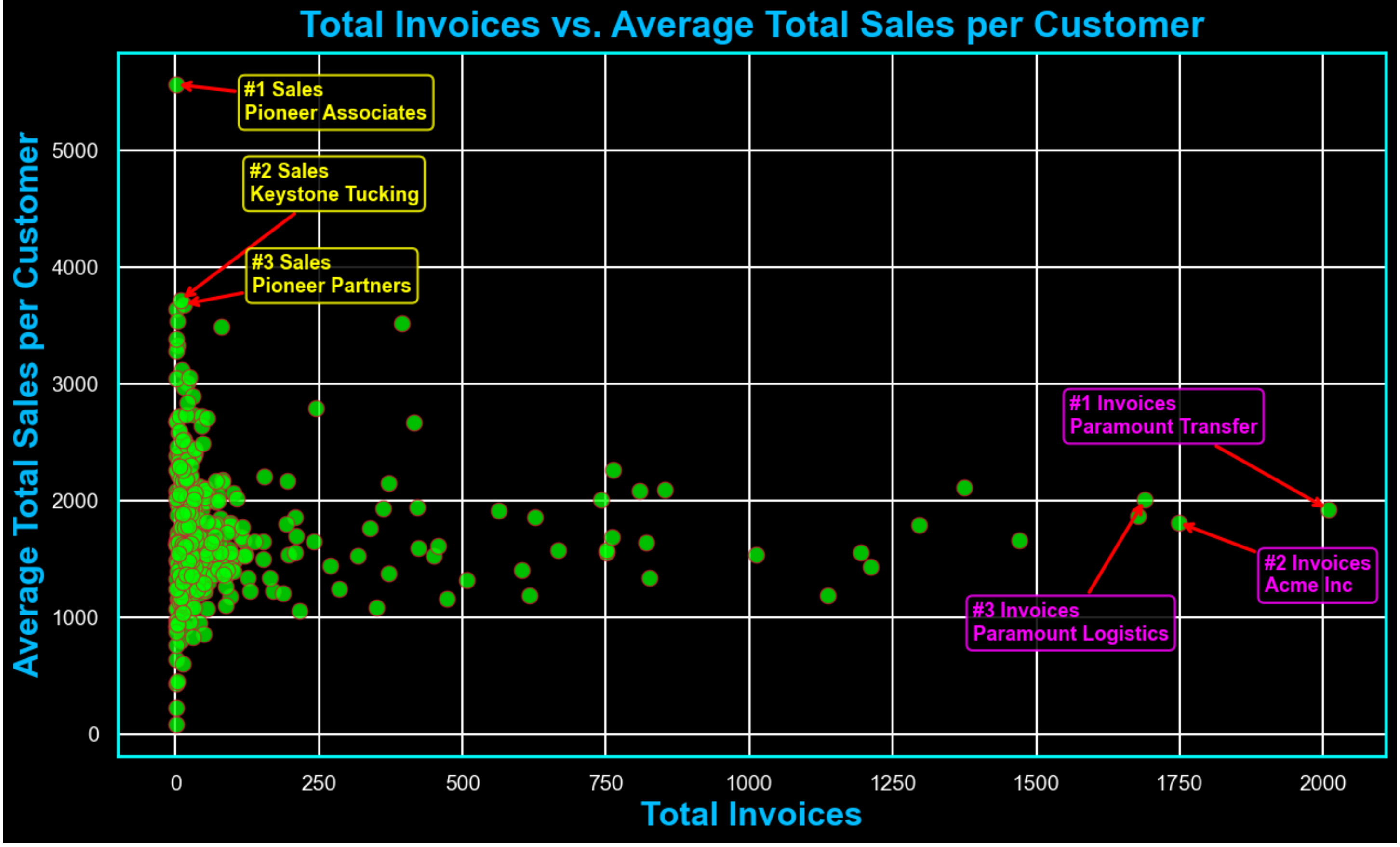


In [21]:
```
# Summary print outputs (keep as you have)
ti_median = customer_df["TotalInvoices"].median()
ti_mean = customer_df["TotalInvoices"].mean()
```

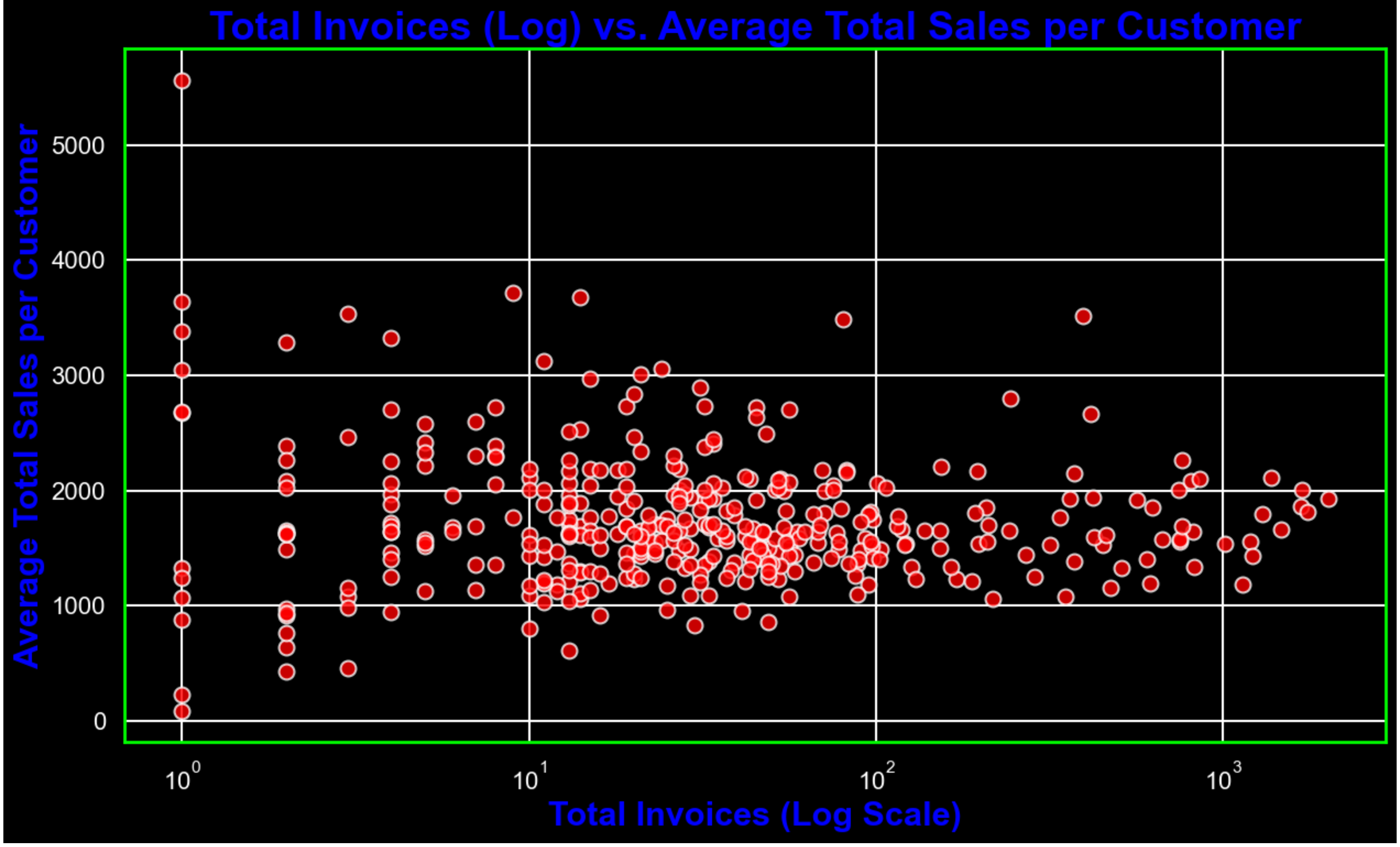
```python
ti_min = customer_df["TotalInvoices"].min()
ti_max = customer_df["TotalInvoices"].max()
ti_25 = customer_df["TotalInvoices"].quantile(0.25)
ti_75 = customer_df["TotalInvoices"].quantile(0.75)

sales_median = customer_df["AvgTotalSalesPerInvoice"].median()
sales_mean = customer_df["AvgTotalSalesPerInvoice"].mean()
sales_min = customer_df["AvgTotalSalesPerInvoice"].min()
sales_max = customer_df["AvgTotalSalesPerInvoice"].max()
sales_25 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.25)
sales_75 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.75)

print("Total Invoices per Customer summary:")
print(f"  Median: {ti_median:.0f}")
print(f"  Mean:   {ti_mean:.1f}")
print(f"  Min:    {ti_min:.0f}")
print(f"  Max:    {ti_max:.0f}")
print(f"  25th percentile: {ti_25:.0f}")
print(f"  75th percentile: {ti_75:.0f}")
print()
print("Average Total Sales per Customer summary:")
print(f"  Median: ${sales_median:,.0f}")
print(f"  Mean:   ${sales_mean:,.0f}")
print(f"  Min:    ${sales_min:,.0f}")
print(f"  Max:    ${sales_max:,.0f}")
print(f"  25th percentile: ${sales_25:,.0f}")
print(f"  75th percentile: ${sales_75:,.0f}")

# Print Top 10 Customers by Total Invoices
print("Top 10 Customers by Total Invoices:")
top10_invoices = customer_df.nlargest(10, "TotalInvoices")
for idx, row in top10_invoices.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Invoices: {int(row['TotalInvoices']):5d} | Avg Total Sales: ${row['Av

print("\nTop 10 Customers by Average Total Sales per Invoice:")
top10_sales = customer_df.nlargest(10, "AvgTotalSalesPerInvoice")
for idx, row in top10_sales.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Avg Total Sales: ${row['AvgTotalSalesPerInvoice']:,.0f} | Invoices: 
print("\n" + "-"*70 + "\n")

# --- Annotate top customers ---
top3_sales = customer_df.nlargest(3, "AvgTotalSalesPerInvoice")
top3_invoices = customer_df.nlargest(3, "TotalInvoices")

plt.figure(figsize=(13, 8), facecolor='black')
ax = sns.scatterplot(
    x="TotalInvoices", y="AvgTotalSalesPerInvoice", data=customer_df,
    color='lime', edgecolor='crimson', s=110, alpha=0.75
)

plt.title("Total Invoices vs. Average Total Sales per Customer", fontsize=24, color='deepskyblue', fontweight='bold',
plt.xlabel("Total Invoices", fontsize=22, color='deepskyblue', fontweight='bold')
plt.ylabel("Average Total Sales per Customer", fontsize=22, color='deepskyblue', fontweight='bold')
plt.xticks(fontsize=14, color='white')
plt.yticks(fontsize=14, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('cyan')
    spine.set_linewidth(2)
plt.tight_layout()

# Find top 3 by sales and by invoices
top3_sales = customer_df.nlargest(3, "AvgTotalSalesPerInvoice")
top3_invoices = customer_df.nlargest(3, "TotalInvoices")

# Manually set label offsets so they don't overlap (tweak as needed)
sales_offsets = [(120, -300), (120, 850), (120, 100)]
invoices_offsets = [(-450, 650), (150, -600), (-300, -1200)]

# Annotate Top 3 by Sales (yellow)
for i, (_, row) in enumerate(top3_sales.iterrows()):
    dx, dy = sales_offsets[i]
    plt.annotate(
```

```python
        f"#{i+1} Sales\n{row['CustName']}",
        xy=(row["TotalInvoices"], row["AvgTotalSalesPerInvoice"]),
        xytext=(row["TotalInvoices"]+dx, row["AvgTotalSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='yellow', edgecolor='red', arrowstyle='->', linewidth=2.3),
        fontsize=13, color='yellow', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='yellow', alpha=0.85)
    )

# Annotate Top 3 by Total Invoices (magenta)
for i, (_, row) in enumerate(top3_invoices.iterrows()):
    dx, dy = invoices_offsets[i]
    plt.annotate(
        f"#{i+1} Invoices\n{row['CustName']}",
        xy=(row["TotalInvoices"], row["AvgTotalSalesPerInvoice"]),
        xytext=(row["TotalInvoices"]+dx, row["AvgTotalSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='magenta', edgecolor='red', arrowstyle='->', linewidth=2.3),
        fontsize=13, color='magenta', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='magenta', alpha=0.85)
    )

plt.show()
```

```
Total Invoices per Customer summary:
  Median: 28
  Mean:   117.6
  Min:    1
  Max:    2008
  25th percentile: 13
  75th percentile: 68

Average Total Sales per Customer summary:
  Median: $1,641
  Mean:   $1,729
  Min:    $86
  Max:    $5,563
  25th percentile: $1,386
  75th percentile: $1,997
Top 10 Customers by Total Invoices:
21. Paramount Transfer        | Invoices:  2008 | Avg Total Sales: $1,927
121. Acme Inc                 | Invoices:  1747 | Avg Total Sales: $1,814
34. Paramount Logistics       | Invoices:  1689 | Avg Total Sales: $2,007
85. Liberty Industries        | Invoices:  1677 | Avg Total Sales: $1,867
142. Goldstar Transfer        | Invoices:  1470 | Avg Total Sales: $1,662
151. Stonegate Partners       | Invoices:  1375 | Avg Total Sales: $2,113
311. Unity Resources          | Invoices:  1295 | Avg Total Sales: $1,797
305. Evergreen Associates     | Invoices:  1212 | Avg Total Sales: $1,432
35. Aurora Inc                | Invoices:  1193 | Avg Total Sales: $1,558
10. Evergreen LLC             | Invoices:  1136 | Avg Total Sales: $1,186

Top 10 Customers by Average Total Sales per Invoice:
44. Pioneer Associates        | Avg Total Sales: $5,563 | Invoices:     1
327. Keystone Tucking         | Avg Total Sales: $3,718 | Invoices:     9
313. Pioneer Partners         | Avg Total Sales: $3,681 | Invoices:    14
45. Pioneer Truck Lines       | Avg Total Sales: $3,637 | Invoices:     1
221. Evergreen Transfer       | Avg Total Sales: $3,539 | Invoices:     3
307. Sterling Systems         | Avg Total Sales: $3,520 | Invoices:   395
149. Synergy Solutions        | Avg Total Sales: $3,492 | Invoices:    80
192. Sterling Corp            | Avg Total Sales: $3,383 | Invoices:     1
37. Evergreen Transfer        | Avg Total Sales: $3,329 | Invoices:     4
65. Keystone Global           | Avg Total Sales: $3,287 | Invoices:     2

----------------------------------------------------------------------
```



In [22]:
```python
plt.figure(figsize=(13, 8), facecolor='black')
ax = plt.gca()
plt.scatter(customer_df["TotalInvoices"], customer_df["AvgTotalSalesPerInvoice"],
            s=100, c='red', edgecolors='white', alpha=0.8)
plt.xscale('log')
plt.title("Total Invoices (Log) vs. Average Total Sales per Customer", fontsize=26, color='blue', fontweight='bold')
plt.xlabel("Total Invoices (Log Scale)", fontsize=22, color='blue', fontweight='bold')
plt.ylabel("Average Total Sales per Customer", fontsize=22, color='blue', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('lime')
    spine.set_linewidth(2)
plt.tight_layout()
plt.show()
```

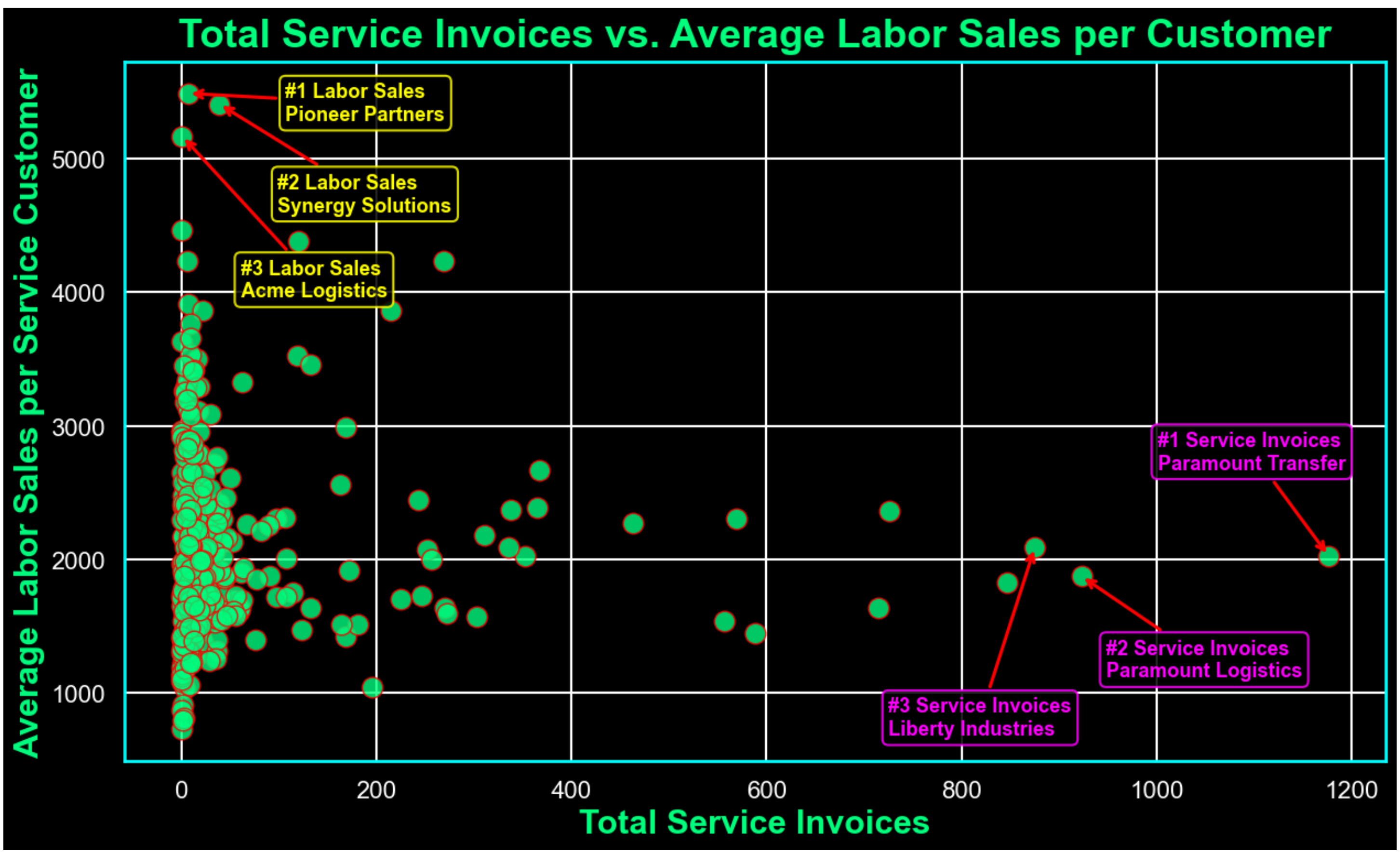


In [23]:

```python
# --- SERVICE ONLY: Group by customer for service invoices ---
service_mask = df["ROtype"].isin(["RESALE", "TRUCK", "TRAILER"])
service_group = df[service_mask].groupby(["CustNo", "CustName"]).agg(
    TotalServiceInvoices=("InvoiceNo", "count"),
    AvgLaborSalesPerInvoice=("LaborBilled$", "mean"),
).reset_index()

# --- Print summaries ---
labor_median = service_group["AvgLaborSalesPerInvoice"].median()
labor_mean = service_group["AvgLaborSalesPerInvoice"].mean()
labor_min = service_group["AvgLaborSalesPerInvoice"].min()
labor_max = service_group["AvgLaborSalesPerInvoice"].max()
labor_25 = service_group["AvgLaborSalesPerInvoice"].quantile(0.25)
labor_75 = service_group["AvgLaborSalesPerInvoice"].quantile(0.75)

print("Average Labor Sales per Service Customer summary:")
print(f"  Median: ${labor_median:,.0f}")
print(f"  Mean:   ${labor_mean:,.0f}")
print(f"  Min:    ${labor_min:,.0f}")
print(f"  Max:    ${labor_max:,.0f}")
print(f"  25th percentile: ${labor_25:,.0f}")
print(f"  75th percentile: ${labor_75:,.0f}")

print("\nTop 10 Customers by Total Service Invoices:")
top10_service_invoices = service_group.nlargest(10, "TotalServiceInvoices")
for idx, row in top10_service_invoices.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Service Invoices: {int(row['TotalServiceInvoices']):5d} | Avg Labor S

print("\nTop 10 Customers by Avg Labor Sales per Service Invoice:")
top10_labor_sales = service_group.nlargest(10, "AvgLaborSalesPerInvoice")
for idx, row in top10_labor_sales.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Avg Labor Sales: ${row['AvgLaborSalesPerInvoice']:,.0f} | Service Inv
print("\n" + "-"*70 + "\n")

# --- Annotate top 3 by labor sales and top 3 by service invoices ---
top3_labor = service_group.nlargest(3, "AvgLaborSalesPerInvoice")
top3_servinv = service_group.nlargest(3, "TotalServiceInvoices")

plt.figure(figsize=(13, 8), facecolor='black')
ax = sns.scatterplot(
    x="TotalServiceInvoices", y="AvgLaborSalesPerInvoice", data=service_group,
```

```python
    color='springgreen', edgecolor='red', s=180, alpha=0.80
)
plt.title("Total Service Invoices vs. Average Labor Sales per Customer", fontsize=26, color='springgreen', fontweight
plt.xlabel("Total Service Invoices", fontsize=22, color='springgreen', fontweight='bold')
plt.ylabel("Average Labor Sales per Service Customer", fontsize=22, color='springgreen', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('cyan')
    spine.set_linewidth(2)
plt.tight_layout()

sales_offsets = [(100, -200), (60, -800), (60, -1200)]
invoices_offsets = [(-175, 650), (25, -750), (-150, -1400)]

for i, (_, row) in enumerate(top3_labor.iterrows()):
    dx, dy = sales_offsets[i]
    plt.annotate(
        f"#{i+1} Labor Sales\n{row['CustName']}",
        xy=(row["TotalServiceInvoices"], row["AvgLaborSalesPerInvoice"]),
        xytext=(row["TotalServiceInvoices"]+dx, row["AvgLaborSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='yellow', edgecolor='red', arrowstyle='->', linewidth=2.2),
        fontsize=13, color='yellow', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='yellow', alpha=0.85)
    )

for i, (_, row) in enumerate(top3_servinv.iterrows()):
    dx, dy = invoices_offsets[i]
    plt.annotate(
        f"#{i+1} Service Invoices\n{row['CustName']}",
        xy=(row["TotalServiceInvoices"], row["AvgLaborSalesPerInvoice"]),
        xytext=(row["TotalServiceInvoices"]+dx, row["AvgLaborSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='magenta', edgecolor='red', arrowstyle='->', linewidth=2.2),
        fontsize=13, color='magenta', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='magenta', alpha=0.85)
    )

plt.show()
```

```
Average Labor Sales per Service Customer summary:
  Median: $1,920
  Mean:   $2,057
  Min:    $729
  Max:    $5,487
  25th percentile: $1,621
  75th percentile: $2,341

Top 10 Customers by Total Service Invoices:
19. Paramount Transfer          | Service Invoices:  1176 | Avg Labor Sales: $2,028
31. Paramount Logistics         | Service Invoices:   923 | Avg Labor Sales: $1,872
82. Liberty Industries          | Service Invoices:   875 | Avg Labor Sales: $2,088
118. Acme Inc                    | Service Invoices:   847 | Avg Labor Sales: $1,830
146. Stonegate Partners          | Service Invoices:   726 | Avg Labor Sales: $2,366
137. Goldstar Transfer           | Service Invoices:   715 | Avg Labor Sales: $1,635
32. Aurora Inc                  | Service Invoices:   588 | Avg Labor Sales: $1,444
301. Unity Resources             | Service Invoices:   569 | Avg Labor Sales: $2,306
295. Evergreen Associates        | Service Invoices:   557 | Avg Labor Sales: $1,538
353. Acme Systems                | Service Invoices:   463 | Avg Labor Sales: $2,273

Top 10 Customers by Avg Labor Sales per Service Invoice:
303. Pioneer Partners            | Avg Labor Sales: $5,487 | Service Invoices:     7
144. Synergy Solutions           | Avg Labor Sales: $5,406 | Service Invoices:    39
95. Acme Logistics              | Avg Labor Sales: $5,165 | Service Invoices:     1
41. Pioneer Associates          | Avg Labor Sales: $4,464 | Service Invoices:     1
26. Stonegate Truck Lines       | Avg Labor Sales: $4,382 | Service Invoices:   120
317. Keystone Tucking            | Avg Labor Sales: $4,235 | Service Invoices:     6
297. Sterling Systems            | Avg Labor Sales: $4,234 | Service Invoices:   269
36. Paramount Global            | Avg Labor Sales: $3,911 | Service Invoices:     7
240. Synergy Transfer            | Avg Labor Sales: $3,864 | Service Invoices:   215
200. Pioneer Partners            | Avg Labor Sales: $3,862 | Service Invoices:    22

----------------------------------------------------------------------
```

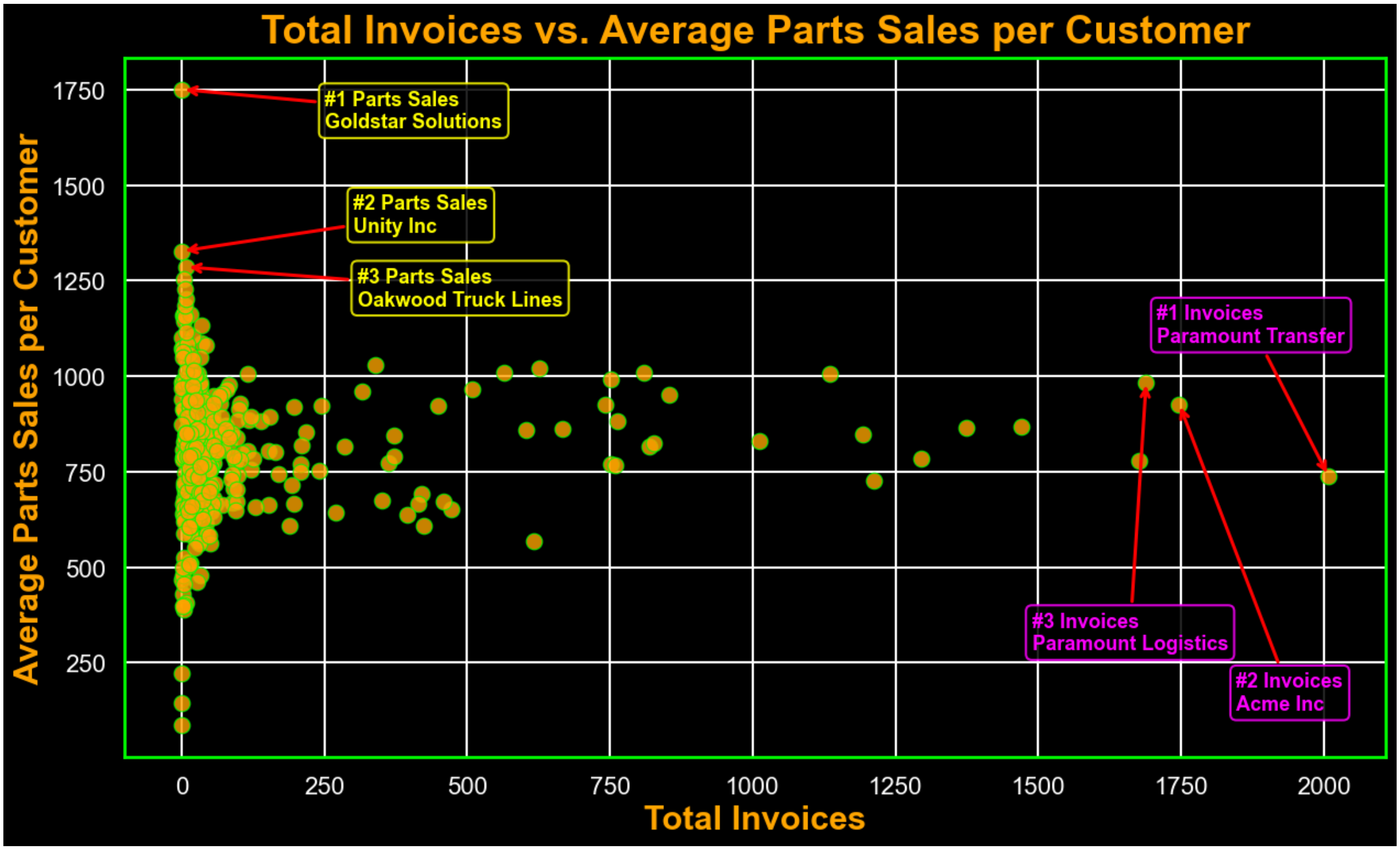


In [24]:
```
# --- Calculate for ALL invoices: Group by customer ---
parts_group = df.groupby(["CustNo", "CustName"]).agg(
    TotalInvoices=("InvoiceNo", "count"),
    AvgPartsSalesPerInvoice=("PartsSales$", "mean"),
).reset_index()

# Merge for annotation (optional)
```

```python
customer_df = customer_df.merge(parts_group, on=["CustNo", "CustName"], suffixes=('', '_all'), how='left')

# Print summaries
parts_median = parts_group["AvgPartsSalesPerInvoice"].median()
parts_mean = parts_group["AvgPartsSalesPerInvoice"].mean()
parts_min = parts_group["AvgPartsSalesPerInvoice"].min()
parts_max = parts_group["AvgPartsSalesPerInvoice"].max()
parts_25 = parts_group["AvgPartsSalesPerInvoice"].quantile(0.25)
parts_75 = parts_group["AvgPartsSalesPerInvoice"].quantile(0.75)

print("Average Parts Sales per Customer summary:")
print(f"  Median: ${parts_median:,.0f}")
print(f"  Mean:   ${parts_mean:,.0f}")
print(f"  Min:    ${parts_min:,.0f}")
print(f"  Max:    ${parts_max:,.0f}")
print(f"  25th percentile: ${parts_25:,.0f}")
print(f"  75th percentile: ${parts_75:,.0f}")

print("\nTop 10 Customers by Total Invoices:")
top10_inv = parts_group.nlargest(10, "TotalInvoices")
for idx, row in top10_inv.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Invoices: {int(row['TotalInvoices']):5d} | Avg Parts Sales: ${row['Av

print("\nTop 10 Customers by Avg Parts Sales per Invoice:")
top10_part_sales = parts_group.nlargest(10, "AvgPartsSalesPerInvoice")
for idx, row in top10_part_sales.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Avg Parts Sales: ${row['AvgPartsSalesPerInvoice']:,.0f} | Invoices: 
print("\n" + "-"*70 + "\n")

# Top 3 by Avg Parts Sales and by Invoices
top3_parts = parts_group.nlargest(3, "AvgPartsSalesPerInvoice")
top3_inv = parts_group.nlargest(3, "TotalInvoices")

plt.figure(figsize=(13, 8), facecolor='black')
ax = sns.scatterplot(
    x="TotalInvoices", y="AvgPartsSalesPerInvoice", data=parts_group,
    color='orange', edgecolor='lime', s=110, alpha=0.80
)
plt.title("Total Invoices vs. Average Parts Sales per Customer", fontsize=26, color='orange', fontweight='bold', pad=
plt.xlabel("Total Invoices", fontsize=22, color='orange', fontweight='bold')
plt.ylabel("Average Parts Sales per Customer", fontsize=22, color='orange', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('lime')
    spine.set_linewidth(2)
plt.tight_layout()

sales_offsets = [(250, -100), (300, 50), (300, -100)]
invoices_offsets = [(-300, 350), (100, -800), (-200, -700)]

for i, (_, row) in enumerate(top3_parts.iterrows()):
    dx, dy = sales_offsets[i]
    plt.annotate(
        f"#{i+1} Parts Sales\n{row['CustName']}",
        xy=(row["TotalInvoices"], row["AvgPartsSalesPerInvoice"]),
        xytext=(row["TotalInvoices"]+dx, row["AvgPartsSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='yellow', edgecolor='red', arrowstyle='->', linewidth=2.2),
        fontsize=13, color='yellow', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='yellow', alpha=0.85)
    )

for i, (_, row) in enumerate(top3_inv.iterrows()):
    dx, dy = invoices_offsets[i]
    plt.annotate(
        f"#{i+1} Invoices\n{row['CustName']}",
        xy=(row["TotalInvoices"], row["AvgPartsSalesPerInvoice"]),
        xytext=(row["TotalInvoices"]+dx, row["AvgPartsSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='magenta', edgecolor='red', arrowstyle='->', linewidth=2.2),
        fontsize=13, color='magenta', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='magenta', alpha=0.85)
    )
```

```
plt.show()
```

```
Average Parts Sales per Customer summary:
  Median: $802
  Mean:   $804
  Min:    $86
  Max:    $1,750
  25th percentile: $697
  75th percentile: $893

Top 10 Customers by Total Invoices:
21. Paramount Transfer         | Invoices:  2008 | Avg Parts Sales: $739
121. Acme Inc                   | Invoices:  1747 | Avg Parts Sales: $927
34. Paramount Logistics        | Invoices:  1689 | Avg Parts Sales: $984
85. Liberty Industries         | Invoices:  1677 | Avg Parts Sales: $777
142. Goldstar Transfer          | Invoices:  1470 | Avg Parts Sales: $867
151. Stonegate Partners         | Invoices:  1375 | Avg Parts Sales: $864
311. Unity Resources            | Invoices:  1295 | Avg Parts Sales: $784
305. Evergreen Associates       | Invoices:  1212 | Avg Parts Sales: $725
35. Aurora Inc                 | Invoices:  1193 | Avg Parts Sales: $846
10. Evergreen LLC              | Invoices:  1136 | Avg Parts Sales: $1,005

Top 10 Customers by Avg Parts Sales per Invoice:
275. Goldstar Solutions         | Avg Parts Sales: $1,750 | Invoices:     1
159. Unity Inc                  | Avg Parts Sales: $1,327 | Invoices:     1
82. Oakwood Truck Lines        | Avg Parts Sales: $1,286 | Invoices:     8
205. Sunset Solutions           | Avg Parts Sales: $1,254 | Invoices:     3
381. Evergreen Tucking          | Avg Parts Sales: $1,228 | Invoices:     5
283. Liberty Inc                | Avg Parts Sales: $1,203 | Invoices:     8
207. Summit Tucking             | Avg Parts Sales: $1,183 | Invoices:     5
37. Evergreen Transfer         | Avg Parts Sales: $1,164 | Invoices:     4
141. Goldstar Global            | Avg Parts Sales: $1,161 | Invoices:    15
16. Stonegate Group            | Avg Parts Sales: $1,158 | Invoices:     2

----------------------------------------------------------------------
```

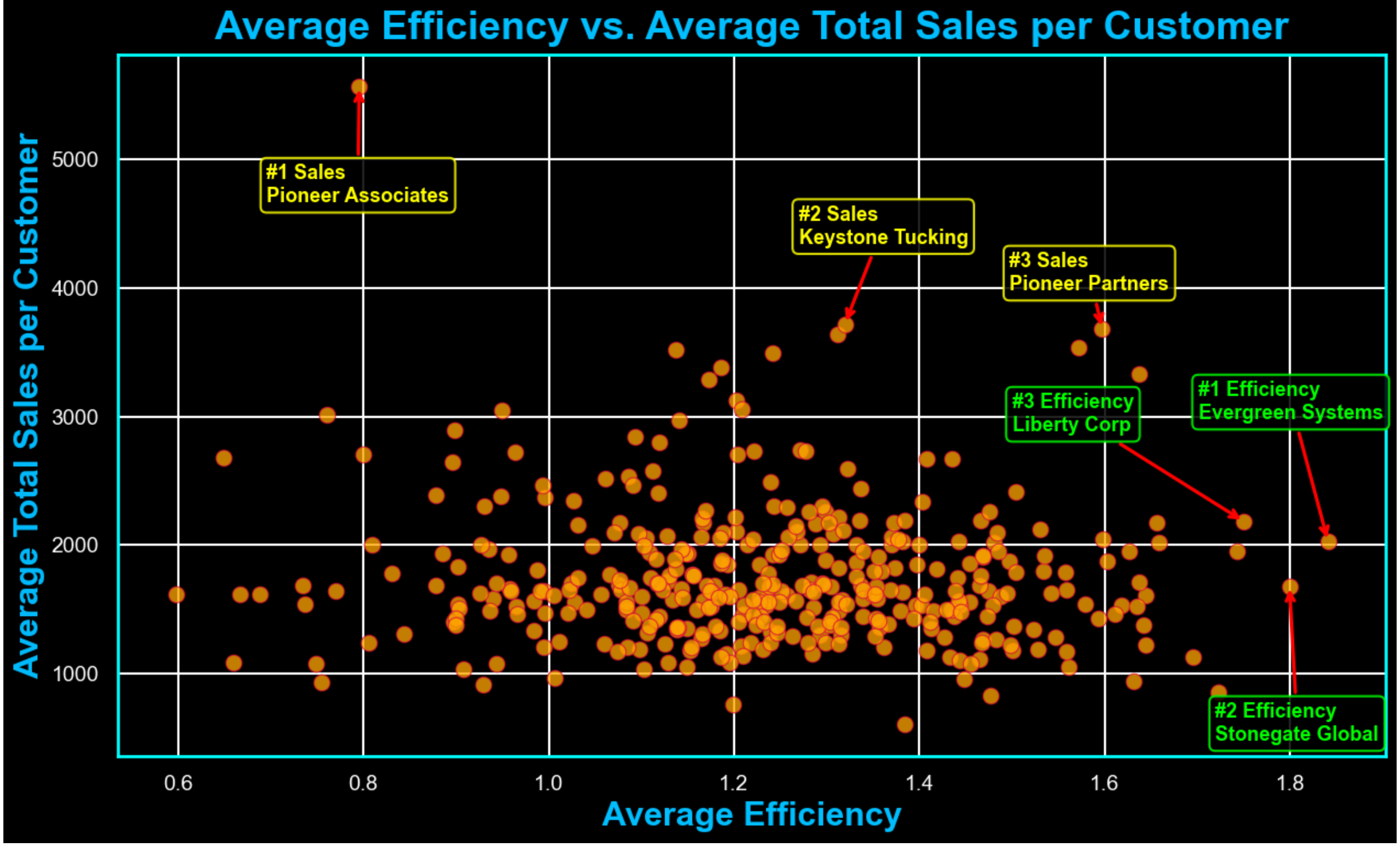


In [25]:
```
# ----- Print summary stats -----
eff_median = customer_df["AvgEfficiency"].median()
eff_mean = customer_df["AvgEfficiency"].mean()
eff_min = customer_df["AvgEfficiency"].min()
eff_max = customer_df["AvgEfficiency"].max()
```

```python
eff_25 = customer_df["AvgEfficiency"].quantile(0.25)
eff_75 = customer_df["AvgEfficiency"].quantile(0.75)

sales_median = customer_df["AvgTotalSalesPerInvoice"].median()
sales_mean = customer_df["AvgTotalSalesPerInvoice"].mean()
sales_min = customer_df["AvgTotalSalesPerInvoice"].min()
sales_max = customer_df["AvgTotalSalesPerInvoice"].max()
sales_25 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.25)
sales_75 = customer_df["AvgTotalSalesPerInvoice"].quantile(0.75)

print("Average Efficiency per Customer summary:")
print(f"  Median: {eff_median:.2f}")
print(f"  Mean:   {eff_mean:.2f}")
print(f"  Min:    {eff_min:.2f}")
print(f"  Max:    {eff_max:.2f}")
print(f"  25th percentile: {eff_25:.2f}")
print(f"  75th percentile: {eff_75:.2f}\n")

print("Average Total Sales per Customer summary:")
print(f"  Median: ${sales_median:,.0f}")
print(f"  Mean:   ${sales_mean:,.0f}")
print(f"  Min:    ${sales_min:,.0f}")
print(f"  Max:    ${sales_max:,.0f}")
print(f"  25th percentile: ${sales_25:,.0f}")
print(f"  75th percentile: ${sales_75:,.0f}")

# Top 10 customers by efficiency and by average sales
print("\nTop 10 Customers by Average Efficiency:")
top10_eff = customer_df.nlargest(10, "AvgEfficiency")
for idx, row in top10_eff.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Efficiency: {row['AvgEfficiency']:.2f} | Avg Total Sales: ${row['Avg

print("\nTop 10 Customers by Average Total Sales per Invoice:")
top10_sales = customer_df.nlargest(10, "AvgTotalSalesPerInvoice")
for idx, row in top10_sales.iterrows():
    print(f"{idx+1:2d}. {row['CustName']:<25} | Avg Total Sales: ${row['AvgTotalSalesPerInvoice']:,.0f} | Efficiency:
print("\n" + "-"*70 + "\n")

# --- Annotate top customers ---
top3_sales = customer_df.nlargest(3, "AvgTotalSalesPerInvoice")
top3_eff = customer_df.nlargest(3, "AvgEfficiency")

plt.figure(figsize=(13, 8), facecolor='black')
ax = sns.scatterplot(
    x="AvgEfficiency", y="AvgTotalSalesPerInvoice", data=customer_df,
    color='orange', edgecolor='crimson', s=110, alpha=0.77
)

plt.title("Average Efficiency vs. Average Total Sales per Customer", fontsize=26, color='deepskyblue', fontweight='bo
plt.xlabel("Average Efficiency", fontsize=22, color='deepskyblue', fontweight='bold')
plt.ylabel("Average Total Sales per Customer", fontsize=22, color='deepskyblue', fontweight='bold')
plt.xticks(fontsize=14, color='white')
plt.yticks(fontsize=14, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('cyan')
    spine.set_linewidth(2)
plt.tight_layout()

# Label offsets (tweak for your data layout)
sales_offsets = [(-0.1, -900), (-.05, 625), (-.1, 300)]
eff_offsets = [(-0.14, 950), (-0.08, -1200), (-.25, 700)]

# Annotate Top 3 by Sales (yellow)
for i, (_, row) in enumerate(top3_sales.iterrows()):
    dx, dy = sales_offsets[i]
    plt.annotate(
        f"#{i+1} Sales\n{row['CustName']}",
        xy=(row["AvgEfficiency"], row["AvgTotalSalesPerInvoice"]),
        xytext=(row["AvgEfficiency"]+dx, row["AvgTotalSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='yellow', edgecolor='red', arrowstyle='->', linewidth=2.3),
        fontsize=13, color='yellow', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='yellow', alpha=0.85)
```

```python
    )

# Annotate Top 3 by Efficiency (lime)
for i, (_, row) in enumerate(top3_eff.iterrows()):
    dx, dy = eff_offsets[i]
    plt.annotate(
        f"#{i+1} Efficiency\n{row['CustName']}",
        xy=(row["AvgEfficiency"], row["AvgTotalSalesPerInvoice"]),
        xytext=(row["AvgEfficiency"]+dx, row["AvgTotalSalesPerInvoice"]+dy),
        arrowprops=dict(facecolor='lime', edgecolor='red', arrowstyle='->', linewidth=2.3),
        fontsize=13, color='lime', fontweight='bold',
        bbox=dict(boxstyle='round', fc='black', ec='lime', alpha=0.85)
    )
plt.show()
```

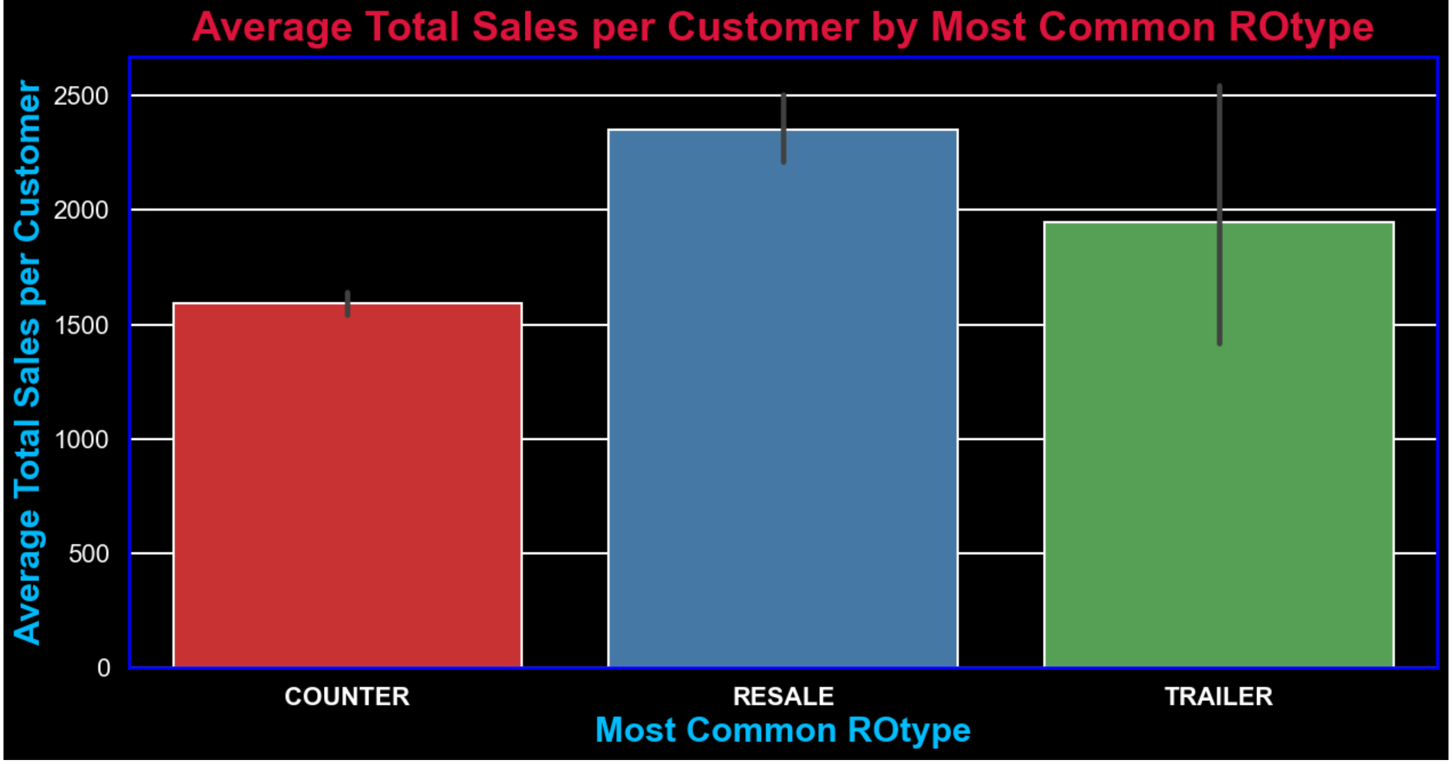
```
Average Efficiency per Customer summary:
  Median: 1.24
  Mean:   1.24
  Min:    0.60
  Max:    1.84
  25th percentile: 1.11
  75th percentile: 1.38

Average Total Sales per Customer summary:
  Median: $1,641
  Mean:   $1,729
  Min:    $86
  Max:    $5,563
  25th percentile: $1,386
  75th percentile: $1,997

Top 10 Customers by Average Efficiency:
125. Evergreen Systems         | Efficiency: 1.84 | Avg Total Sales: $2,029
153. Stonegate Global          | Efficiency: 1.80 | Avg Total Sales: $1,683
111. Liberty Corp              | Efficiency: 1.75 | Avg Total Sales: $2,184
18. Keystone Industries        | Efficiency: 1.74 | Avg Total Sales: $1,956
317. Evergreen LLC             | Efficiency: 1.72 | Avg Total Sales: $857
301. Keystone Inc              | Efficiency: 1.70 | Avg Total Sales: $1,135
226. Keystone Associates       | Efficiency: 1.66 | Avg Total Sales: $2,020
69. Liberty Associates         | Efficiency: 1.66 | Avg Total Sales: $2,176
181. Unity LLC                 | Efficiency: 1.64 | Avg Total Sales: $1,229
51. Evergreen Logistics        | Efficiency: 1.64 | Avg Total Sales: $1,608

Top 10 Customers by Average Total Sales per Invoice:
44. Pioneer Associates         | Avg Total Sales: $5,563 | Efficiency: 0.80
327. Keystone Tucking          | Avg Total Sales: $3,718 | Efficiency: 1.32
313. Pioneer Partners          | Avg Total Sales: $3,681 | Efficiency: 1.60
45. Pioneer Truck Lines        | Avg Total Sales: $3,637 | Efficiency: 1.31
221. Evergreen Transfer        | Avg Total Sales: $3,539 | Efficiency: 1.57
307. Sterling Systems          | Avg Total Sales: $3,520 | Efficiency: 1.14
149. Synergy Solutions         | Avg Total Sales: $3,492 | Efficiency: 1.24
192. Sterling Corp             | Avg Total Sales: $3,383 | Efficiency: 1.19
37. Evergreen Transfer         | Avg Total Sales: $3,329 | Efficiency: 1.64
65. Keystone Global            | Avg Total Sales: $3,287 | Efficiency: 1.17

----------------------------------------------------------------------
```



In [26]:

```python
plt.figure(figsize=(13, 7), facecolor='black')

# Print summary: Average total sales by Most Common ROtype
rotype_means = customer_df.groupby("MostCommonROtype")["AvgTotalSalesPerInvoice"].agg(['mean', 'count']).sort_values(

print("Average Total Sales per Customer by Most Common ROtype:")
for idx, row in rotype_means.iterrows():
    print(f"  {idx:12s} | Avg Sale: ${row['mean']:,.0f} | Customers: {int(row['count'])}")
print()

palette = sns.color_palette("bright", n_colors=customer_df["MostCommonROtype"].nunique())

ax = sns.barplot(
    x="MostCommonROtype",
    y="AvgTotalSalesPerInvoice",
    data=customer_df,
    hue="MostCommonROtype",
    palette="Set1",
    legend=False
)

plt.title("Average Total Sales per Customer by Most Common ROtype", fontsize=26, color='crimson', fontweight='bold',
plt.xlabel("Most Common ROtype", fontsize=22, color='deepskyblue', fontweight='bold')
plt.ylabel("Average Total Sales per Customer", fontsize=22, color='deepskyblue', fontweight='bold')
plt.xticks(fontsize=16, color='white', fontweight='bold')
plt.yticks(fontsize=16, color='white')
ax.set_facecolor('black')
for spine in ax.spines.values():
    spine.set_edgecolor('blue')
    spine.set_linewidth(2)
plt.tight_layout()
plt.show()
```

```
Average Total Sales per Customer by Most Common ROtype:
  RESALE       | Avg Sale: $2,352 | Customers: 67
  TRAILER      | Avg Sale: $1,944 | Customers: 5
  COUNTER      | Avg Sale: $1,593 | Customers: 315
```



Statistical Test: Difference Between RESALE and COUNTER Customers

We use a t-test to see if average total sales differ significantly between RESALE and COUNTER dominant customers.

In [27]:
```
from scipy.stats import ttest_ind

# Filter customers by most common ROtype
resale_sales = customer_df[customer_df["MostCommonROtype"] == "RESALE"]["AvgTotalSalesPerInvoice"]
counter_sales = customer_df[customer_df["MostCommonROtype"] == "COUNTER"]["AvgTotalSalesPerInvoice"]

# Perform two-sample t-test
t_stat, p_value = ttest_ind(resale_sales, counter_sales, nan_policy='omit')
print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: 11.407577033806557
P-value: 4.0199041135806196e-26
```

Prepare Data for Modeling

We select numeric and categorical features, encode categorical fields, and fill any missing values before modeling.

In [28]:
```
# Prepare features for modeling (drop ID columns, encode categoricals)
X = customer_df.drop(columns=["CustNo", "CustName", "AvgTotalSalesPerInvoice"])
X = pd.get_dummies(X, columns=["MostCommonROtype", "MostCommonDept", "MostCommonLocation"], drop_first=True)

# Fill any missing values with zero (safe for dummy/agg columns)
X = X.fillna(0)

y = customer_df["AvgTotalSalesPerInvoice"]
X = X.loc[:, ~X.columns.duplicated()]


# Print statements to confirm prep
print("Feature matrix X shape:", X.shape)
print("Target vector y shape:", y.shape)
print("First 5 rows of feature matrix X:")
print(X.head())
print("\nFeature columns used in modeling:")
print(list(X.columns))
```

```
Feature matrix X shape: (387, 17)
Target vector y shape: (387,)
First 5 rows of feature matrix X:
   TotalInvoices  AvgLaborGM  AvgPartsGM  AvgEfficiency  TotalInvoices_all  \
0            742    0.658874    0.345703       1.370037                742   
1             23    0.574704    0.223554       1.024120                 23   
2             82    0.611540    0.232640       1.076672                 82   
3             13    0.592245    0.323652       1.501356                 13   
4             11    0.756308    0.335894       1.298439                 11   

   AvgPartsSalesPerInvoice  MostCommonROtype_RESALE  MostCommonROtype_TRAILER  \
0               925.104474                    False                     False   
1               675.424783                    False                     False   
2               825.246585                     True                     False   
3               724.214615                    False                     False   
4               731.129091                    False                     False   

   MostCommonDept_20  MostCommonDept_30  MostCommonDept_40  MostCommonDept_50  \
0               True              False              False              False   
1              False              False              False              False   
2              False              False               True              False   
3               True              False              False              False   
4              False              False               True              False   

   MostCommonLocation_Chicago  MostCommonLocation_Dallas  \
0                       False                      False   
1                       False                      False   
2                       False                      False   
3                       False                       True   
4                       False                      False   

   MostCommonLocation_Green Bay  MostCommonLocation_Los Angeles  \
0                         False                            True   
1                          True                           False   
2                         False                           False   
3                         False                           False   
4                          True                           False   

   MostCommonLocation_New York  
0                        False  
1                        False  
2                        False  
3                        False  
4                        False  

Feature columns used in modeling:
['TotalInvoices', 'AvgLaborGM', 'AvgPartsGM', 'AvgEfficiency', 'TotalInvoices_all', 'AvgPartsSalesPerInvoice', 'MostC
ommonROtype_RESALE', 'MostCommonROtype_TRAILER', 'MostCommonDept_20', 'MostCommonDept_30', 'MostCommonDept_40', 'Most
CommonDept_50', 'MostCommonLocation_Chicago', 'MostCommonLocation_Dallas', 'MostCommonLocation_Green Bay', 'MostCommo
nLocation_Los Angeles', 'MostCommonLocation_New York']
```

Train/Test Split

We split the customer data into training and test sets.

In [29]:
```python
from sklearn.model_selection import train_test_split

# Split data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("Train/Test Split Results:")
print(f"  X_train shape: {X_train.shape}")
print(f"  X_test shape:  {X_test.shape}")
print(f"  y_train shape: {y_train.shape}")
print(f"  y_test shape:  {y_test.shape}")
print(f"  Training set percent: {100*len(X_train)/len(X):.1f}%")
print(f"  Test set percent:     {100*len(X_test)/len(X):.1f}%")
```

```
Train/Test Split Results:
  X_train shape: (309, 17)
  X_test shape:  (78, 17)
  y_train shape: (309,)
  y_test shape:  (78,)
  Training set percent: 79.8%
  Test set percent:     20.2%
```

Model 1: Linear Regression

We begin with a linear regression baseline and report its metrics.

In [30]:
```
# Import model and metrics for linear regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# If available, import the new RMSE function for future compatibility
try:
    from sklearn.metrics import root_mean_squared_error
    use_new_rmse = True
except ImportError:
    use_new_rmse = False

# Train a linear regression model on the training data
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lr = lr.predict(X_test)

# Evaluate model performance with common regression metrics:
# R2: How well the model explains variance (1 = perfect, 0 = average)
# MAE: Average prediction error in the same units as the target (dollars)
# RMSE: Root mean squared error; larger errors penalized more heavily.
# Note: As of scikit-learn 1.4, the preferred way to compute RMSE is with root_mean_squared_error().
# Older code often uses mean_squared_error(..., squared=False), which is being deprecated.

print("------ Linear Regression Model Performance ------")
print(f"  Features used: {X_train.shape[1]}")
print(f"  Test set size: {len(X_test)} customers\n")
print(f"  R² (Test):      {r2_score(y_test, y_pred_lr):.6f}")
print(f"  MAE (Test):     ${mean_absolute_error(y_test, y_pred_lr):,.2f}")
if use_new_rmse:
    print(f"  RMSE (Test):    ${root_mean_squared_error(y_test, y_pred_lr):,.2f}")
else:
    print(f"  RMSE (Test):    ${mean_squared_error(y_test, y_pred_lr, squared=False):,.2f}")
print("\n-------------------------------------------------")
```

```
------ Linear Regression Model Performance ------
  Features used: 17
  Test set size: 78 customers

  R² (Test):      0.567384
  MAE (Test):     $236.47
  RMSE (Test):    $302.15

-------------------------------------------------
```

Model 2: Random Forest Regressor

Next, we use a random forest to capture nonlinear effects and feature interactions.

In [31]:
```
# Import model and metrics for random forest regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

try:
    from sklearn.metrics import root_mean_squared_error
    use_new_rmse = True
except ImportError:
```

```
    use_new_rmse = False

# Train the random forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict on test set
y_pred_rf = rf.predict(X_test)

# Prepare output
n_features = X_train.shape[1]
n_test = len(y_test)
r2 = r2_score(y_test, y_pred_rf)
mae = mean_absolute_error(y_test, y_pred_rf)
rmse = root_mean_squared_error(y_test, y_pred_rf) if use_new_rmse else mean_squared_error(y_test, y_pred_rf, squared=

print("------ Random Forest Model Performance ------")
print(f"  Features used: {n_features}")
print(f"  Test set size: {n_test} customers\n")
print(f"  R² (Test):      {r2:.6f}")
print(f"  MAE (Test):     ${mae:,.2f}")
print(f"  RMSE (Test):    ${rmse:,.2f}\n")
print("------------------------------------------------")
```

```
------ Random Forest Model Performance ------
  Features used: 17
  Test set size: 78 customers

  R² (Test):      0.525416
  MAE (Test):     $250.68
  RMSE (Test):    $316.47

------------------------------------------------
```

In [32]:
```
# Model 3: XGBoost Regressor

try:
    from xgboost import XGBRegressor
    from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb.fit(X_train, y_train)
    y_pred_xgb = xgb.predict(X_test)

    try:
        from sklearn.metrics import root_mean_squared_error
        rmse = root_mean_squared_error(y_test, y_pred_xgb)
    except ImportError:
        rmse = mean_squared_error(y_test, y_pred_xgb, squared=False)

    n_features = X_train.shape[1]
    n_test = len(y_test)
    r2 = r2_score(y_test, y_pred_xgb)
    mae = mean_absolute_error(y_test, y_pred_xgb)

    print("------ XGBoost Model Performance ------")
    print(f"  Features used: {n_features}")
    print(f"  Test set size: {n_test} customers\n")
    print(f"  R² (Test):      {r2:.6f}")
    print(f"  MAE (Test):     ${mae:,.2f}")
    print(f"  RMSE (Test):    ${rmse:,.2f}\n")
    print("------------------------------------------------")
except ImportError:
    print("XGBoost is not installed. Please install xgboost using pip or conda and restart the kernel.")
```

```
------ XGBoost Model Performance ------
  Features used: 17
  Test set size: 78 customers

  R² (Test):       0.426515
  MAE (Test):      $269.12
  RMSE (Test):     $347.88

-------------------------------------------------
```

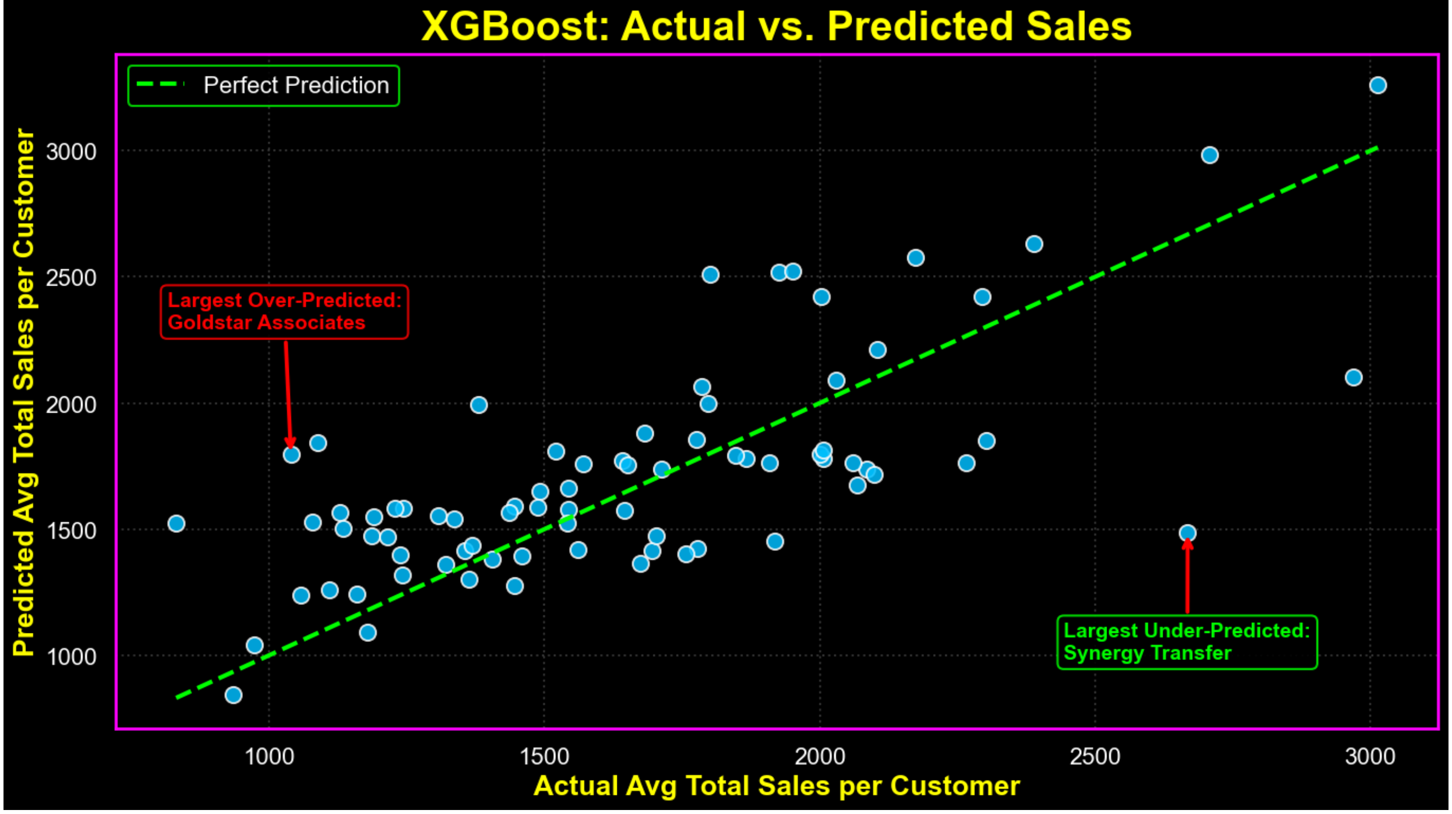
Model Evaluation: Visualizations

We visualize predicted vs actual and residuals for the best-performing model.

In [33]:
```python
# Use predictions from your best model
if 'y_pred_xgb' in locals():
    y_pred = y_pred_xgb
    model_label = "XGBoost"
else:
    y_pred = y_pred_rf
    model_label = "Random Forest"

# ---- Get test set metadata ----
# If you already have test_idx from your train_test_split, otherwise get indices another way
try:
    test_idx = X_test.index
except:
    test_idx = customer_df.index[-len(y_test):]  # fallback for default split

test_customers = customer_df.loc[test_idx].copy()
test_customers["Actual"] = y_test
test_customers["Predicted"] = y_pred
test_customers["Error"] = test_customers["Predicted"] - test_customers["Actual"]

# 3. Print top 10 largest positive and negative errors (over/under)
largest_pos = test_customers.nlargest(10, "Error")
largest_neg = test_customers.nsmallest(10, "Error")

print(f"{model_label} Model: Actual vs. Predicted Sales Plot")
print(f"  Test set size: {len(y_test)} customers")
print(f"  Actual sales range: ${y_test.min():,.0f} to ${y_test.max():,.0f}")
print(f"  Predicted sales range: ${y_pred.min():,.0f} to ${y_pred.max():,.0f}")
print(f"  Mean Absolute Error: ${mean_absolute_error(y_test, y_pred):,.2f}")
print(f"  R² Score: {r2_score(y_test, y_pred):.3f}\n")

print("Top 10 Largest Positive Prediction Errors (Over-predicted):")
for _, row in largest_pos.iterrows():
    print(f"{row['CustName']:<24} | Predicted: ${row['Predicted']:,.0f} | Actual: ${row['Actual']:,.0f} | Error: +${r

print("\nTop 10 Largest Negative Prediction Errors (Under-predicted):")
for _, row in largest_neg.iterrows():
    print(f"{row['CustName']:<24} | Predicted: ${row['Predicted']:,.0f} | Actual: ${row['Actual']:,.0f} | Error: ${ro

# ---- Plot ----
plt.figure(figsize=(14, 8), facecolor='black')
plt.scatter(
    y_test, y_pred,
    alpha=0.85,
    s=120,
    c='deepskyblue',
    edgecolor='white',
    linewidth=1.4
)
# Reference line (perfect prediction)
plt.plot(
    [y_test.min(), y_test.max()],
    [y_test.min(), y_test.max()],
    color='lime', linestyle='--', linewidth=3, label="Perfect Prediction"
)

plt.title(f"{model_label}: Actual vs. Predicted Sales", fontsize=28, color='yellow', fontweight='bold', pad=10)
```

```python
plt.xlabel("Actual Avg Total Sales per Customer", fontsize=19, color='yellow', fontweight='bold')
plt.ylabel("Predicted Avg Total Sales per Customer", fontsize=19, color='yellow', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
plt.grid(True, linestyle=':', color='gray', alpha=0.4)
plt.gca().set_facecolor('black')
for spine in plt.gca().spines.values():
    spine.set_edgecolor('magenta')
    spine.set_linewidth(2)
plt.legend(facecolor='black', edgecolor='lime', fontsize=16, loc='upper left', labelcolor='white')

# 1: Red arrow for largest positive error
row_pos = largest_pos.iloc[0]
plt.annotate(
    f"Largest Over-Predicted:\n{row_pos['CustName']}",
    xy=(row_pos["Actual"], row_pos["Predicted"]),
    xytext=(row_pos["Actual"] - 225, row_pos["Predicted"] + 500),
    arrowprops=dict(facecolor='red', edgecolor='red', arrowstyle='->', lw=2.7),
    fontsize=14, color='red', fontweight='bold',
    bbox=dict(boxstyle='round', fc='black', ec='red', alpha=0.85)
)

# 2: Red arrow for largest negative error
row_neg = largest_neg.iloc[0]
plt.annotate(
    f"Largest Under-Predicted:\n{row_neg['CustName']}",
    xy=(row_neg["Actual"], row_neg["Predicted"]),
    xytext=(row_neg["Actual"] - 225, row_neg["Predicted"] - 500),
    arrowprops=dict(facecolor='black', edgecolor='red', arrowstyle='->', lw=2.7),
    fontsize=14, color='lime', fontweight='bold',
    bbox=dict(boxstyle='round', fc='black', ec='lime', alpha=0.85)
)

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

```
XGBoost Model: Actual vs. Predicted Sales Plot
  Test set size: 78 customers
  Actual sales range: $832 to $3,013
  Predicted sales range: $844 to $3,262
  Mean Absolute Error: $269.12
  R² Score: 0.427

Top 10 Largest Positive Prediction Errors (Over-predicted):
Goldstar Associates      | Predicted: $1,796 | Actual: $1,040 | Error: +$756
Unity Global             | Predicted: $1,842 | Actual: $1,089 | Error: +$753
Sunset Truck Lines       | Predicted: $2,509 | Actual: $1,801 | Error: +$707
Evergreen Corp           | Predicted: $1,523 | Actual: $832 | Error: +$691
Evergreen Global         | Predicted: $1,995 | Actual: $1,381 | Error: +$614
Stonegate Industries     | Predicted: $2,520 | Actual: $1,927 | Error: +$593
Paramount LLC            | Predicted: $2,522 | Actual: $1,950 | Error: +$572
Aurora Partners          | Predicted: $1,528 | Actual: $1,080 | Error: +$448
Stonegate Tucking        | Predicted: $1,564 | Actual: $1,128 | Error: +$436
Summit Solutions         | Predicted: $2,421 | Actual: $2,002 | Error: +$419

Top 10 Largest Negative Prediction Errors (Under-predicted):
Synergy Transfer         | Predicted: $1,487 | Actual: $2,668 | Error: $-1,181
Goldstar Global          | Predicted: $2,104 | Actual: $2,968 | Error: $-864
Sterling Transfer        | Predicted: $1,765 | Actual: $2,266 | Error: $-501
Keystone Logistics       | Predicted: $1,452 | Actual: $1,918 | Error: $-466
Synergy Industries       | Predicted: $1,850 | Actual: $2,303 | Error: $-453
Sunset Tucking           | Predicted: $1,674 | Actual: $2,069 | Error: $-395
Aurora Group             | Predicted: $1,717 | Actual: $2,099 | Error: $-383
Sterling Group           | Predicted: $1,422 | Actual: $1,778 | Error: $-355
Aurora Transfer          | Predicted: $1,402 | Actual: $1,756 | Error: $-354
Sterling Associates      | Predicted: $1,737 | Actual: $2,085 | Error: $-349
```

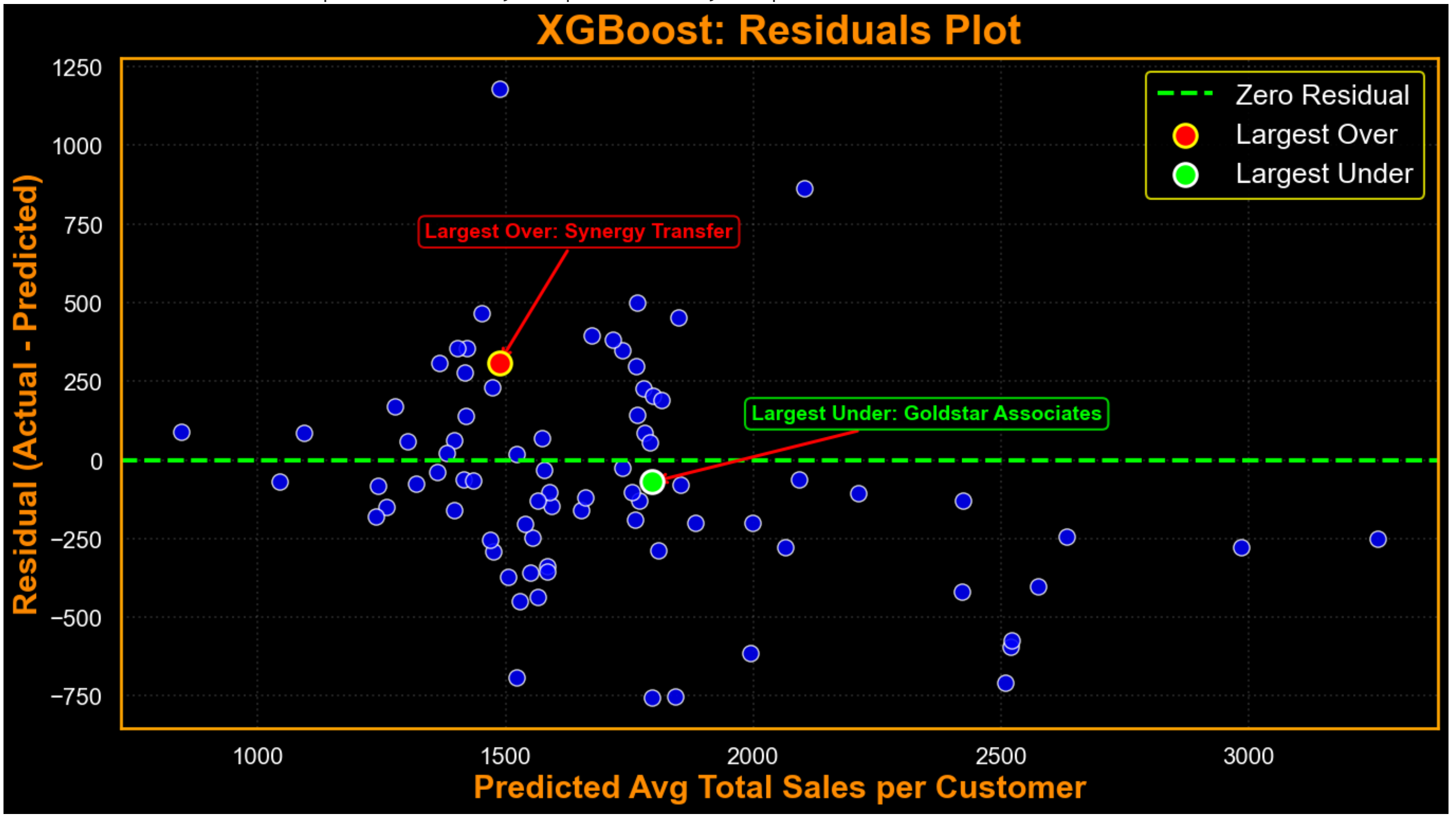


In [34]:

```
from sklearn.metrics import mean_absolute_error, r2_score

# --- Use predictions from your best model (assumes y_pred_xgb, y_test, customer_df, X_test defined) ---
if 'y_pred_xgb' in locals():
    y_pred = y_pred_xgb
    model_label = "XGBoost"
else:
    y_pred = y_pred_rf
    model_label = "Random Forest"

# --- Prepare residuals and test customer metadata ---
residuals = y_test - y_pred

# If you have test_idx from train_test_split, use it; else fallback:
try:
    test_idx = X_test.index
except:
    test_idx = customer_df.index[-len(y_test):]

test_customers = customer_df.loc[test_idx].copy()
test_customers["Actual"] = y_test
test_customers["Predicted"] = y_pred
test_customers["Residual"] = residuals

# Top 10 over- and under-predicted
largest_pos = test_customers.nlargest(10, "Residual")
largest_neg = test_customers.nsmallest(10, "Residual")

print(f"{model_label} Model: Residuals Plot")
print(f"  Test set size: {len(y_test)} customers")
print(f"  Mean Absolute Error: ${mean_absolute_error(y_test, y_pred):,.2f}")
print(f"  R² Score: {r2_score(y_test, y_pred):.3f}\n")

print("Top 10 Largest Positive Residuals (Actual > Predicted):")
for _, row in largest_pos.iterrows():
    print(f"{row['CustName']:<24} | Predicted: ${row['Predicted']:,.0f} | Actual: ${row['Actual']:,.0f} | Residual: -

print("\nTop 10 Largest Negative Residuals (Actual < Predicted):")
for _, row in largest_neg.iterrows():
    print(f"{row['CustName']:<24} | Predicted: ${row['Predicted']:,.0f} | Actual: ${row['Actual']:,.0f} | Residual: $

# --- Residuals Plot (visual) ---
plt.figure(figsize=(14, 8), facecolor='black')
```

```python
plt.scatter(
    y_pred, residuals,
    alpha=0.85, s=120, c='blue',
    edgecolor='white', linewidth=1.0
)
plt.axhline(0, color='lime', linestyle='--', linewidth=3, label="Zero Residual")

# Highlight largest positive/negative residuals
idx_max = np.argmax(residuals)
idx_min = np.argmin(residuals)
plt.scatter(
    y_pred[idx_max], residuals[idx_max],
    s=240, c='red', edgecolor='yellow', linewidth=2.2, marker='o', zorder=10, label='Largest Over'
)
plt.scatter(
    y_pred[idx_min], residuals[idx_min],
    s=240, c='lime', edgecolor='white', linewidth=2.2, marker='o', zorder=10, label='Largest Under'
)

# Annotate
plt.annotate(
    f"Largest Over: {test_customers.iloc[idx_max]['CustName']}",
    xy=(y_pred[idx_max], residuals[idx_max]),
    xytext=(y_pred[idx_max] - 150, residuals[idx_max] + 400),
    arrowprops=dict(facecolor='red', edgecolor='red', arrowstyle='->', lw=2.2),
    fontsize=14, color='red', fontweight='bold',
    bbox=dict(boxstyle='round', fc='black', ec='red', alpha=0.82)
)
plt.annotate(
    f"Largest Under: {test_customers.iloc[idx_min]['CustName']}",
    xy=(y_pred[idx_min], residuals[idx_min]),
    xytext=(y_pred[idx_min] + 200, residuals[idx_min] + 200),
    arrowprops=dict(facecolor='red', edgecolor='red', arrowstyle='->', lw=2.2),
    fontsize=14, color='lime', fontweight='bold',
    bbox=dict(boxstyle='round', fc='black', ec='lime', alpha=0.82)
)

plt.title(f"{model_label}: Residuals Plot", fontsize=28, color='darkorange', fontweight='bold', pad=10)
plt.xlabel("Predicted Avg Total Sales per Customer", fontsize=22, color='darkorange', fontweight='bold')
plt.ylabel("Residual (Actual - Predicted)", fontsize=22, color='darkorange', fontweight='bold')
plt.xticks(fontsize=16, color='white')
plt.yticks(fontsize=16, color='white')
plt.grid(True, linestyle=':', color='gray', alpha=0.3)
plt.gca().set_facecolor('black')
for spine in plt.gca().spines.values():
    spine.set_edgecolor('orange')
    spine.set_linewidth(2)
plt.legend(facecolor='black', edgecolor='yellow', fontsize=19, loc='upper right', labelcolor='white')
plt.tight_layout()
plt.show()
```

```
XGBoost Model: Residuals Plot
  Test set size: 78 customers
  Mean Absolute Error: $269.12
  R² Score: 0.427

Top 10 Largest Positive Residuals (Actual > Predicted):
Synergy Transfer          | Predicted: $1,487 | Actual: $2,668 | Residual: +$1,181
Goldstar Global           | Predicted: $2,104 | Actual: $2,968 | Residual: +$864
Sterling Transfer         | Predicted: $1,765 | Actual: $2,266 | Residual: +$501
Keystone Logistics        | Predicted: $1,452 | Actual: $1,918 | Residual: +$466
Synergy Industries        | Predicted: $1,850 | Actual: $2,303 | Residual: +$453
Sunset Tucking            | Predicted: $1,674 | Actual: $2,069 | Residual: +$395
Aurora Group              | Predicted: $1,717 | Actual: $2,099 | Residual: +$383
Sterling Group            | Predicted: $1,422 | Actual: $1,778 | Residual: +$355
Aurora Transfer           | Predicted: $1,402 | Actual: $1,756 | Residual: +$354
Sterling Associates       | Predicted: $1,737 | Actual: $2,085 | Residual: +$349

Top 10 Largest Negative Residuals (Actual < Predicted):
Goldstar Associates       | Predicted: $1,796 | Actual: $1,040 | Residual: $-756
Unity Global              | Predicted: $1,842 | Actual: $1,089 | Residual: $-753
Sunset Truck Lines        | Predicted: $2,509 | Actual: $1,801 | Residual: $-707
Evergreen Corp            | Predicted: $1,523 | Actual: $832 | Residual: $-691
Evergreen Global          | Predicted: $1,995 | Actual: $1,381 | Residual: $-614
Stonegate Industries      | Predicted: $2,520 | Actual: $1,927 | Residual: $-593
Paramount LLC             | Predicted: $2,522 | Actual: $1,950 | Residual: $-572
Aurora Partners           | Predicted: $1,528 | Actual: $1,080 | Residual: $-448
Stonegate Tucking         | Predicted: $1,564 | Actual: $1,128 | Residual: $-436
Summit Solutions          | Predicted: $2,421 | Actual: $2,002 | Residual: $-419
```



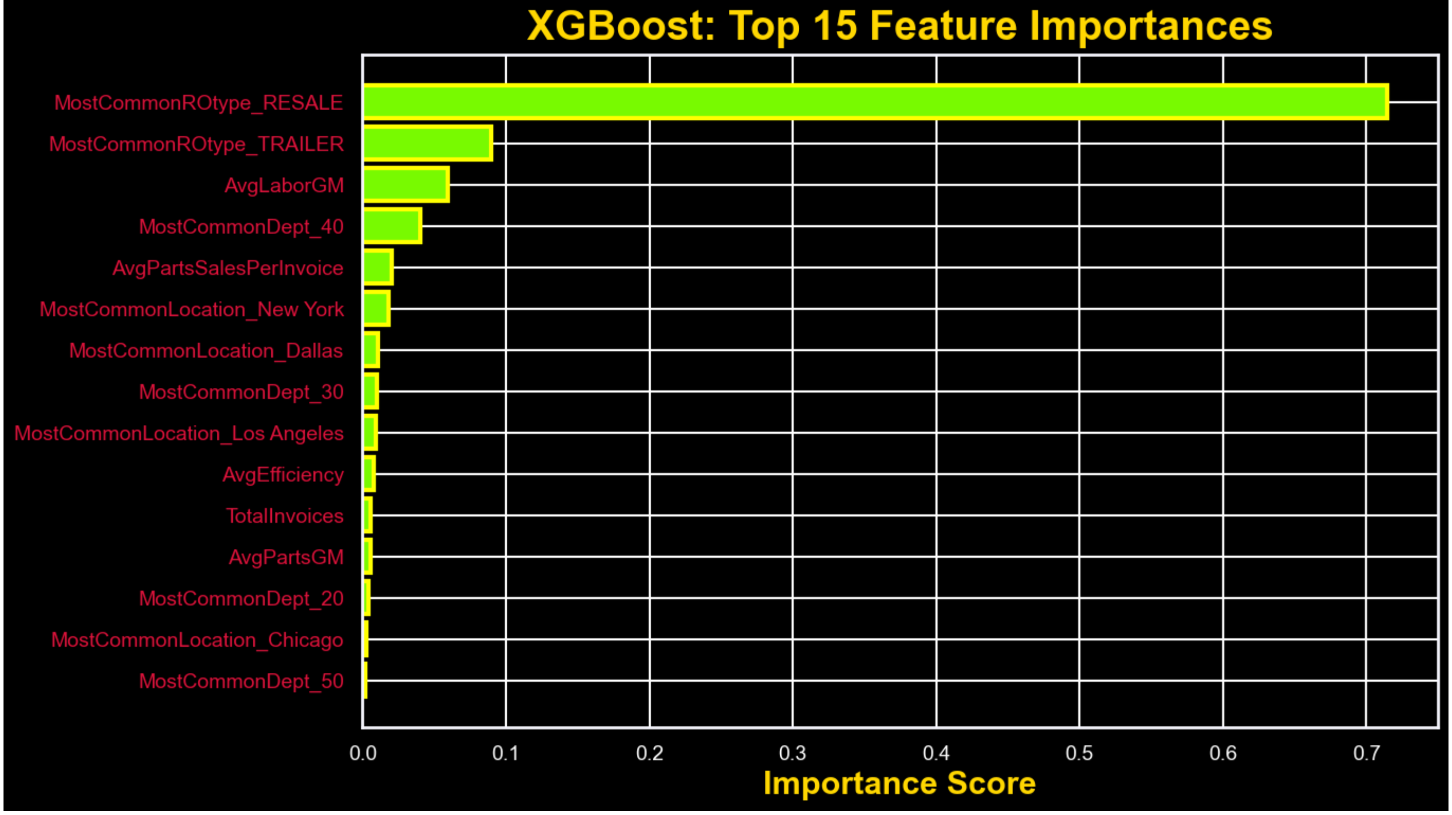
Feature Importance

We visualize which features most influence the model's predictions.

In [35]:
```python
# Determine which model to use for feature importance
if 'xgb' in locals():
    importances = xgb.feature_importances_
    feature_names = X.columns
    model_name = "XGBoost"
elif 'rf' in locals():
    importances = rf.feature_importances_
    feature_names = X.columns
    model_name = "Random Forest"
else:
```

```
    importances = None
    feature_names = []

if importances is not None:
    # Get top N features for readability
    N = 15
    indices = np.argsort(importances)[::-1][:N]
    top_features = [(feature_names[i], importances[i]) for i in indices]

    # Print the top 10 features with importances
    print(f"{model_name} Model: Top 10 Feature Importances")
    for i, (feat, imp) in enumerate(top_features[:10], 1):
        print(f"{i:2d}. {feat:<35} | Importance: {imp:.4f}")
    print()

    # Plot
    plt.figure(figsize=(14, 8), facecolor='black')
    bars = plt.barh(
        range(N),
        [importances[i] for i in indices],
        color='lawngreen',
        edgecolor='yellow',
        linewidth=3
    )
    plt.yticks(range(N), [feature_names[i] for i in indices], fontsize=14, color='crimson')
    plt.gca().invert_yaxis()
    plt.xlabel('Importance Score', fontsize=22, color='gold', fontweight='bold')
    plt.title(f'{model_name}: Top {N} Feature Importances', fontsize=28, color='gold', fontweight='bold', pad=10)
    plt.xticks(fontsize=14, color='white')
    plt.gca().set_facecolor('black')
    for spine in plt.gca().spines.values():
        spine.set_edgecolor('ghostwhite')
        spine.set_linewidth(2)
    plt.tight_layout()
    plt.show()
else:
    print("Feature importance not available.")
```

```
XGBoost Model: Top 10 Feature Importances
 1. MostCommonROtype_RESALE             | Importance: 0.7141
 2. MostCommonROtype_TRAILER            | Importance: 0.0900
 3. AvgLaborGM                          | Importance: 0.0593
 4. MostCommonDept_40                   | Importance: 0.0404
 5. AvgPartsSalesPerInvoice             | Importance: 0.0201
 6. MostCommonLocation_New York         | Importance: 0.0178
 7. MostCommonLocation_Dallas           | Importance: 0.0105
 8. MostCommonDept_30                   | Importance: 0.0099
 9. MostCommonLocation_Los Angeles      | Importance: 0.0090
10. AvgEfficiency                       | Importance: 0.0080
```



Conclusion

This project built and evaluated multiple regression models to predict average total sales per dealership customer using operational and categorical data. The analysis confirmed that customer segmentation, especially by job type (RESALE vs COUNTER), is the strongest driver of sales. Simpler models like linear regression performed just as well as advanced methods for this dataset, with an R^2 around 0.57 and a mean absolute error near $236.

The XGBoost model, while robust, did not improve accuracy, likely due to the linear nature of business relationships in this data. Feature importance analysis reinforced that customer type, labor gross margin, and specific departments or locations are most predictive of customer value.

Business teams should use these insights to focus on growing the RESALE segment, increasing service work among parts-heavy customers, and monitoring key accounts for changes in purchasing behavior. Outlier accounts identified by the model offer immediate opportunities for review, engagement, or risk management.

With more real-world data or additional features, model accuracy may improve further. This project provides a solid foundation for data-driven customer segmentation and growth planning in dealership operations.