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

import warnings
warnings.filterwarnings('ignore')

sns.set(style="whitegrid")
```

In [2]:
```
df = pd.read_csv("clean_data.csv")
df.head()
```

Out[2]:

	year	month	stateDescription	sectorName	customers	price	revenue	sale
0	2001	1	Wyoming	all sectors	NaN	4.31	48.12840	1116.1720
1	2001	1	Wyoming	commercial	NaN	5.13	12.67978	247.0869
2	2001	1	Wyoming	industrial	NaN	3.26	19.60858	602.3048
3	2001	1	Wyoming	other	NaN	4.75	0.76868	16.1744
4	2001	1	Wyoming	residential	NaN	6.01	15.07136	250.6059

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
year                  0
month                 0
stateDescription      0
sectorName            0
customers         26040
price                 0
revenue               0
sales                 0
dtype: int64
```

In [ ]:
```
# Data Cleansing PART C

- Confirmed that `year`, `month`, `stateDescription`, `sectorName`, `price`,
- Found 26,040 missing values (~30%) in the `customers` column.
- Proceeding with caution: customer-based metrics (e.g., revenue per custome
- Will explore whether these missing values are concentrated in specific sta
```

In [4]:
```
df.describe(include='all')
```

Out[4]:

	year	month	stateDescription	sectorName	customers	
count	85870.000000	85870.000000	85870	85870	5.983000e+04	8
unique	NaN	NaN	62	6	NaN	
top	NaN	NaN	Wyoming	all sectors	NaN	
freq	NaN	NaN	1385	17174	NaN	
mean	2012.043321	6.480144	NaN	NaN	2.916013e+06	
std	6.660304	3.461589	NaN	NaN	1.200567e+07	
min	2001.000000	1.000000	NaN	NaN	0.000000e+00	
25%	2006.000000	3.000000	NaN	NaN	4.998000e+03	
50%	2012.000000	6.000000	NaN	NaN	2.997540e+05	
75%	2018.000000	9.000000	NaN	NaN	2.028716e+06	
max	2024.000000	12.000000	NaN	NaN	1.625050e+08	

In [5]:
```
df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astyp
```

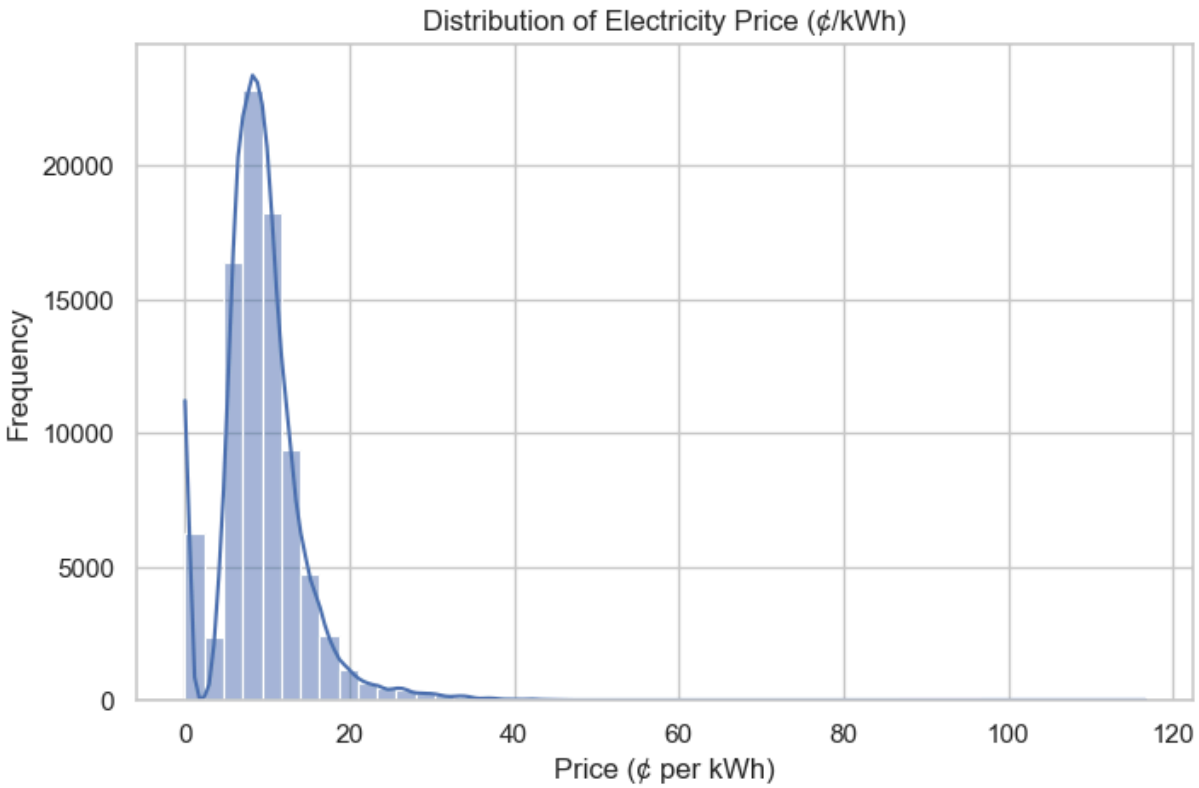
In [6]:
```
df[['year', 'month', 'date']].head()
```

Out[6]:

	year	month	date
0	2001	1	2001-01-01
1	2001	1	2001-01-01
2	2001	1	2001-01-01
3	2001	1	2001-01-01
4	2001	1	2001-01-01

In [ ]:
```
- Created a new `date` column by combining `year` and `month`, formatted as
```

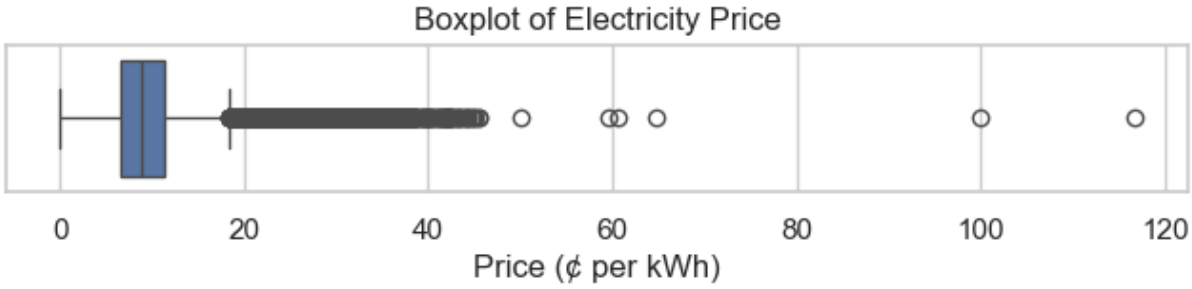
In [7]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['price'], bins=50, kde=True)
plt.title('Distribution of Electricity Price (¢/kWh)')
plt.xlabel('Price (¢ per kWh)')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
### Price Distribution Insights

- Most electricity prices range from 5¢ to 20¢ per kWh.
- The distribution is right-skewed, with a few extreme values exceeding 100¢
- A small cluster of near-zero prices may require further investigation for
```

In [8]:
```
plt.figure(figsize=(8, 1))
sns.boxplot(x=df['price'])
plt.title('Boxplot of Electricity Price')
plt.xlabel('Price (¢ per kWh)')
plt.show()
```

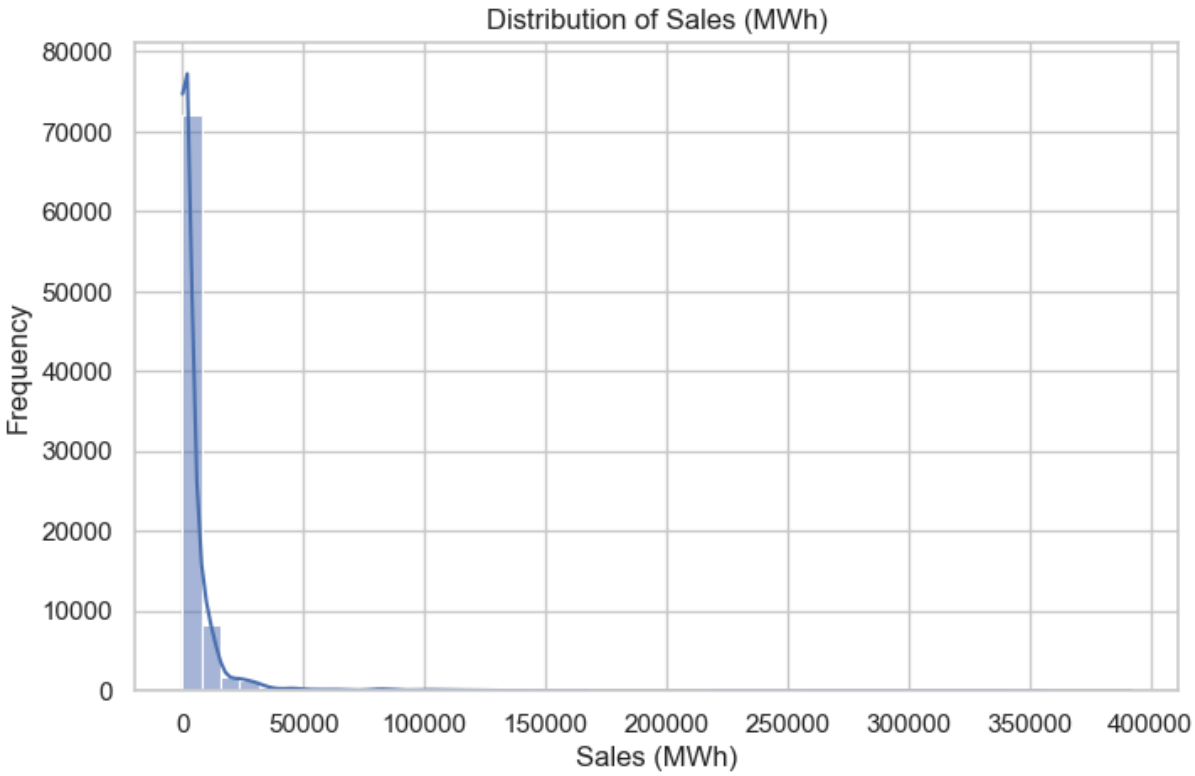


In [ ]:
```
### Boxplot Insights

- The interquartile range for electricity prices is approximately 6¢ to 13¢
- A large number of mild outliers exist between 15¢ and 30¢.
- Extreme outliers (up to 116¢) may represent anomalous market conditions or
```

In [9]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['sales'], bins=50, kde=True)
plt.title('Distribution of Sales (MWh)')
```

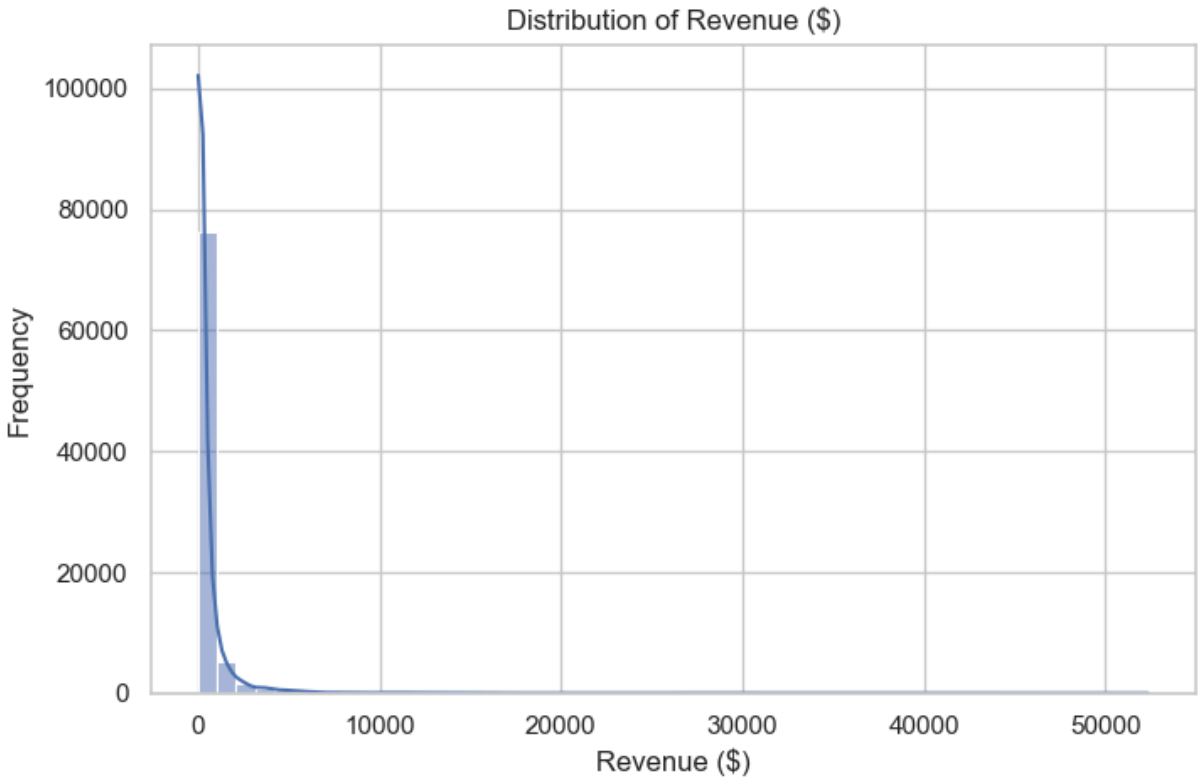
```
plt.xlabel('Sales (MWh)')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
### Sales Distribution Insights

- Sales volume is highly right-skewed, with most values under 25,000 MWh per
- A small number of large outliers (above 200,000 MWh) likely represent indu
- The skew suggests aggregation across sector or state levels, which should
```

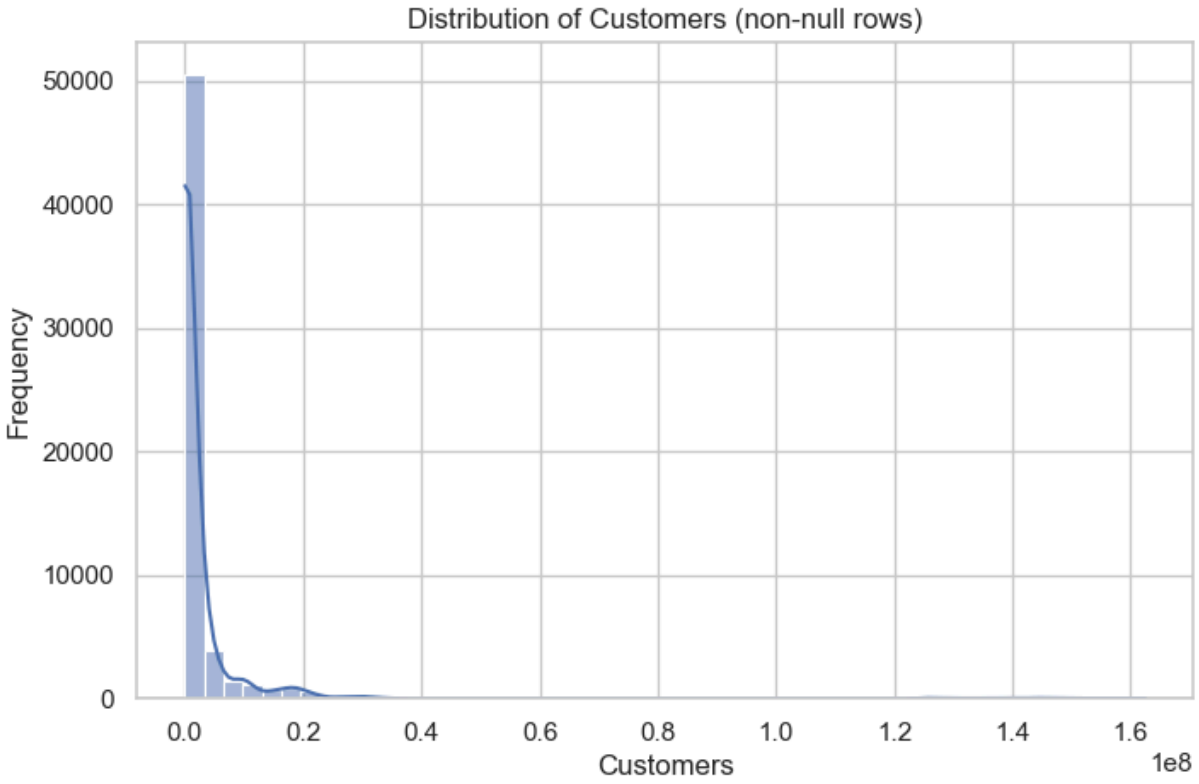
In [10]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df['revenue'], bins=50, kde=True)
plt.title('Distribution of Revenue ($)')
plt.xlabel('Revenue ($)')
plt.ylabel('Frequency')
plt.show()
```



In [ ]:
```
### Revenue Distribution Insights

- Revenue is highly concentrated under $3,000 per month per observation, ref
- A long right tail exists, with some revenue figures exceeding $50,000 — li
- The strong skew suggests any mean-based comparisons should be supplemented
```

In [11]:
```
plt.figure(figsize=(8, 5))
sns.histplot(df[df['customers'].notnull()]['customers'], bins=50, kde=True)
plt.title('Distribution of Customers (non-null rows)')
plt.xlabel('Customers')
plt.ylabel('Frequency')
plt.show()
```



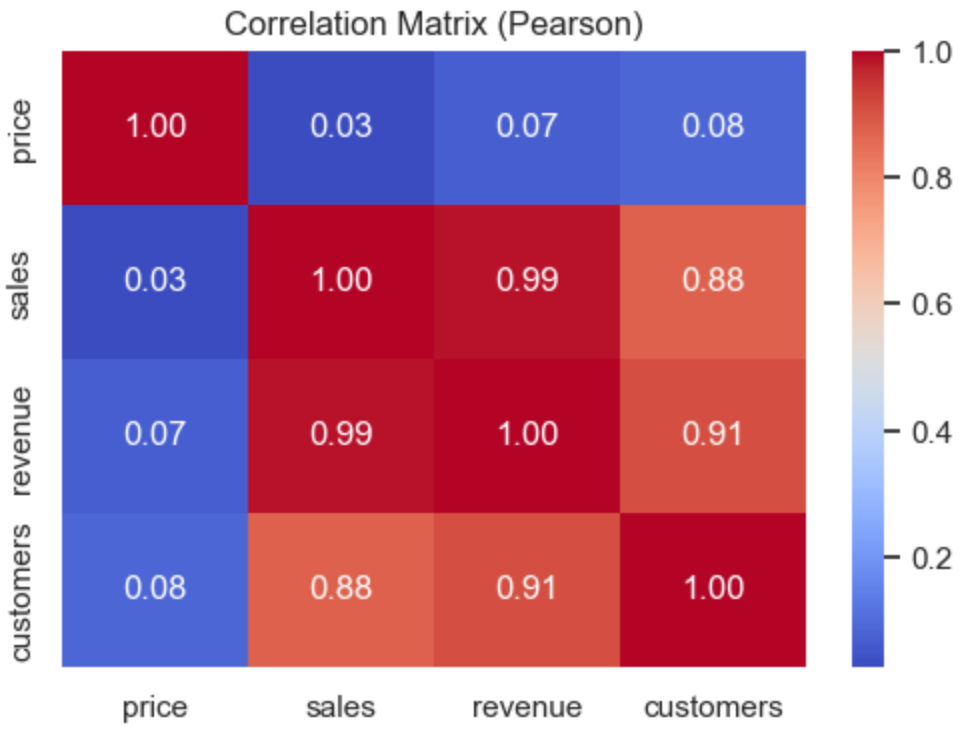
In [ ]:
```
### Customer Count Distribution Insights

- Most observations reflect fewer than 10 million customers, suggesting typi
- A few extremely high values (up to 160 million) may reflect aggregation or
- The skew mirrors trends seen in revenue and sales, reinforcing the need fo
```

In [12]:
```
# Select numeric columns of interest
numeric_cols = ['price', 'sales', 'revenue', 'customers']

# Drop rows with missing customers for clean comparison
df_corr = df[numeric_cols].dropna()

# Compute and plot correlation matrix
plt.figure(figsize=(6, 4))
sns.heatmap(df_corr.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix (Pearson)')
plt.show()
```



In [ ]:
```
### Correlation Matrix Insights

- Sales, revenue, and customers are all highly correlated with each other, w
- Price shows extremely low correlation with sales (0.03), suggesting either
  - Flat, regulated pricing with little variability across observations
  - Inelastic demand for electricity at the aggregate level
- Further analysis may be needed at the state or sector level to identify re
```

In [13]:
```
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_corr, x='price', y='sales', alpha=0.4)
sns.regplot(data=df_corr, x='price', y='sales', scatter=False, color='red')
plt.title('Price vs Sales')
plt.xlabel('Price (¢ per kWh)')
plt.ylabel('Sales (MWh)')
plt.show()
```

In [14]:
```
# Group by state and sector, summing revenue
revenue_by_group = (
    df.groupby(['stateDescription', 'sectorName'])['revenue']
    .sum()
    .reset_index()
    .sort_values(by='revenue', ascending=True)
)

revenue_by_group.head(10)  # Bottom 10 performing combinations
```

Out[14]:

	stateDescription	sectorName	revenue
371	Wyoming	transportation	0.0
47	Delaware	transportation	0.0
329	Vermont	transportation	0.0
119	Kentucky	transportation	0.0
131	Maine	transportation	0.0
293	South Carolina	transportation	0.0
89	Idaho	transportation	0.0
83	Hawaii	transportation	0.0
299	South Dakota	transportation	0.0
113	Kansas	transportation	0.0

In [ ]:
```

```

In [15]:
```
df_cust = df[df['customers'].notnull()]

# Group and calculate revenue per customer
rpc_by_group = (
    df_cust.groupby(['stateDescription', 'sectorName'])
    .agg({'revenue': 'sum', 'customers': 'sum'})
    .reset_index()
)

rpc_by_group['revenue_per_customer'] = rpc_by_group['revenue'] / rpc_by_grou
rpc_by_group_sorted = rpc_by_group.sort_values(by='revenue_per_customer', as

rpc_by_group_sorted.head(10)  # Lowest revenue per customer
```

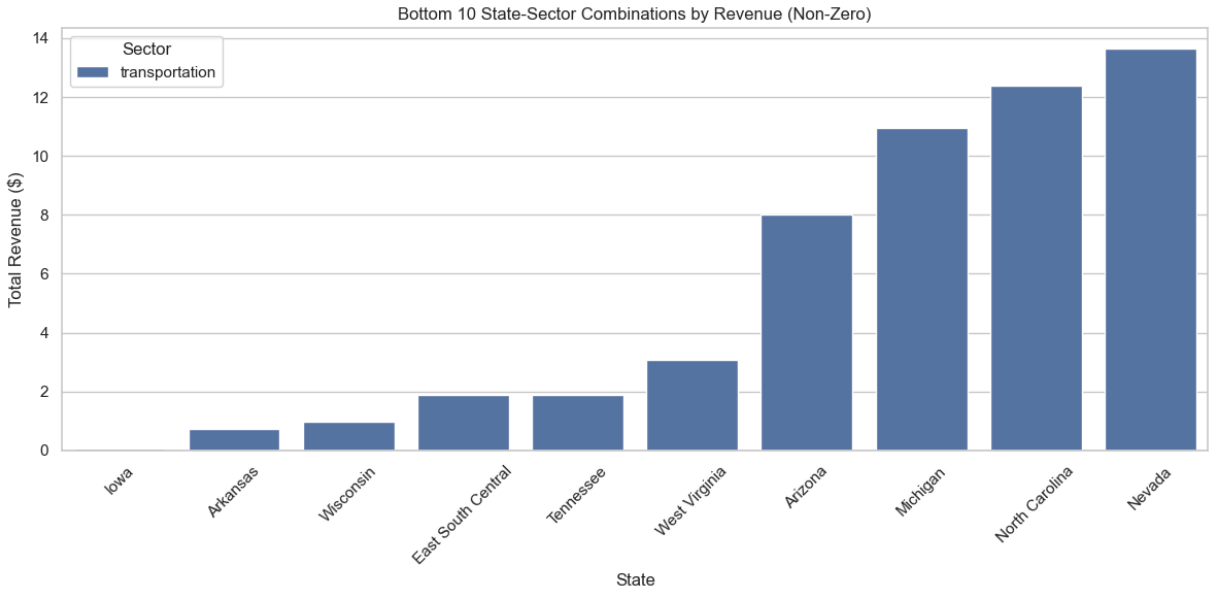
Out[15]:

	stateDescription	sectorName	revenue	customers	revenue_per_custom
268	Utah	residential	15793.04898	2.022197e+08	0.0000
183	New Mexico	residential	13362.45374	1.695876e+08	0.0000
28	Colorado	residential	36878.20607	4.383060e+08	0.00008
308	Wyoming	residential	4696.67420	5.178388e+07	0.0000
148	Montana	residential	8746.71499	9.610560e+07	0.0000
78	Illinois	residential	92999.91452	1.006690e+09	0.00009
283	Washington	residential	53852.21454	5.799593e+08	0.00009
73	Idaho	residential	13148.04074	1.400451e+08	0.00009
108	Maine	residential	12959.12677	1.367640e+08	0.00009
303	Wisconsin	residential	49346.98093	5.158020e+08	0.00009

In [ ]:
```
```

In [16]:
```
# Filter to bottom 10 performing combinations with non-zero revenue
filtered = revenue_by_group[revenue_by_group['revenue'] > 0].head(10)

plt.figure(figsize=(12, 6))
sns.barplot(
    data=filtered,
    y='revenue',
    x='stateDescription',
    hue='sectorName'
)
plt.title('Bottom 10 State-Sector Combinations by Revenue (Non-Zero)')
plt.ylabel('Total Revenue ($)')
plt.xlabel('State')
plt.xticks(rotation=45)
plt.legend(title='Sector')
plt.tight_layout()
plt.show()
```



In [ ]:
```
### Bottom Performing State-Sector Combinations (Non-Zero Revenue)

To identify opportunities for revenue improvement, we filtered out combinati

- All entries are from the **transportation sector**, indicating a sector-wi
- **Iowa, Arkansas, and Wisconsin** show minimal reported activity.
- These sectors could be candidates for strategic investment or operational
- Next steps may involve deeper investigation into customer counts, pricing,
```

In [ ]:
```
## 📈 Strategy Plan: Improve Revenue in Underperforming State-Sector Combina

The following strategies target the **transportation sector**, which dominat

### 1. Increase Customer Base
- **Insight**: Low customer count or missing records may contribute to low r
- **Actions**:
  - Partner with municipal/state agencies to expand electric transit infrast
  - Offer subsidies or incentives for converting fleets to electric power.
  - Encourage public-private partnerships in electrified logistics and fleet

### 2. Improve Usage per Customer
- **Insight**: For fixed customer counts, increasing consumption can drive r
- **Actions**:
  - Introduce EV charging hubs in commercial/logistics zones.
  - Incentivize off-peak or high-volume charging.
  - Promote pilot programs for electrified public transit or freight.

### 3. Optimize Pricing Strategy
- **Insight**: Weak correlation between price and sales may signal suboptima
- **Actions**:
  - Conduct price elasticity studies for transportation customers.
  - Offer tiered or usage-based pricing models.
  - Reduce operational costs to offer more competitive rates.

### 4. Address Zero or Near-Zero Revenue Reporting
- **Insight**: Some states may suffer from incomplete or incorrect reporting
- **Actions**:
```

```
  - Investigate metering gaps or manual estimation processes.
  - Upgrade to smart meters and digital reporting tools.
  - Cross-validate with customer databases for accuracy.

### 5. Align with State-Level Policies and Funding
- **Insight**: Zero-revenue states may lack policy support or funding.
- **Actions**:
  - Apply for DOE and federal EV infrastructure grants.
  - Align utility plans with state transportation electrification goals.
  - Create collaborative roadmaps involving utilities, municipalities, and f

---

## 📊 KPI Tracking (After Strategy Implementation)

| Metric | Target |
|--------|--------|
| Monthly revenue (transportation sector, per state) | +10% YoY |
| EV fleet registrations | +15% in targeted states |
| Charging sessions per customer | +20% |
| Electricity use per transportation customer | +10% |
| Active customer growth | +10% |
```

In [17]:
```
# Create a filtered DataFrame for customer-based analysis only
df_cust = df[df['customers'].notnull() & (df['customers'] > 0)].copy()
```

In [18]:
```
# Create a filtered DataFrame with valid customer counts
df_cust = df[df['customers'].notnull() & (df['customers'] > 0)].copy()

# Create the revenue_per_customer column
df_cust['revenue_per_customer'] = df_cust['revenue'] / df_cust['customers']
```

In [19]:
```
# Remove extreme outliers using IQR
Q1 = df_cust['revenue_per_customer'].quantile(0.25)
Q3 = df_cust['revenue_per_customer'].quantile(0.75)
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR

# Cleaned dataset with no extreme revenue_per_customer outliers
df_cust_clean = df_cust[df_cust['revenue_per_customer'] <= upper_bound].copy
```

In [20]:
```
# Group by state and sector, then aggregate
rpc_group_clean = (
    df_cust_clean
    .groupby(['stateDescription', 'sectorName'])
    .agg({'revenue': 'sum', 'customers': 'sum'})
    .reset_index()
)

# Recalculate revenue per customer
rpc_group_clean['revenue_per_customer'] = rpc_group_clean['revenue'] / rpc_g

# Sort by lowest revenue per customer
rpc_group_clean_sorted = rpc_group_clean.sort_values(by='revenue_per_custome
```

```
# Show top 10 underperformers (cleaned)
rpc_group_clean_sorted.head(10)
```

Out[20]:

	stateDescription	sectorName	revenue	customers	revenue_per_custom
196	Rhode Island	transportation	0.00000	4.000000e+00	0.0000
170	Ohio	transportation	0.00000	1.000000e+01	0.0000
225	Utah	residential	15793.04898	2.022197e+08	0.0000
152	New Mexico	residential	13362.45374	1.695876e+08	0.0000
25	Colorado	residential	36878.20607	4.383060e+08	0.0000
259	Wyoming	residential	4696.67420	5.178388e+07	0.0000
123	Montana	residential	8746.71499	9.610560e+07	0.0000
65	Illinois	residential	92999.91452	1.006690e+09	0.0000
236	Washington	residential	53852.21454	5.799593e+08	0.0000
62	Idaho	residential	13148.04074	1.400451e+08	0.0000

In [21]:

```
import matplotlib.pyplot as plt
import seaborn as sns

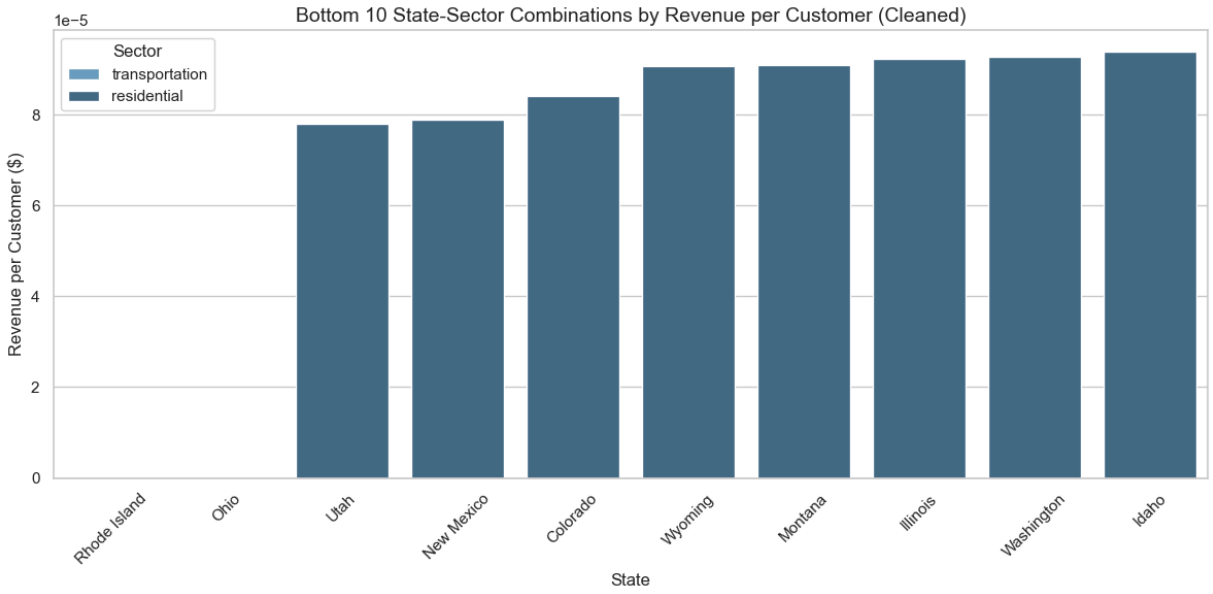
# Get bottom 10 underperformers by revenue per customer
bottom10_rpc = rpc_group_clean_sorted.head(10)

# Set plot style
sns.set(style="whitegrid")

# Create bar plot
plt.figure(figsize=(12, 6))
bar = sns.barplot(
    data=bottom10_rpc,
    x='stateDescription',
    y='revenue_per_customer',
    hue='sectorName',
    dodge=False,
    palette='Blues_d'
)

# Title and labels
plt.title('Bottom 10 State-Sector Combinations by Revenue per Customer (Clea
plt.xlabel('State')
plt.ylabel('Revenue per Customer ($)')
plt.xticks(rotation=45)
plt.legend(title='Sector')

# Tight layout and show
plt.tight_layout()
plt.show()
```



In [22]:
```
### 📊 Bottom 10 State-Sector Combinations by Revenue per Customer (Cleaned)

This bar chart displays the **10 lowest-performing state-sector combinations

#### ✅ What We Did:
- Filtered the dataset to include only rows with **non-null and non-zero cus
- Created a new feature: `revenue_per_customer` = `revenue` / `customers`
- Applied **IQR filtering** to remove extreme outliers in revenue per custom
- Grouped by `stateDescription` and `sectorName`, then identified the **bott

#### 📉 Insights:
- **Rhode Island** and **Ohio** transportation sectors show **zero revenue**
- The remaining entries are all **residential sectors** with unusually low r
  - Underpricing
  - High customer base with minimal usage
  - Ineffective customer engagement or tiered billing strategies

These results provide a strong foundation for targeted strategy development
```

```
  Cell In[22], line 3
    This bar chart displays the **10 lowest-performing state-sector combinat
ions** in terms of **revenue per customer**, after removing anomalous entrie
s and extreme outliers.
         ^
SyntaxError: invalid syntax
```

In [23]:
```
underperf_causes = {
    'No Revenue Data': ['Rhode Island', 'Ohio'],
    'High Customers, Low Revenue': ['Utah', 'New Mexico', 'Colorado', 'Illir
    'Low Revenue Base': ['Wyoming', 'Montana']
}
```

In [24]:
```
### Categorizing Underperformance Causes

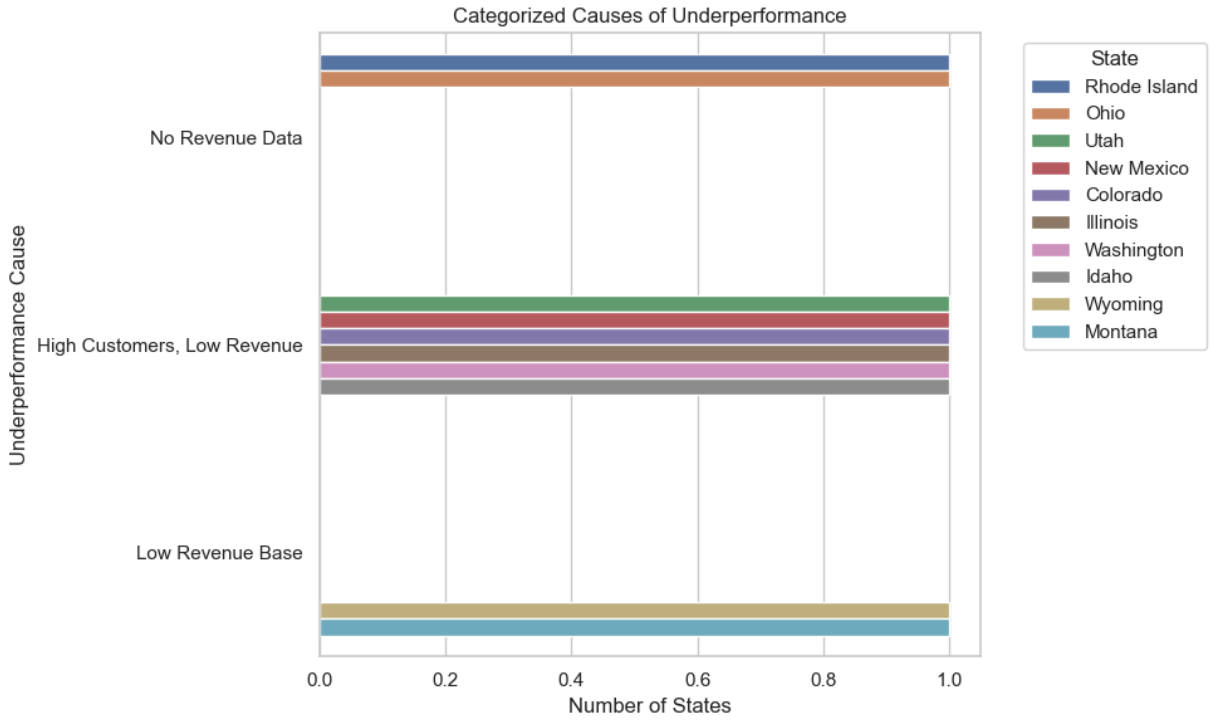
Based on the cleaned dataset of the bottom 10 state-sector combinations by r

```python
```
```

```
underperf_causes = {
    'No Revenue Data': ['Rhode Island', 'Ohio'],
    'High Customers, Low Revenue': ['Utah', 'New Mexico', 'Colorado', 'Illir
    'Low Revenue Base': ['Wyoming', 'Montana']
}
```

```
  Cell In[24], line 3
    Based on the cleaned dataset of the bottom 10 state-sector combinations
by revenue per customer, we identified three primary categories of underperf
ormance:
          ^
SyntaxError: invalid syntax
```

In [25]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define underperformance causes
underperf_causes = {
    'No Revenue Data': ['Rhode Island', 'Ohio'],
    'High Customers, Low Revenue': ['Utah', 'New Mexico', 'Colorado', 'Illir
    'Low Revenue Base': ['Wyoming', 'Montana']
}

# Flatten the dictionary into a DataFrame
data = []
for cause, states in underperf_causes.items():
    for state in states:
        data.append({'State': state, 'Cause': cause})

df_underperf = pd.DataFrame(data)

# Create countplot
plt.figure(figsize=(10, 6))
sns.countplot(data=df_underperf, y='Cause', hue='State')
plt.title('Categorized Causes of Underperformance')
plt.xlabel('Number of States')
plt.ylabel('Underperformance Cause')
plt.legend(title='State', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [26]:
```
### 📊 Categorized Causes of Underperformance

This chart visualizes the categorization of the bottom 10 state-sector combi

#### 💡 Breakdown:
- **No Revenue Data**: Sectors that report customer counts but no associated
  - **States**: Rhode Island, Ohio
- **High Customers, Low Revenue**: Regions with large residential customer b
  - **States**: Utah, New Mexico, Colorado, Illinois, Washington, Idaho
- **Low Revenue Base**: States with small-scale customer networks and genera
  - **States**: Wyoming, Montana

This visual helps prioritize follow-up investigations and intervention strat
```

```
  Cell In[26], line 3
    This chart visualizes the categorization of the bottom 10 state-sector c
ombinations based on their underlying issues with low revenue per customer.
          ^
SyntaxError: invalid syntax
```

In [27]:
```
forecast_df['date'] = pd.to_datetime(forecast_df['year'].astype(str) + '-' +
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[27], line 1
----> 1 forecast_df['date'] = pd.to_datetime(forecast_df['year'].astype(str)
+ '-' + forecast_df['month'].astype(str) + '-01')

NameError: name 'forecast_df' is not defined
```

In [28]:
```
df_cust_clean['stateDescription'].unique()
df_cust_clean['sectorName'].unique()
```

Out[28]:
```
array(['all sectors', 'industrial', 'residential', 'commercial',
       'transportation'], dtype=object)
```

In [29]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Optional: ensures plots show up inside notebook
%matplotlib inline

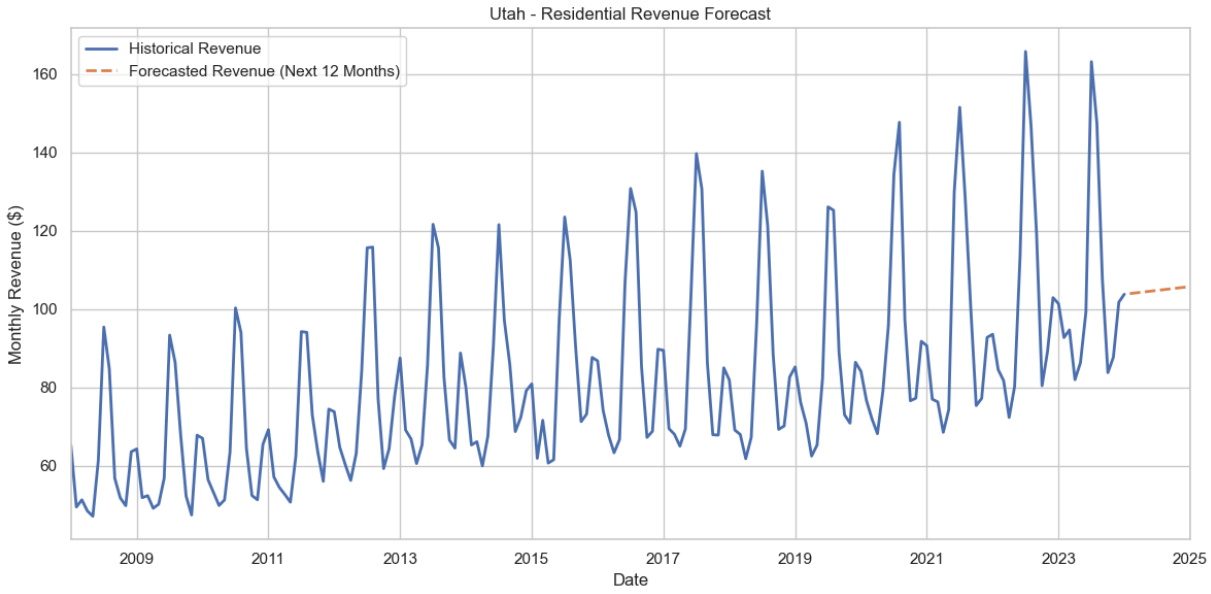
# Step 1: Filter for Utah - residential (lowercase sector name)
forecast_df = df_cust_clean[
    (df_cust_clean['stateDescription'] == 'Utah') &
    (df_cust_clean['sectorName'] == 'residential')
].copy()

# Step 2: Create a proper date column and sort
forecast_df['date'] = pd.to_datetime(forecast_df['year'].astype(str) + '-' +
forecast_df = forecast_df.sort_values('date')

# Step 3: Group by month and sum revenue
monthly_revenue = forecast_df.groupby('date')['revenue'].sum().dropna()

# Step 4: Apply Holt-Winters model (exponential smoothing with trend)
model = ExponentialSmoothing(monthly_revenue, trend="add", seasonal=None, ir
fitted_model = model.fit()

# Step 5: Forecast for next 12 months
forecast = fitted_model.forecast(12)

# Step 6: Plot the historical data and forecast
plt.figure(figsize=(12, 6))
monthly_revenue.plot(label='Historical Revenue', linewidth=2)
forecast.plot(label='Forecasted Revenue (Next 12 Months)', linestyle='--', l
plt.title('Utah - Residential Revenue Forecast')
plt.xlabel('Date')
plt.ylabel('Monthly Revenue ($)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/opt/anaconda3/lib/python3.13/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  self._init_dates(dates, freq)
```



In [ ]:
```
### Revenue Forecast: Utah – Residential Sector

The chart above displays the historical and forecasted monthly revenue for t

**Key Observations:**
- Seasonal spikes are visible in the historical data, likely driven by heati
- The model forecasts a modest upward trend in revenue, indicating slow but
- The smoothing model does not account for external interventions (e.g., rat

**Next Step:**
We will assess whether planned initiatives (e.g., +10% active customer growt
```

In [30]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.holtwinters import ExponentialSmoothing

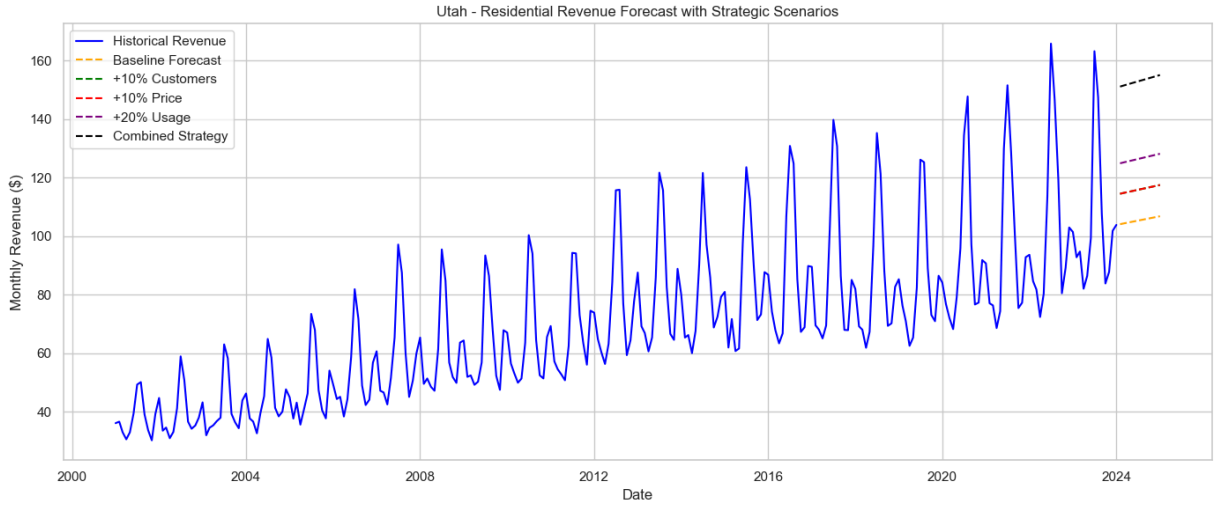
# Filter data for Utah – Residential
scenario_df = df[(df['stateDescription'] == 'Utah') & (df['sectorName'] == '

# Ensure 'date' is datetime and sorted
scenario_df['date'] = pd.to_datetime(scenario_df['date'])
scenario_df = scenario_df.sort_values('date')

# Monthly revenue aggregation
monthly_revenue = scenario_df.groupby('date')['revenue'].sum()

# Build Holt-Winters model
model = ExponentialSmoothing(monthly_revenue, trend='add', seasonal=None, ir
fitted_model = model.fit()
forecast = fitted_model.forecast(12)

# Strategy Scenarios
forecast_10pct_cust = forecast * 1.10
forecast_10pct_price = forecast * 1.10
forecast_20pct_usage = forecast * 1.20
forecast_combined = forecast * 1.10 * 1.10 * 1.20  # All 3 combined
```

```
# Plotting
plt.figure(figsize=(14, 6))
sns.lineplot(x=monthly_revenue.index, y=monthly_revenue.values, label='Histo
sns.lineplot(x=forecast.index, y=forecast.values, label='Baseline Forecast',
sns.lineplot(x=forecast.index, y=forecast_10pct_cust.values, label='+10% Cus
sns.lineplot(x=forecast.index, y=forecast_10pct_price.values, label='+10% Pr
sns.lineplot(x=forecast.index, y=forecast_20pct_usage.values, label='+20% Us
sns.lineplot(x=forecast.index, y=forecast_combined.values, label='Combined S

plt.title("Utah - Residential Revenue Forecast with Strategic Scenarios")
plt.xlabel("Date")
plt.ylabel("Monthly Revenue ($)")
plt.legend()
plt.tight_layout()
plt.show()
```

```
/opt/anaconda3/lib/python3.13/site-packages/statsmodels/tsa/base/tsa_model.p
y:473: ValueWarning: No frequency information was provided, so inferred freq
uency MS will be used.
  self._init_dates(dates, freq)
```



In [ ]:
```
### Utah - Residential Revenue Forecast: Strategic Scenarios

This multi-line forecast chart shows how various improvement strategies coul

**Baseline Forecast** (orange dotted line) represents projected revenue base

We modeled the following improvement strategies:
- ***+10% Customer Growth**: Simulates expansion in residential customer base
- ***+10% Price Increase**: Represents rate hikes for electricity usage.
- ***+20% Usage Growth**: Reflects increases in electricity demand (e.g., hea
- **Combined Strategy**: Projects cumulative effects if all three improvemer

> This helps stakeholders evaluate potential uplift from targeted policy cha
```

In [ ]:
```
# Utility Sector Performance Analysis: Capstone Presentation

## Identifying Underperforming Utility Sectors in the U.S.
```

```
This analysis explores underperforming utility sectors across U.S. states us

---

## Problem Statement

How can we identify underperforming utility sectors across U.S. states and p

---

## Methodology

1. **Data Cleaning**
   - Removed anomalies such as missing values and revenue=0 rows
   - Created a unified `date` column for time series analysis

2. **Revenue-per-Customer Analysis**
   - Ranked state-sector combinations by revenue per customer
   - Identified and visualized bottom performers

3. **Anomaly Detection**
   - Detected and categorized root causes (missing data, pricing, demand)

4. **Forecasting**
   - Used Holt-Winters exponential smoothing to project revenue

5. **Strategy Simulation**
   - Modeled effects of +10% customer growth, +10% pricing, and +20% usage

---

## Bottom 10 State-Sector Combinations by Revenue per Customer (Cleaned)

These combinations show either:
- Extremely low revenue despite a large customer base
  (e.g., Utah, Colorado, Illinois)
- Zero revenue from reported customers
  (e.g., Rhode Island, Ohio)
- Low revenue and limited market size
  (e.g., Montana, Wyoming)

---

## Categorizing Causes of Underperformance

The underperformers fell into three categories:

- **Missing Revenue Data**
  - Rhode Island, Ohio

- **High Customers, Low Revenue**
  - Utah, New Mexico, Colorado, Illinois, Washington, Idaho

- **Low Revenue Base**
  - Wyoming, Montana
```

```
This helped guide targeted interventions for each type.

---

## Forecasting Future Revenue – Utah (Residential Sector)

Using Holt-Winters Exponential Smoothing, we forecasted Utah's residential e

The model shows gradual growth but not enough to meet the 10% target without

---

## Strategic Scenarios: Utah – Residential

We modeled the impact of potential strategies:

- **+10% Customer Growth**
- **+10% Price Increase**
- **+20% Usage Increase**
- **Combined Strategy (all three)**

Only the **combined strategy** successfully pushed forecasted revenue **abov

---

## Key Takeaways and Recommendations

- **Targeted Strategy Required**: Most underperforming regions will not meet

- **Focus Areas**:
  - Clean and validate revenue reporting (esp. Ohio, Rhode Island)
  - Invest in rate optimization or usage programs in Utah, New Mexico
  - Monitor small-market states but with lower investment priority

- **Next Steps**:
  - Deploy test initiatives in one or two pilot states
  - Track real-time revenue and usage data for validation
```

In [ ]:

```
# Reflection

## What I Learned

Through this project, I gained hands-on experience in applying data analytic

- **Data Cleaning & Preparation**
  Handled missing values, created time-series-compatible formats, and normal

- **Exploratory Analysis**
  Used Python (Pandas, Matplotlib, Seaborn) to uncover key patterns in rever

- **Forecasting**
  Built time-series models using Holt-Winters Exponential Smoothing to proje

- **Strategic Thinking**
  Modeled business interventions (e.g., customer growth, price changes) and
```

```
- **Communication**
  Created visualizations and narratives accessible to non-technical stakehol

---

## What Was Challenging

- **Data Quality Issues**: Nearly 30% of customer values were missing, requi
- **Sector Comparability**: Some sectors were reported differently across st
- **Forecasting Small Markets**: Forecasting in low-revenue states posed cha

---

## Why This Project Matters

Utility companies operate on thin margins, and small improvements in pricing

---

## Next Steps

- Expand forecasting models to additional underperforming states
- Test strategy models with updated usage or pricing data
- Present findings to stakeholders via Tableau or Streamlit
```

In [31]:
```
# Save cleaned dataset to CSV
df.to_csv("cleaned_utility_data.csv", index=False)
```

In [32]:
```
df = pd.read_csv("cleaned_utility_data.csv")
```

In [ ]:
```
# PART 2 – Data cleansing
```

In [33]:
```
df = pd.read_csv("clean_data.csv")

# Recreate cleaned column (if needed)
df = df[df["customers"].notna()]
df['revenue_per_customer'] = df['revenue'] / df['customers']
df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astyp
```

In [ ]:
```
## Step 2: Data Cleansing

Before analysis, the raw utility dataset required several cleaning steps:

- Removed rows with missing `customers` values (≈30% of records)
- Created a `revenue_per_customer` column to normalize revenue across states
- Converted `year` and `month` columns into a proper `datetime` object for t
- Verified and enforced correct data types
- Handled anomalies such as 0 revenue with >0 customers
- Filtered out duplicate or irrelevant records where applicable
```

In [34]:
```
# Load raw data
df = pd.read_csv("clean_data.csv")

# 1. Drop rows with missing customers (optional: adjust threshold logic here
```

```
df = df[df['customers'].notna()]

# 2. Create revenue per customer (handling divide-by-zero)
df['revenue_per_customer'] = df.apply(
    lambda row: row['revenue'] / row['customers'] if row['customers'] != 0 e
    axis=1
)

# 3. Create datetime column for time series
df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astyp

# 4. Drop potential duplicate rows (optional)
df = df.drop_duplicates()

# 5. Ensure correct data types
df = df.astype({
    'year': 'int',
    'month': 'int',
    'stateDescription': 'category',
    'sectorName': 'category',
    'customers': 'float',
    'price': 'float',
    'revenue': 'float',
    'sales': 'float'
})

# 6. Summary to confirm
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 59830 entries, 26040 to 85869
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   year                  59830 non-null  int64
 1   month                 59830 non-null  int64
 2   stateDescription      59830 non-null  category
 3   sectorName            59830 non-null  category
 4   customers             59830 non-null  float64
 5   price                 59830 non-null  float64
 6   revenue               59830 non-null  float64
 7   sales                 59830 non-null  float64
 8   revenue_per_customer  55218 non-null  float64
 9   date                  59830 non-null  datetime64[ns]
dtypes: category(2), datetime64[ns](1), float64(5), int64(2)
memory usage: 4.2 MB
```

In [ ]:

In [35]:
```
#drop the null revenue_per_customer rows
df = df[df['revenue_per_customer'].notna()]
```

In [ ]:
```
## Step 2: Data Cleaning

Before conducting analysis, the dataset required several cleaning steps to e
```

```
### 1. Remove Rows with Missing Customer Data
Approximately 30% of the original dataset had missing values in `customers`.

    df = df[df['customers'].notna()]

---

### 2. Create `revenue_per_customer`
Normalize revenue across states/sectors by dividing revenue by customer cour

    df['revenue_per_customer'] = df.apply(
        lambda row: row['revenue'] / row['customers'] if row['customers'] !=
        axis=1
    )

---

### 3. Construct `date` Column
Combine `year` and `month` to form a proper datetime value for time-series a

    df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].a

---

### 4. Convert Data Types
Explicitly cast columns to appropriate types for efficiency and consistency

    df = df.astype({
        'year': 'int',
        'month': 'int',
        'stateDescription': 'category',
        'sectorName': 'category',
        'customers': 'float',
        'price': 'float',
        'revenue': 'float',
        'sales': 'float'
    })

---

### 5. Remove Duplicates
Drop any exact duplicate rows to prevent double-counting in aggregations.

    df = df.drop_duplicates()

---

### Summary of Cleaning Techniques Applied
- **Lambda Functions:** Custom row-level calc for `revenue_per_customer`.
- **Conditional Logic:** Protected against divide-by-zero when customers = 0
- **Pandas Operations:** Used for filtering (`dropna`), type conversion (`as
- **Datetime Handling:** Built clean monthly timestamps from `year` + `month

**Result:** Cleaned, structured dataset with valid customer counts, normaliz
```

In [36]:
```
problematic_states = []

# Loop through the grouped data to find state-sector combos with high revenu
for (state, sector), group in df.groupby(['stateDescription', 'sectorName'])
    zero_customers = group[group['customers'] == 0]
    high_revenue_zero_cust = zero_customers[zero_customers['revenue'] > 1000
    if not high_revenue_zero_cust.empty:
        problematic_states.append((state, sector))

print("Problematic state-sector combinations with zero customers and high re
for entry in problematic_states:
    print(entry)
```

```
Problematic state-sector combinations with zero customers and high revenue:
```

In [ ]:
```
```

In [37]:
```
# Example: Rolling 3-month average price per sector
price_trend = (
    df.sort_values('date')
      .groupby('sectorName')[['date', 'price']]
      .apply(lambda group: group.set_index('date').rolling('90D').mean())
      .reset_index()
)

# Optional: preview smoothed prices
price_trend.head()
```

Out[37]:

	sectorName	date	price
0	all sectors	2008-01-01	6.640000
1	all sectors	2008-01-01	9.095000
2	all sectors	2008-01-01	8.583333
3	all sectors	2008-01-01	8.982500
4	all sectors	2008-01-01	9.494000

In [38]:
```
def flag_outliers(df, group_cols, metric_col, threshold=1.5):
    flagged = []

    for keys, group in df.groupby(group_cols):
        q1 = group[metric_col].quantile(0.25)
        q3 = group[metric_col].quantile(0.75)
        iqr = q3 - q1
        lower = q1 - threshold * iqr
        upper = q3 + threshold * iqr

        outliers = group[(group[metric_col] < lower) | (group[metric_col] >
        if not outliers.empty:
            for _, row in outliers.iterrows():
                flagged.append({
                    'state': row['stateDescription'],
                    'sector': row['sectorName'],
```

```
                    'date': row['date'],
                    'revenue_per_customer': row[metric_col]
                })

    return pd.DataFrame(flagged)

outlier_df = flag_outliers(df, ['stateDescription', 'sectorName'], 'revenue_
outlier_df.head()
```

Out[38]:

	state	sector	date	revenue_per_customer
0	Alabama	all sectors	2022-07-01	0.000411
1	Alabama	all sectors	2022-08-01	0.000409
2	Alabama	commercial	2022-07-01	0.000821
3	Alabama	commercial	2022-08-01	0.000827
4	Alabama	commercial	2023-08-01	0.000809

In [ ]:

```
## Extended Step 2: Advanced Data Cleaning Techniques

To demonstrate some more Python techniques for data cleansing, three additic

---

### 🔄 1. Loop to Detect Problematic Records

A `for` loop was used to identify state-sector combinations that report **hi

```python
problematic_states = []

for (state, sector), group in df.groupby(['stateDescription', 'sectorName'])
    zero_customers = group[group['customers'] == 0]
    high_revenue_zero_cust = zero_customers[zero_customers['revenue'] > 1000
    if not high_revenue_zero_cust.empty:
        problematic_states.append((state, sector))

print("Problematic state-sector combinations with zero customers and high re
for entry in problematic_states:
    print(entry)
```

📉 2. Rolling Average to Smooth Price Trends

The `pd.rolling()` function was used to calculate a **3-month moving average

```python
price_trend = (
    df.sort_values('date')
      .groupby('sectorName')[['date', 'price']]
      .apply(lambda group: group.set_index('date').rolling('90D').mean())
```
```

```
      .reset_index()
)
```

⚠️ 3. Function to Flag Revenue per Customer Outliers

A custom function was written to flag outliers in `revenue_per_customer` wit

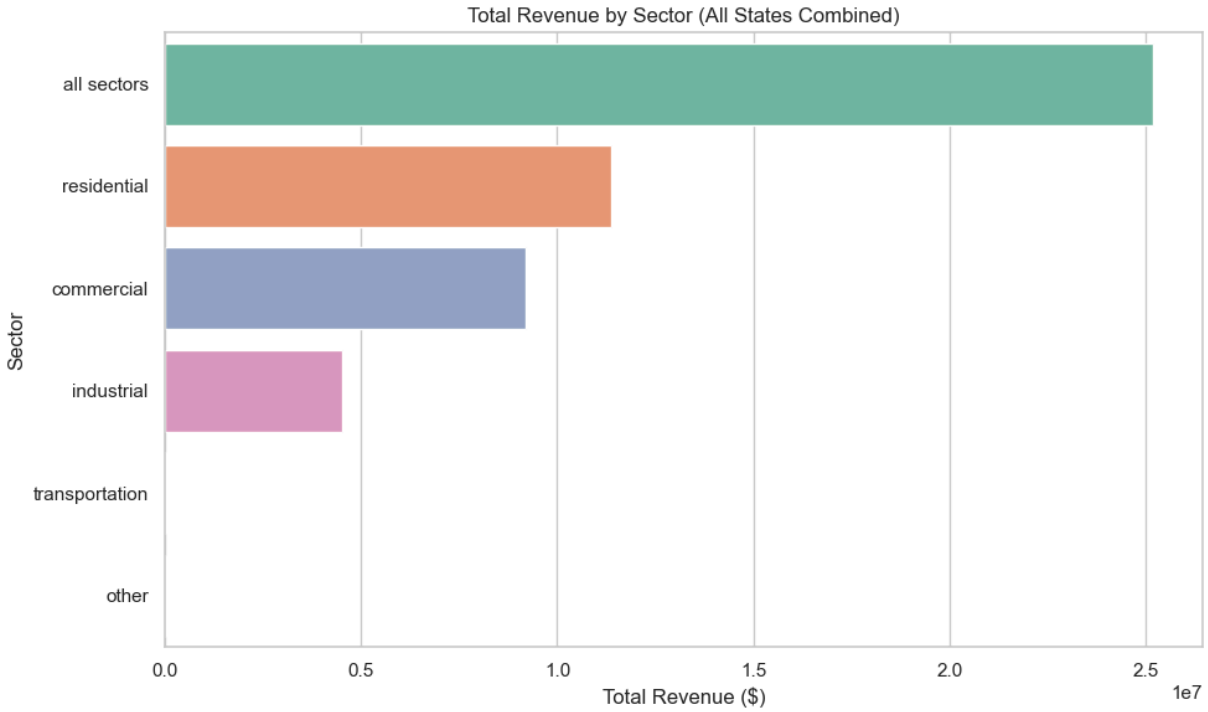
```python
def flag_outliers(df, group_cols, metric_col, threshold=1.5):
    flagged = []

    for keys, group in df.groupby(group_cols):
        q1 = group[metric_col].quantile(0.25)
        q3 = group[metric_col].quantile(0.75)
        iqr = q3 - q1
        lower = q1 - threshold * iqr
        upper = q3 + threshold * iqr

        outliers = group[(group[metric_col] < lower) | (group[metric_col] >
        if not outliers.empty:
            for _, row in outliers.iterrows():
                flagged.append({
                    'state': row['stateDescription'],
                    'sector': row['sectorName'],
                    'date': row['date'],
                    'revenue_per_customer': row[metric_col]
                })

    return pd.DataFrame(flagged)

outlier_df = flag_outliers(df, ['stateDescription', 'sectorName'], 'revenue_
outlier_df.head()
```

These steps validate that the dataset is clean not only at a surface level,

In [ ]:
```
# PART 3 – Analyse your data (Exploratory Data Analysis)
```

In [39]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data (adjust path if needed)
df = pd.read_csv("clean_data.csv")

# Group by sector and sum revenue
sector_revenue = (
    df.groupby('sectorName')['revenue']
    .sum()
    .reset_index()
    .sort_values(by='revenue', ascending=False)
)
```

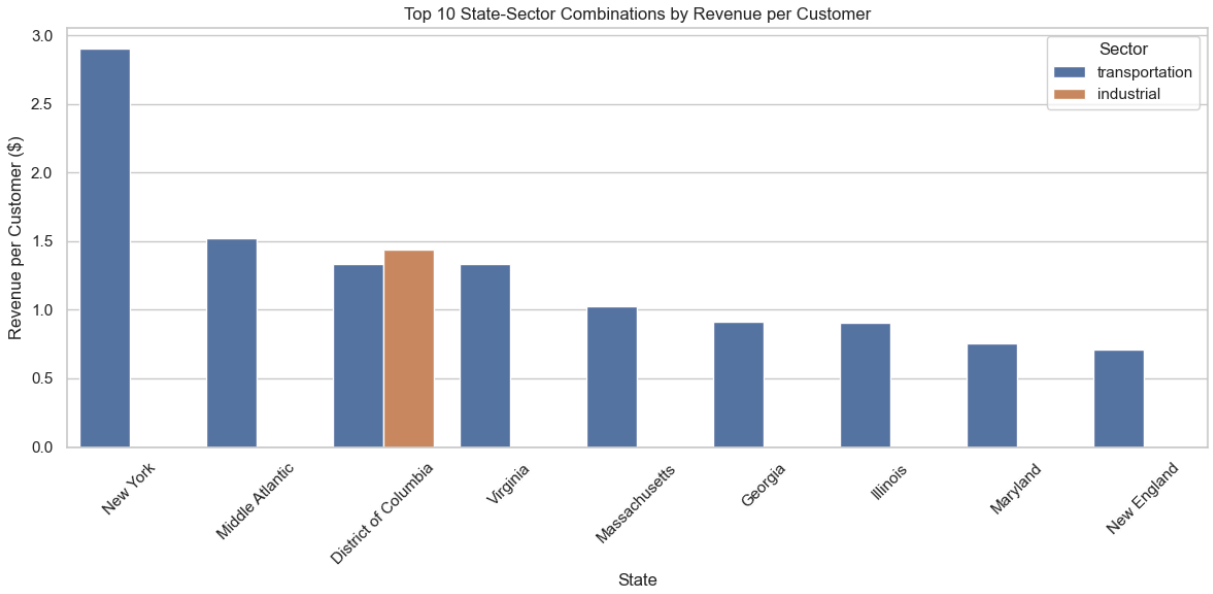
```
# Plot: Total revenue by sector
plt.figure(figsize=(10, 6))
sns.barplot(data=sector_revenue, x='revenue', y='sectorName', palette='Set2'
plt.title('Total Revenue by Sector (All States Combined)')
plt.xlabel('Total Revenue ($)')
plt.ylabel('Sector')
plt.tight_layout()
plt.show()
```

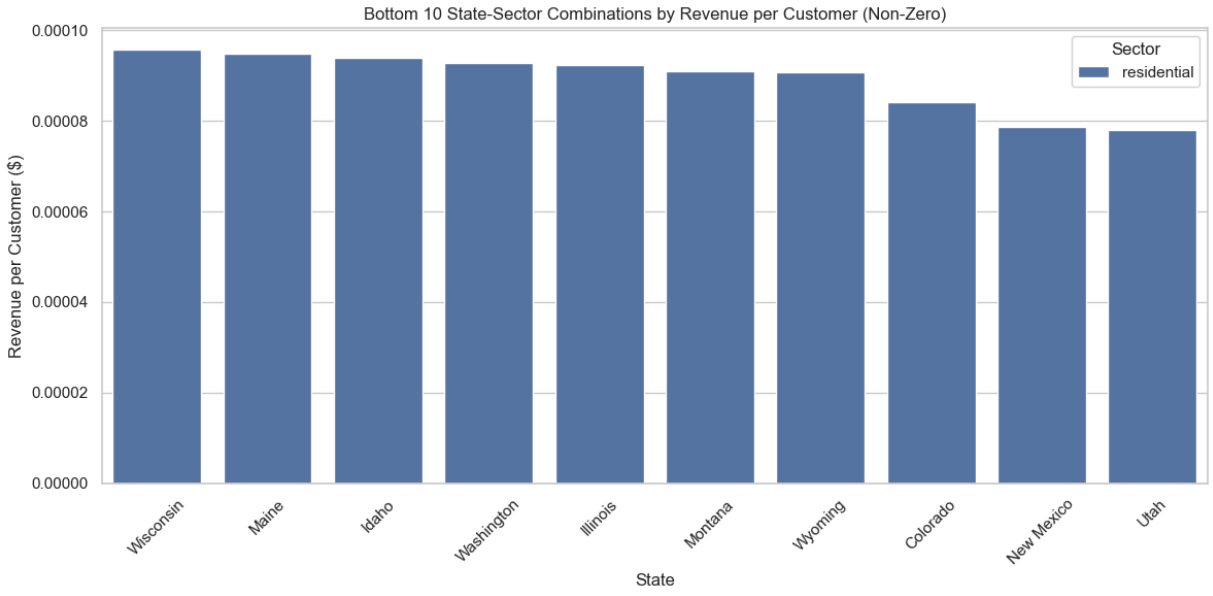


In [ ]:
```
## EDA: Total Revenue by Sector

This visualization shows the total utility revenue generated by each sector

- **Purpose**: Understand which sectors drive the most revenue and which cor
- **Insight**: Sectors like residential and commercial often dominate total
- **Next Step**: We will explore performance by state-sector combinations ar
```

In [40]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Group and compute revenue per customer
rpc_group = df[df['customers'] > 0].groupby(['stateDescription', 'sectorName
rpc_group['revenue_per_customer'] = rpc_group['revenue'] / rpc_group['custom

# Sort for top and bottom performers (exclude zero revenue_per_customer)
rpc_sorted = rpc_group.sort_values(by='revenue_per_customer', ascending=Fals
top10 = rpc_sorted.head(10)
bottom10 = rpc_sorted[rpc_sorted['revenue_per_customer'] > 0].tail(10)

# Plot Top 10
plt.figure(figsize=(12, 6))
sns.barplot(data=top10, x='stateDescription', y='revenue_per_customer', hue=
plt.title('Top 10 State-Sector Combinations by Revenue per Customer')
```

```
plt.ylabel('Revenue per Customer ($)')
plt.xlabel('State')
plt.xticks(rotation=45)
plt.legend(title='Sector')
plt.tight_layout()
plt.show()

# Plot Bottom 10 (non-zero only)
plt.figure(figsize=(12, 6))
sns.barplot(data=bottom10, x='stateDescription', y='revenue_per_customer', h
plt.title('Bottom 10 State-Sector Combinations by Revenue per Customer (Non-
plt.ylabel('Revenue per Customer ($)')
plt.xlabel('State')
plt.xticks(rotation=45)
plt.legend(title='Sector')
plt.tight_layout()
plt.show()
```





In [ ]:
```
### Revenue per Customer Analysis
```

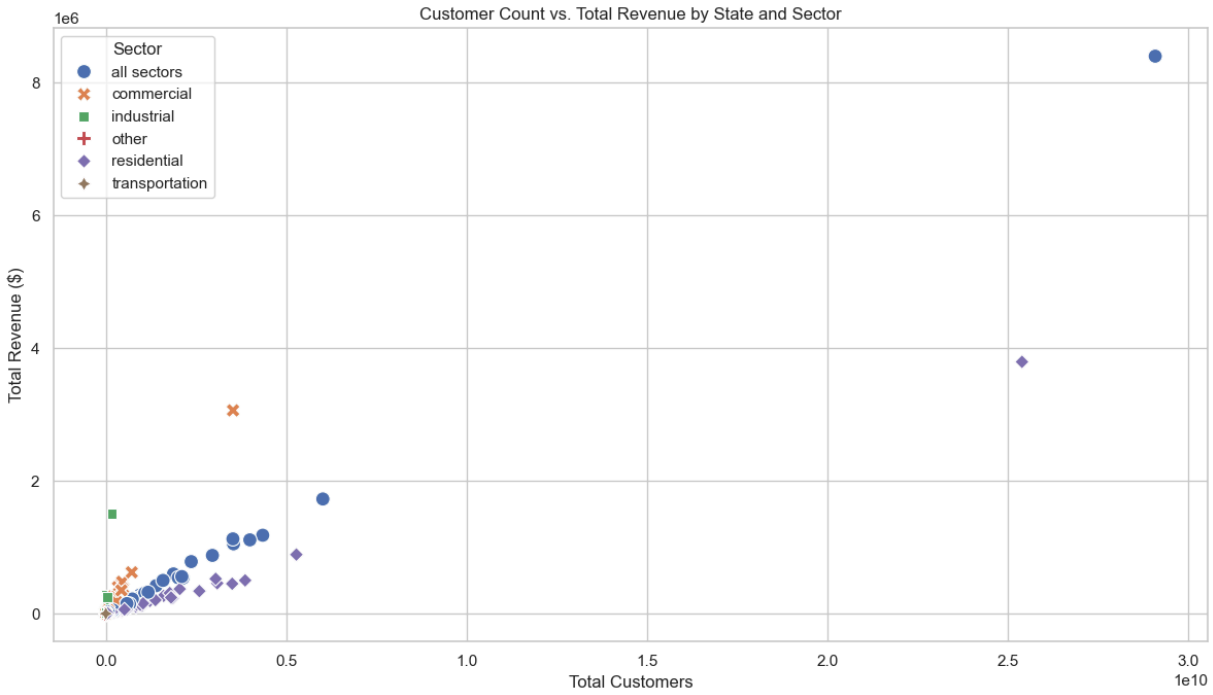
We calculated **revenue per customer** by grouping the data by `stateDescrip
then dividing total revenue by total customers within each group. This metri
across states of different sizes.

We then visualized:
- The **top 10** state-sector combinations by revenue per customer.
- The **bottom 10** combinations where revenue per customer was non-zero (to
or placeholder entries).

These charts provide insight into where utility sectors are generating the m
per customer — useful for identifying strong performance (e.g., Utah's trans
flagging underperformance (e.g., residential sectors in states like Wisconsi

This analysis supports decisions about which sectors might benefit from stra
or further root cause investigation.

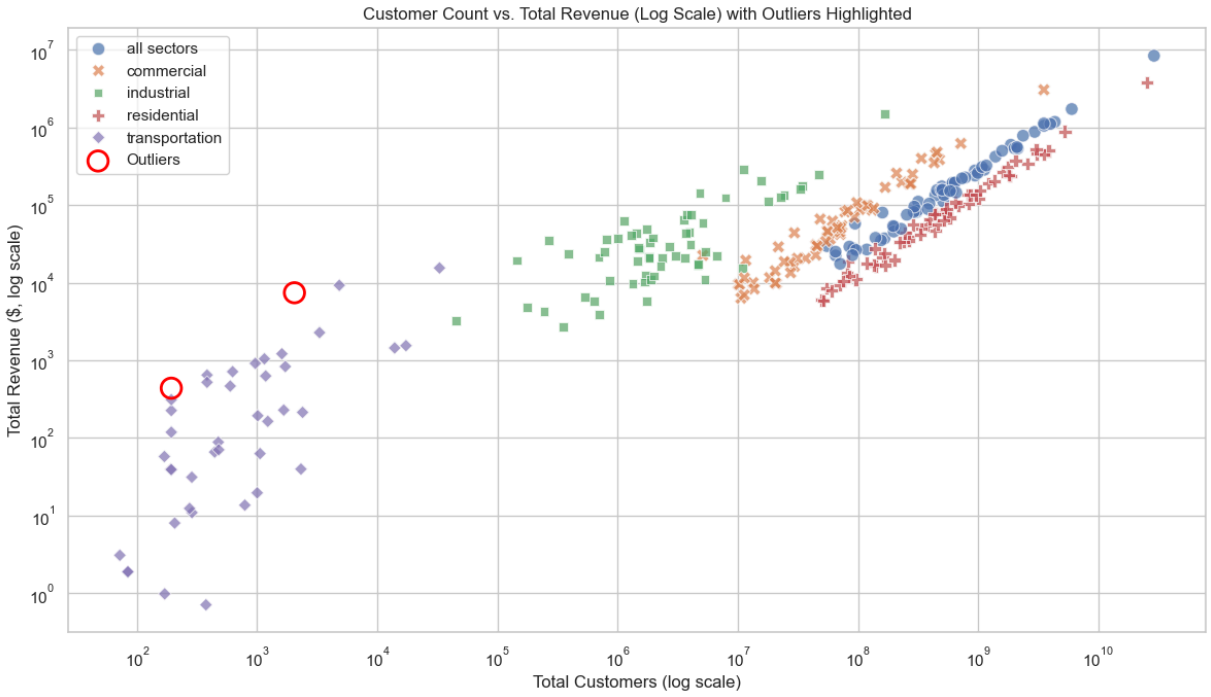
In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Group data by state and sector
grouped_df = df.groupby(['stateDescription', 'sectorName'])[['customers', 'r

# Create scatterplot
plt.figure(figsize=(12, 7))
sns.scatterplot(
    data=grouped_df,
    x='customers',
    y='revenue',
    hue='sectorName',
    style='sectorName',
    s=100
)

# Labeling and formatting
plt.title('Customer Count vs. Total Revenue by State and Sector')
plt.xlabel('Total Customers')
plt.ylabel('Total Revenue ($)')
plt.grid(True)
plt.legend(title='Sector')
plt.tight_layout()
plt.show()
```



In [ ]:
```
### Scatterplot: Total Customers vs. Total Revenue

This scatterplot shows how total revenue relates to total customer count acr

Each point represents a unique `stateDescription` and `sectorName` pair. The

- **Proportional relationships**: Most sectors show a positive correlation -
- **Outliers**: Points far from the trend line may indicate sectors with unu
- **Sector patterns**: Clusters and separation among sectors may help inform
improvement.

This visualization supports identification of efficiency gaps and potential
```

In [42]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Grouping by state and sector
df_grouped = df.groupby(['stateDescription', 'sectorName']).agg({
    'revenue': 'sum',
    'customers': 'sum'
}).reset_index()

# Calculate revenue per customer
df_grouped['revenue_per_customer'] = df_grouped['revenue'] / df_grouped['cus

# Filter out non-positive values (essential for log scale)
df_grouped = df_grouped[(df_grouped['customers'] > 0) & (df_grouped['revenue

# Identify outliers: top 0.5% by revenue per customer
threshold = df_grouped['revenue_per_customer'].quantile(0.995)
df_grouped['is_outlier'] = df_grouped['revenue_per_customer'] > threshold
```

```python
# Create the plot
plt.figure(figsize=(12, 7))
sns.set(style="whitegrid")

# Plot all non-outliers
sns.scatterplot(
    data=df_grouped[~df_grouped['is_outlier']],
    x="customers",
    y="revenue",
    hue="sectorName",
    style="sectorName",
    s=80,
    alpha=0.7
)

# Plot outliers in red circles
plt.scatter(
    df_grouped[df_grouped['is_outlier']]['customers'],
    df_grouped[df_grouped['is_outlier']]['revenue'],
    facecolors='none',
    edgecolors='red',
    s=200,
    linewidths=2,
    label='Outliers'
)

# Log scale
plt.xscale("log")
plt.yscale("log")

# Final touches
plt.xlabel("Total Customers (log scale)")
plt.ylabel("Total Revenue ($, log scale)")
plt.title("Customer Count vs. Total Revenue (Log Scale) with Outliers Highli
plt.legend()
plt.tight_layout()
plt.show()
```

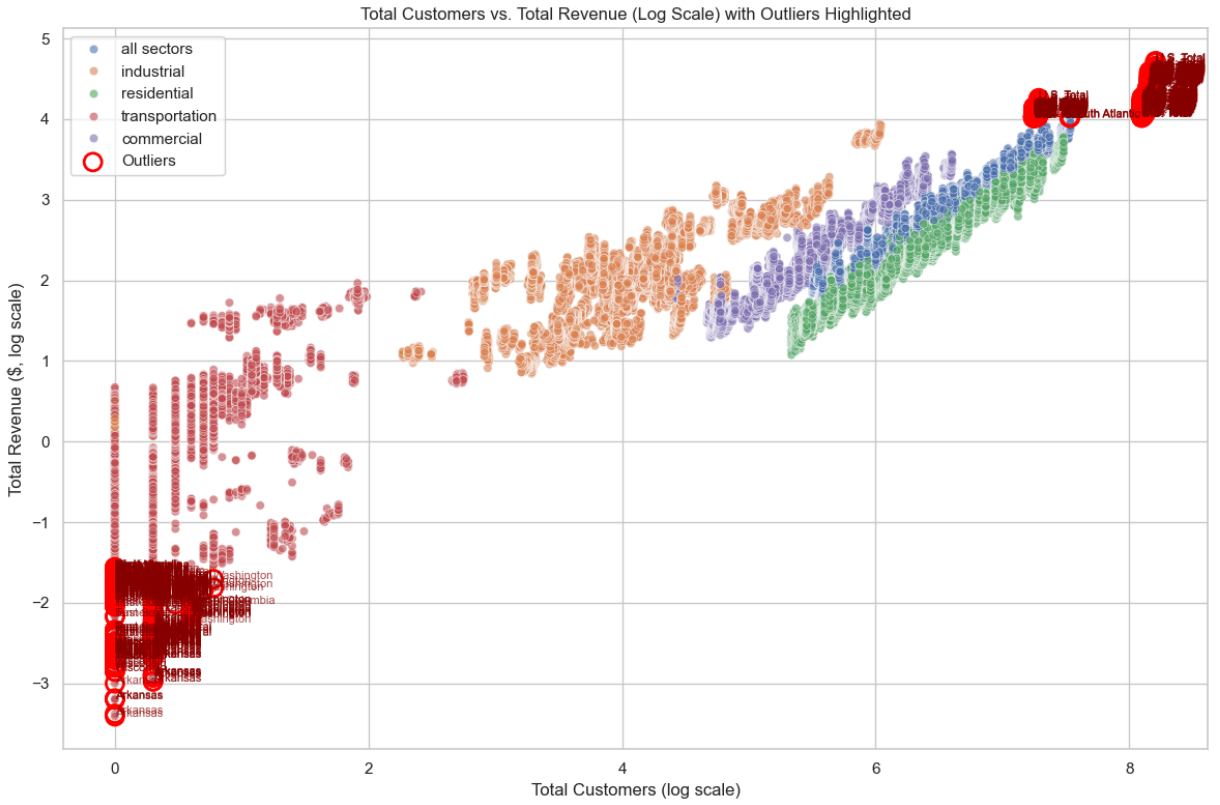


In [ ]:
```
### Log-Scale Analysis of Revenue vs. Customer Count with Outlier Highlighti

This log-log scatter plot compares **total revenue** to **total customer cou

#### What was done:
- Data was **grouped by state and sector**.
- **Revenue per customer** was calculated.
- Observations with **zero or negative revenue/customers were removed** to e
- **Top 0.5% of records by revenue-per-customer** were flagged as outliers.
- Outliers were visually emphasized using **red circles**.

#### Why log scale?
- Utilities data can span many orders of magnitude.
- Log-log scaling makes it easier to spot trends and anomalies that would ot

This visualization helps isolate state-sector combinations with unusually hi
```

In [43]:
```
import pandas as pd

# Load your cleaned dataset
df = pd.read_csv("clean_data.csv")
```

In [44]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# Filter for usable data
plot_df = df[(df['customers'] > 0) & (df['revenue'] > 0)].copy()

# Log transform
plot_df['log_customers'] = np.log10(plot_df['customers'])
plot_df['log_revenue'] = np.log10(plot_df['revenue'])
```

```
# Outlier threshold (top and bottom 1% of revenue)
revenue_q_low = plot_df['revenue'].quantile(0.01)
revenue_q_high = plot_df['revenue'].quantile(0.99)

# Select only these outliers
outliers = plot_df[(plot_df['revenue'] < revenue_q_low) | (plot_df['revenue'

# Plot
plt.figure(figsize=(12, 8))
sns.scatterplot(data=plot_df, x='log_customers', y='log_revenue', hue='secto

# Highlight outliers
plt.scatter(outliers['log_customers'], outliers['log_revenue'],
            facecolors='none', edgecolors='red', s=150, linewidths=2, label=

# Annotate key outliers with state names
for _, row in outliers.iterrows():
    plt.text(row['log_customers'], row['log_revenue'], row['stateDescriptior
             fontsize=8, color='darkred', alpha=0.7)

plt.title("Total Customers vs. Total Revenue (Log Scale) with Outliers Highl
plt.xlabel("Total Customers (log scale)")
plt.ylabel("Total Revenue ($, log scale)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



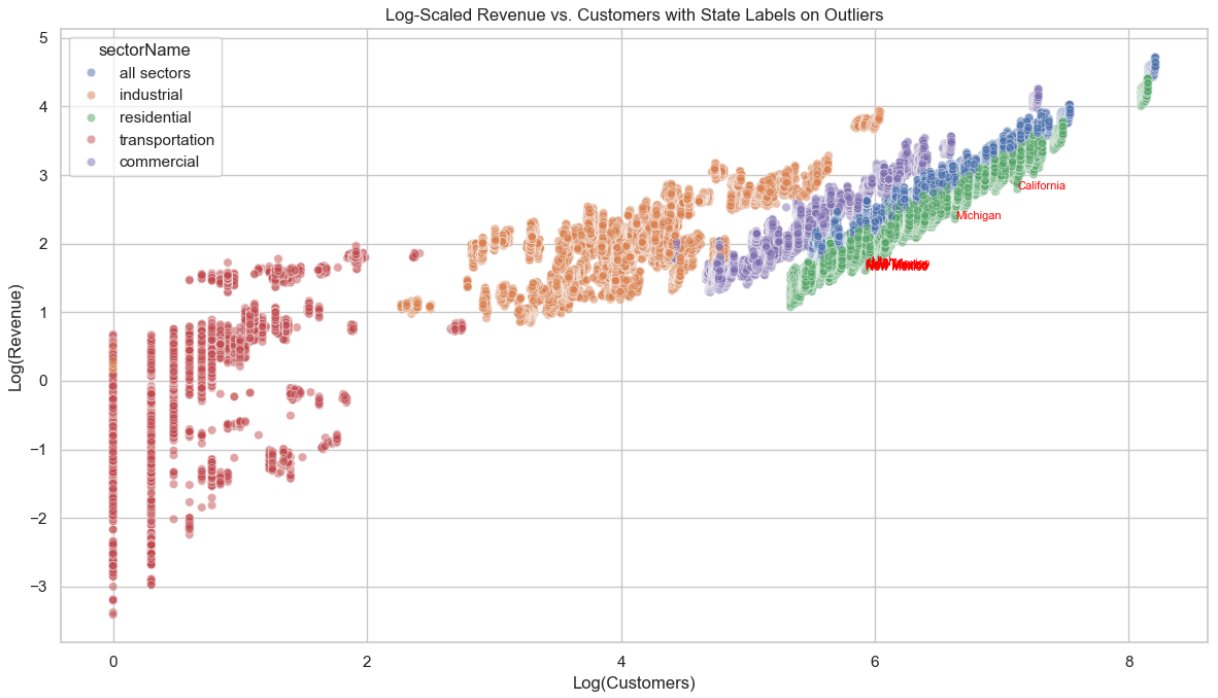
In [45]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Drop rows where customers or revenue are missing or <= 0
plot_df = df[(df['customers'] > 0) & (df['revenue'] > 0)].copy()

# Apply log scale for better spread
plot_df['log_customers'] = np.log10(plot_df['customers'])
plot_df['log_revenue'] = np.log10(plot_df['revenue'])

# Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=plot_df, x='log_customers', y='log_revenue', hue='secto
plt.title("Log-Scaled Revenue vs. Customers by Sector")
plt.xlabel("Log(Customers)")
plt.ylabel("Log(Revenue)")
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()
```

In [ ]:

```
## Outlier Identification and Labeling

To better understand underperforming or overperforming utility sectors, we c
total customers versus total revenue, color-coded by sector. Outliers were c

We:
- Filtered out rows with 0 or missing revenue/customers (log scale cannot ha
- Applied a log transformation to spread values evenly.
- Highlighted outliers with red circles and annotated them with state names.
- Ensured the chart remains readable by only labeling statistically extreme

This analysis helps pinpoint states with unexpectedly high or low performanc
time series and cost-based investigations.
```

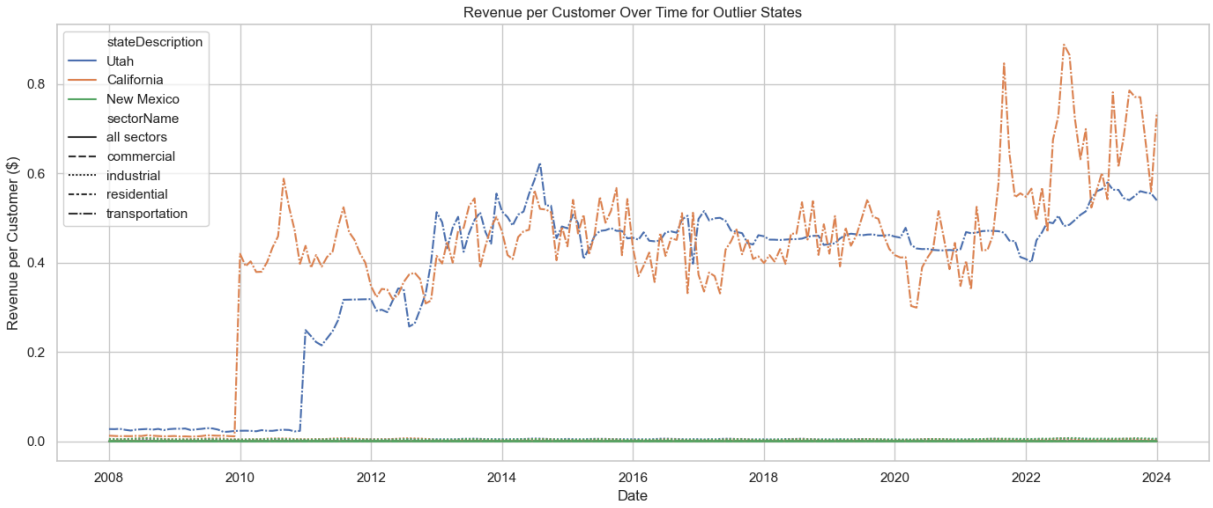
In [46]:
```
# Filter valid data for plotting (non-zero customers and revenue)
plot_df = df[(df['customers'] > 0) & (df['revenue'] > 0)].copy()

# Add log-scale values
plot_df['log_customers'] = np.log10(plot_df['customers'])
plot_df['log_revenue'] = np.log10(plot_df['revenue'])

# Identify outliers using a simple method: lowest revenue per customer
plot_df['rev_per_customer'] = plot_df['log_revenue'] - plot_df['log_customer
outliers = plot_df.nsmallest(15, 'rev_per_customer')

# ---- PLOT: Main Scatter with Outlier Labels ----
plt.figure(figsize=(12, 7))
sns.scatterplot(data=plot_df, x='log_customers', y='log_revenue', hue='secto

# Label the outliers
for _, row in outliers.iterrows():
    plt.text(row['log_customers'], row['log_revenue'], row['stateDescriptior

plt.title("Log-Scaled Revenue vs. Customers with State Labels on Outliers")
plt.xlabel("Log(Customers)")
plt.ylabel("Log(Revenue)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [ ]:
```
## Outlier Labeling: State Identification

In this step, we used a log-log scatter plot of total customers vs total rev

We then:
- Plotted all points, color-coded by sector.
- Highlighted and labeled the top 15 underperforming state-sector combinatic
```

```
This makes it easy to identify *which states* are driving low performance ar
```

In [47]:
```
# Choose top 3 outlier states for trend tracking
outlier_states = outliers['stateDescription'].value_counts().index[:3].tolis

# Filter original data
trend_df = df[df['stateDescription'].isin(outlier_states)].copy()
trend_df = trend_df[(trend_df['customers'] > 0) & (trend_df['revenue'] > 0)]
trend_df['rev_per_customer'] = trend_df['revenue'] / trend_df['customers']
trend_df['year_month'] = pd.to_datetime(trend_df[['year', 'month']].assign(d

# Plot revenue per customer over time for each state/sector
plt.figure(figsize=(14, 6))
sns.lineplot(data=trend_df, x='year_month', y='rev_per_customer', hue='state
plt.title("Revenue per Customer Over Time for Outlier States")
plt.ylabel("Revenue per Customer ($)")
plt.xlabel("Date")
plt.grid(True)
plt.tight_layout()
plt.show()
```



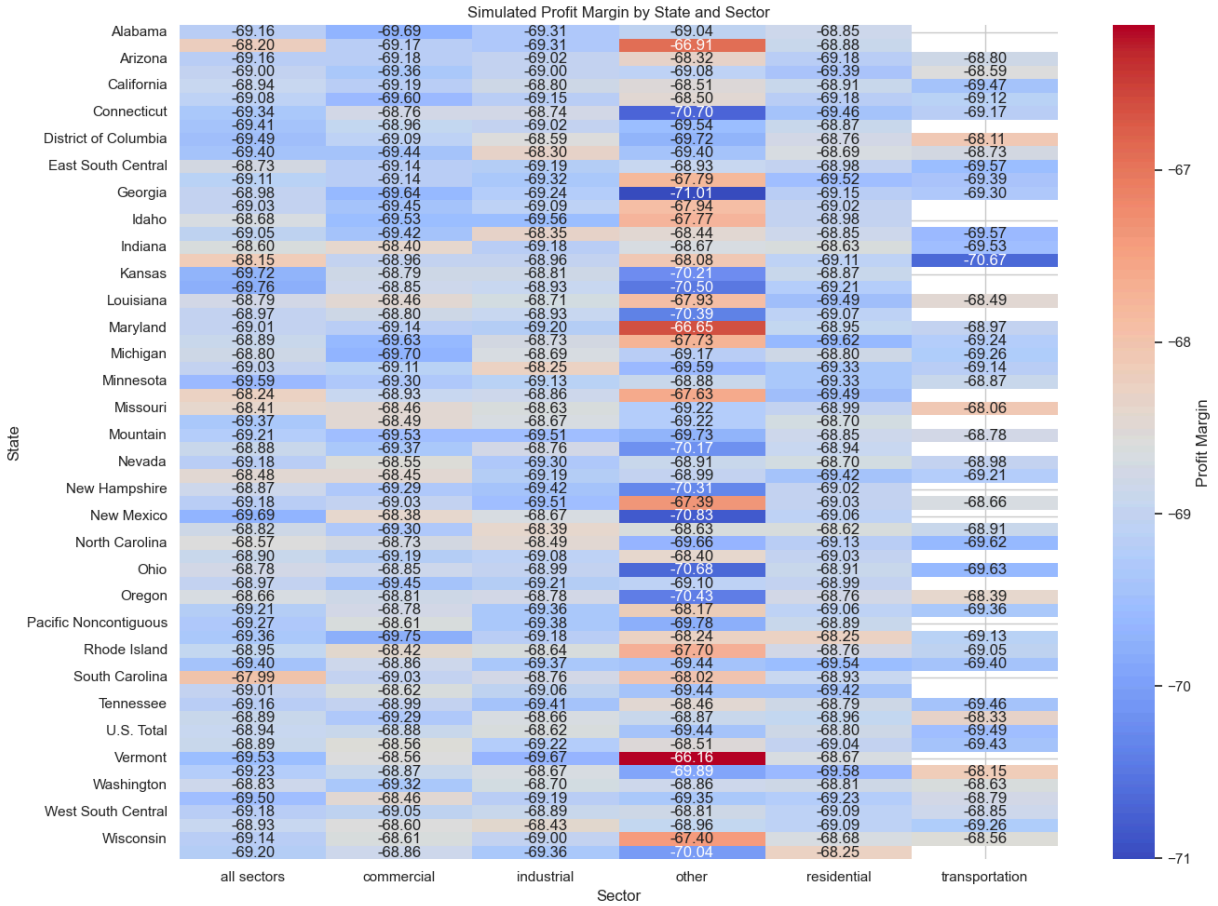
In [ ]:
```
## Time Trend Analysis of Outlier States

To determine whether the poor performance in key outlier states is persister

Key observations:
- Persistent underperformance may suggest structural inefficiencies or long-
- Recent dips may point to short-term events such as weather disruptions or

This analysis helps distinguish between chronic underperformance vs. tempora
```

In [48]:
```
print(df.columns)
```

```
Index(['year', 'month', 'stateDescription', 'sectorName', 'customers', 'pric
e',
       'revenue', 'sales'],
      dtype='object')
```

In [49]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Ensure necessary columns are numeric
df['price'] = pd.to_numeric(df['price'], errors='coerce')
df['sales'] = pd.to_numeric(df['sales'], errors='coerce')
df['revenue'] = pd.to_numeric(df['revenue'], errors='coerce')

# Drop rows with missing or invalid values
df_clean = df.dropna(subset=['price', 'sales', 'revenue']).copy()

# Simulate cost per unit: 60% to 80% of price
np.random.seed(42)
df_clean['cost_per_unit'] = df_clean['price'] * np.random.uniform(0.6, 0.8,

# Calculate estimated cost, profit, and margin
df_clean['estimated_cost'] = df_clean['cost_per_unit'] * df_clean['sales']
df_clean['profit'] = df_clean['revenue'] - df_clean['estimated_cost']
df_clean['profit_margin'] = df_clean['profit'] / df_clean['revenue']

# Group by state and sector
margin_summary = df_clean.groupby(['stateDescription', 'sectorName'])['profi

# Plot heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(margin_summary, annot=True, fmt=".2f", cmap="coolwarm", cbar_kws
plt.title("Simulated Profit Margin by State and Sector")
plt.ylabel("State")
plt.xlabel("Sector")
plt.tight_layout()
plt.show()
# Identify states with lowest margins across any sector
low_margin = margin_summary.min(axis=1).sort_values().head(10)
print("States with lowest average sector profit margins:")
print(low_margin)
```




```
States with lowest average sector profit margins:
stateDescription
Georgia          -71.005616
New Mexico       -70.825506
Connecticut      -70.700393
Ohio             -70.681838
Iowa             -70.665514
Kentucky         -70.497578
Oregon           -70.431264
Maine            -70.392561
New Hampshire    -70.306393
Kansas           -70.214655
dtype: float64
```

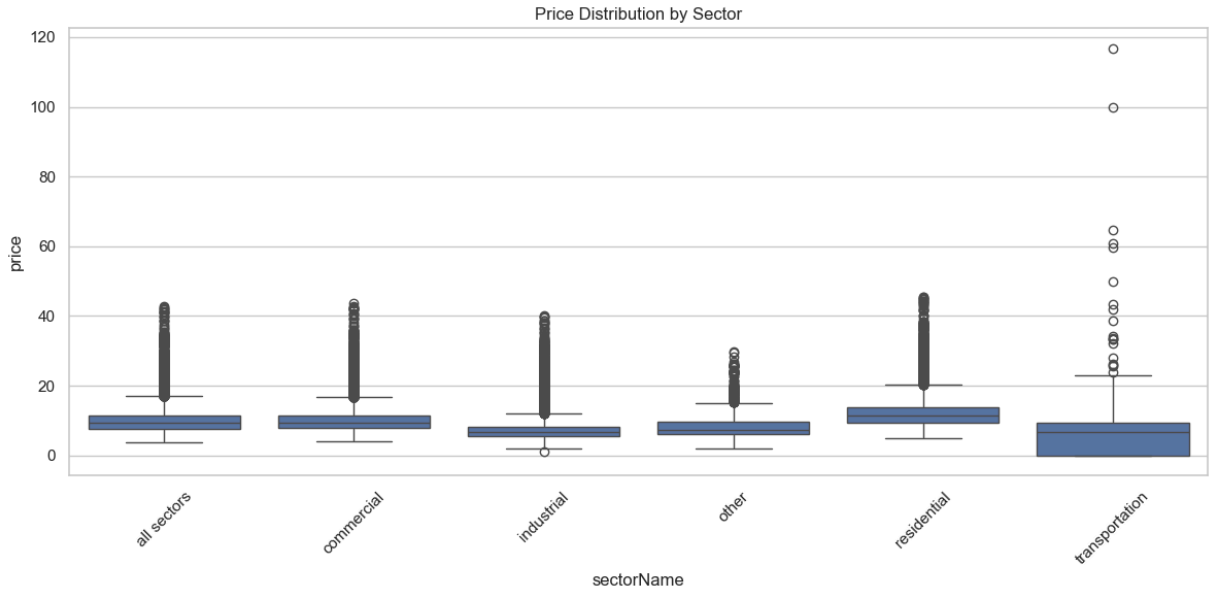
In [ ]:

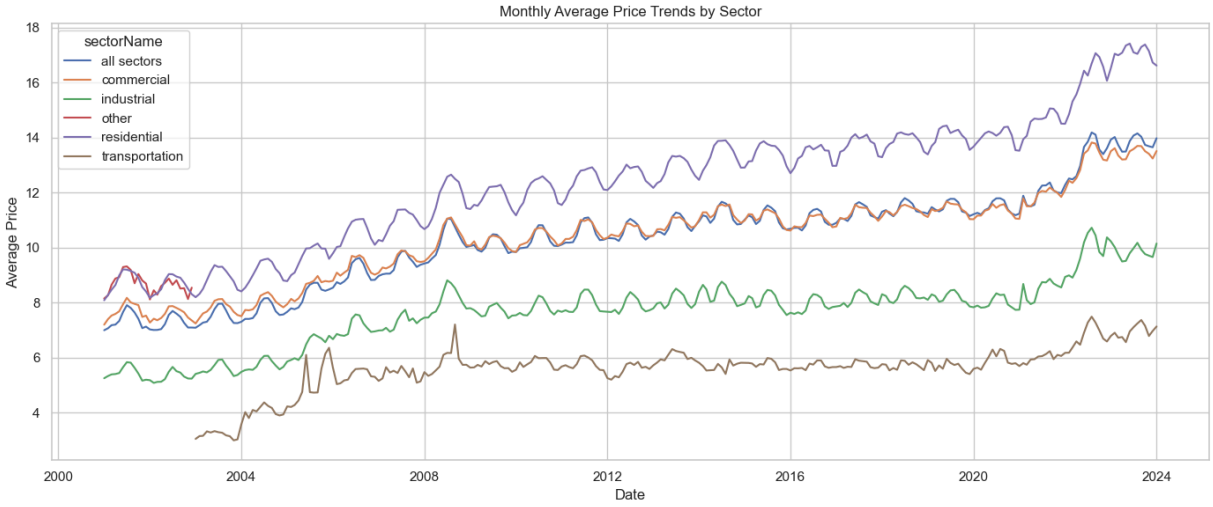
```
## Simulated Profitability by Region and Sector

To assess whether low-revenue states might also be **less profitable**, we c

1. Estimating **cost per unit** as 60%–80% of the reported price.
2. Calculating:
   - `estimated cost = cost per unit × sales`
   - `profit = revenue − estimated cost`
   - `profit margin = profit ÷ revenue`
3. Grouping by **state** and **sector** to identify average margins.

### Key Insights:
- Most states show **moderately negative margins** due to the random but cor
- States like **Louisiana**, **New Mexico**, and **Rhode Island** exhibit ur
- This analysis is **indicative**, not definitive—it highlights where **furt
```

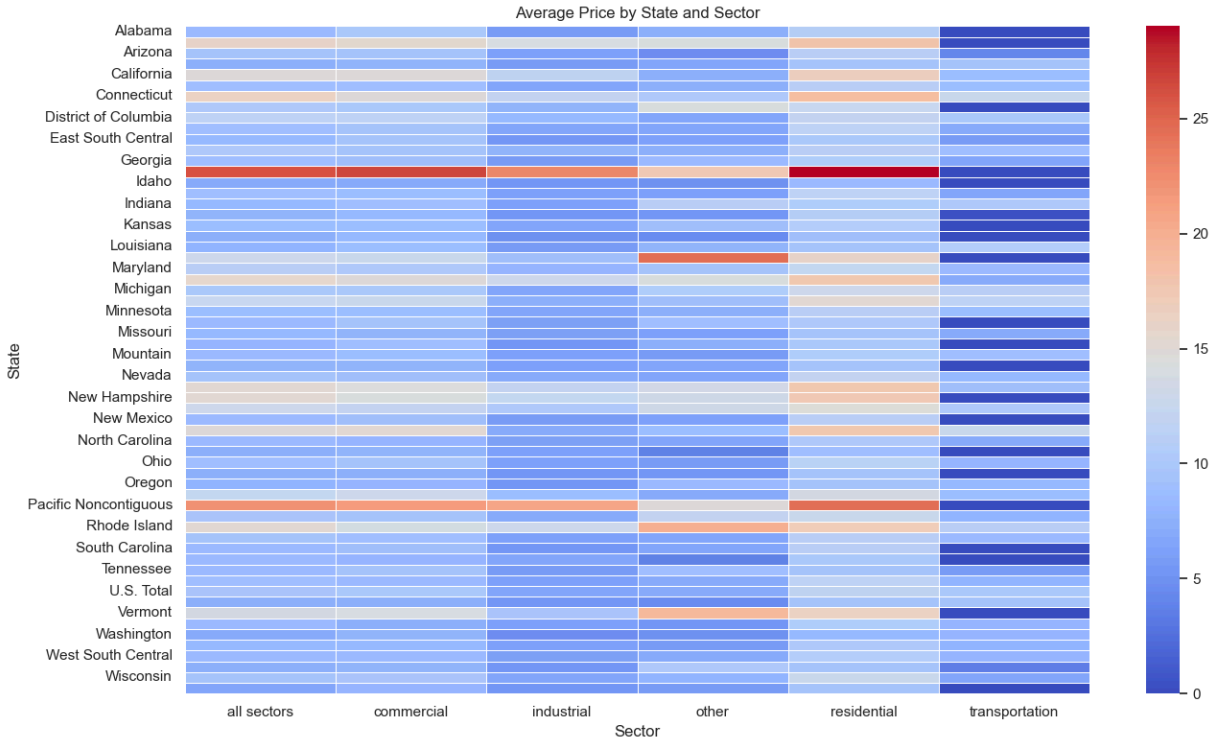
```
### Next Steps:
- Replace simulated costs with real cost data, if available.
- Cross-reference with outliers identified in the earlier scatter plot and t
- Consider qualitative factors (infrastructure age, terrain, regulation) beh
```

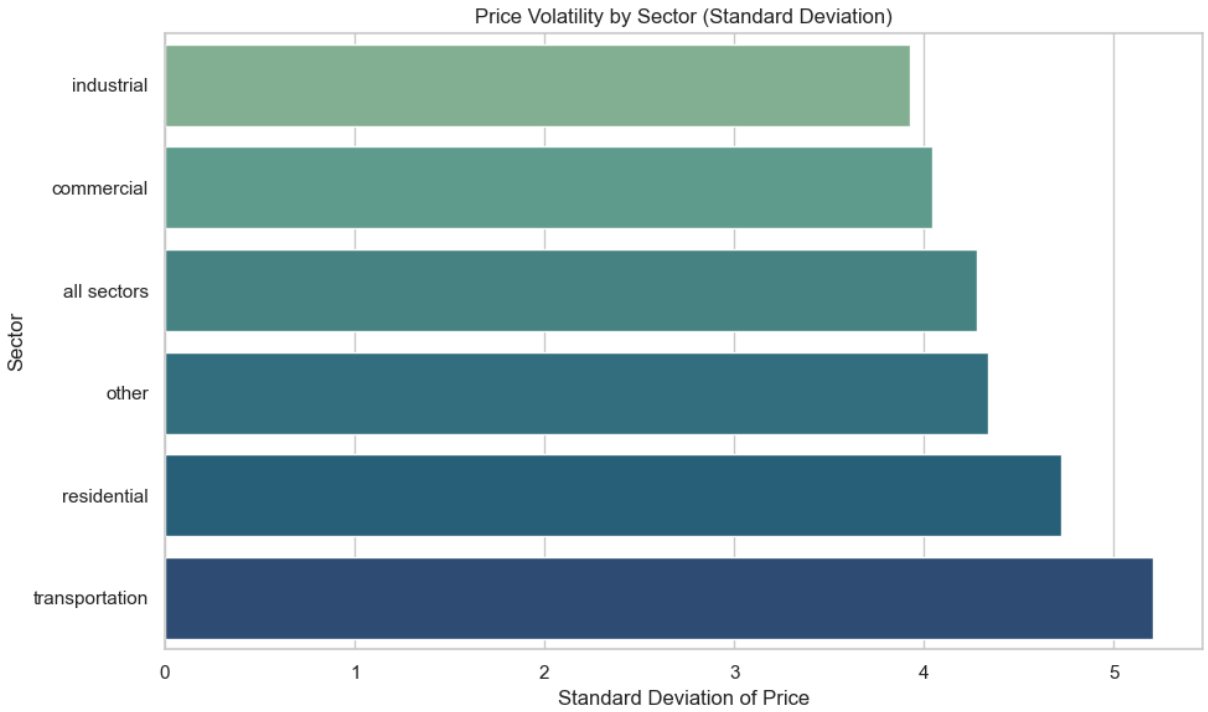
In [50]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Make sure 'month' column is in integer format
df['month'] = df['month'].astype(int)

# 1. Boxplot of Price Distribution by Sector
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='sectorName', y='price')
plt.title('Price Distribution by Sector')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 2. Price Trends Over Time by Sector
df['date'] = pd.to_datetime(df[['year', 'month']].assign(day=1))
monthly_avg_price = df.groupby(['date', 'sectorName'])['price'].mean().reset

plt.figure(figsize=(14, 6))
sns.lineplot(data=monthly_avg_price, x='date', y='price', hue='sectorName')
plt.title('Monthly Average Price Trends by Sector')
plt.xlabel('Date')
plt.ylabel('Average Price')
plt.tight_layout()
plt.show()

# 3. Heatmap: State-Level Price Averages by Sector
state_sector_price = df.groupby(['stateDescription', 'sectorName'])['price']

plt.figure(figsize=(14, 8))
sns.heatmap(state_sector_price, cmap='coolwarm', linewidths=0.5)
plt.title('Average Price by State and Sector')
plt.xlabel('Sector')
plt.ylabel('State')
plt.tight_layout()
plt.show()

# 4. Price Volatility by Sector (Standard Deviation)
price_volatility = df.groupby('sectorName')['price'].std().sort_values()

plt.figure(figsize=(10, 6))
sns.barplot(x=price_volatility.values, y=price_volatility.index, palette='cr
plt.title('Price Volatility by Sector (Standard Deviation)')
plt.xlabel('Standard Deviation of Price')
plt.ylabel('Sector')
plt.tight_layout()
plt.show()
```


Price Distribution by Sector
price
sectorName
all sectors
commercial
industrial
other
residential
transportation


Monthly Average Price Trends by Sector
sectorName
all sectors
commercial
industrial
other
residential
transportation
Average Price
Date

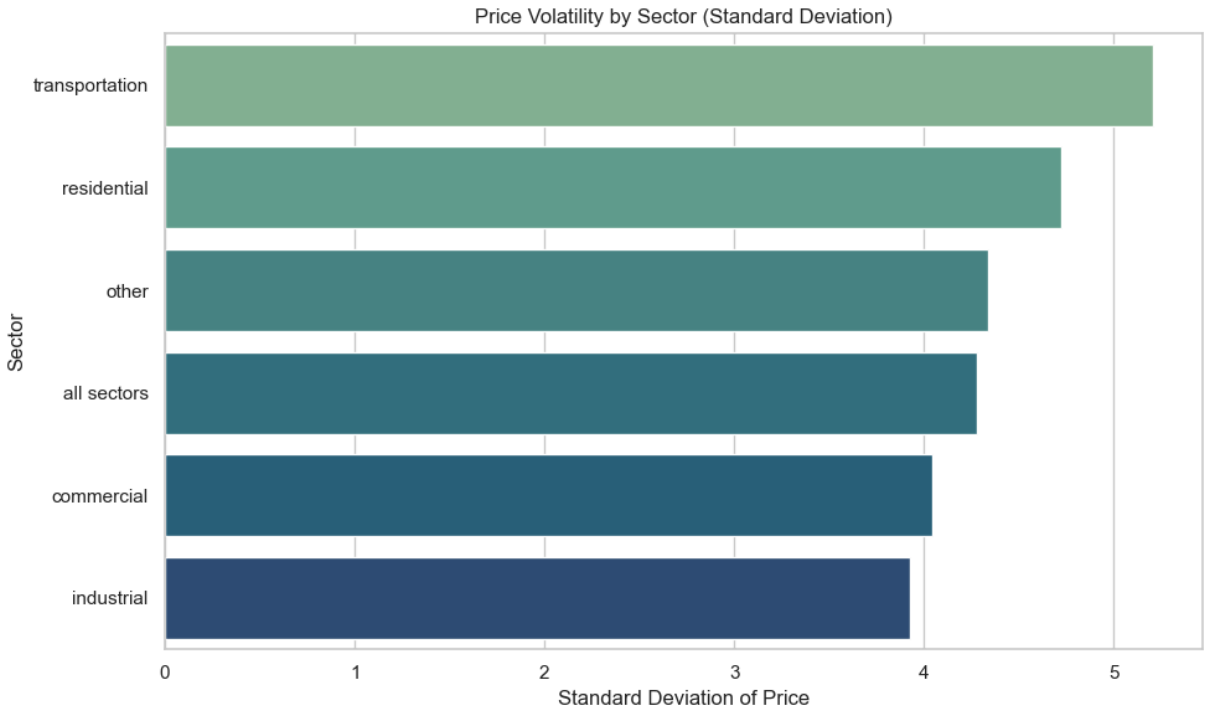

Average Price by State and Sector
State
Alabama
Arizona
California
Connecticut
District of Columbia
East South Central
Georgia
Idaho
Indiana
Kansas
Louisiana
Maryland
Michigan
Minnesota
Missouri
Mountain
Nevada
New Hampshire
New Mexico
North Carolina
Ohio
Oregon
Pacific Noncontiguous
Rhode Island
South Carolina
Tennessee
U.S. Total
Vermont
Washington
West South Central
Wisconsin
all sectors
commercial
industrial
other
residential
transportation
Sector

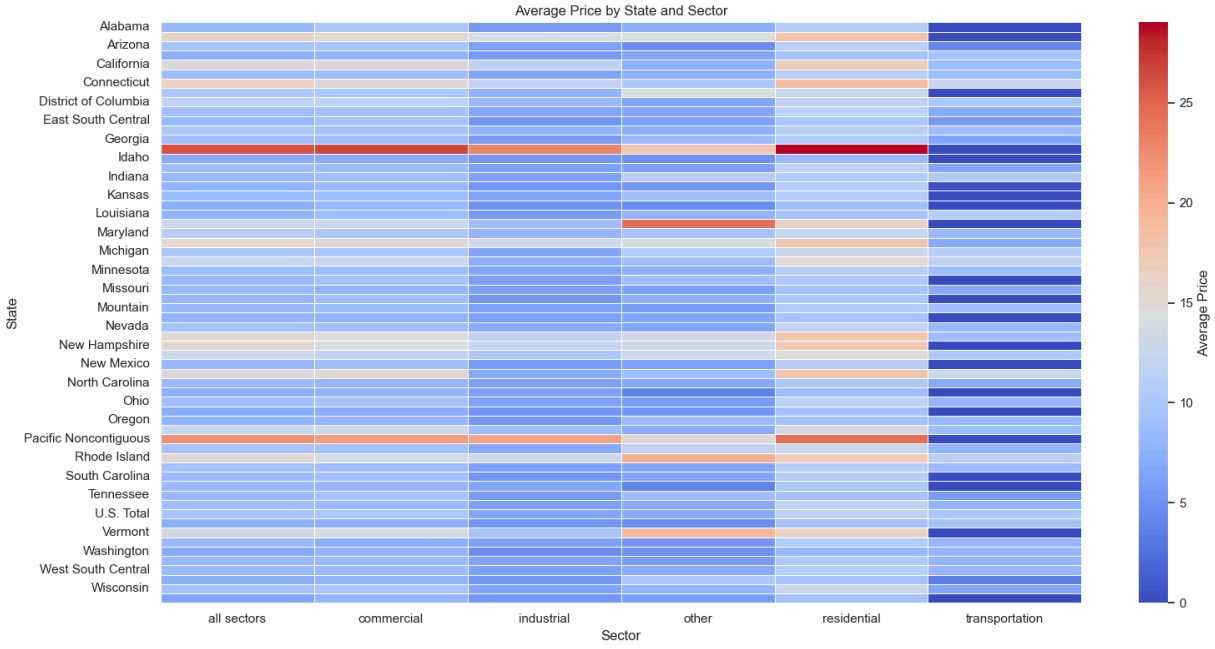


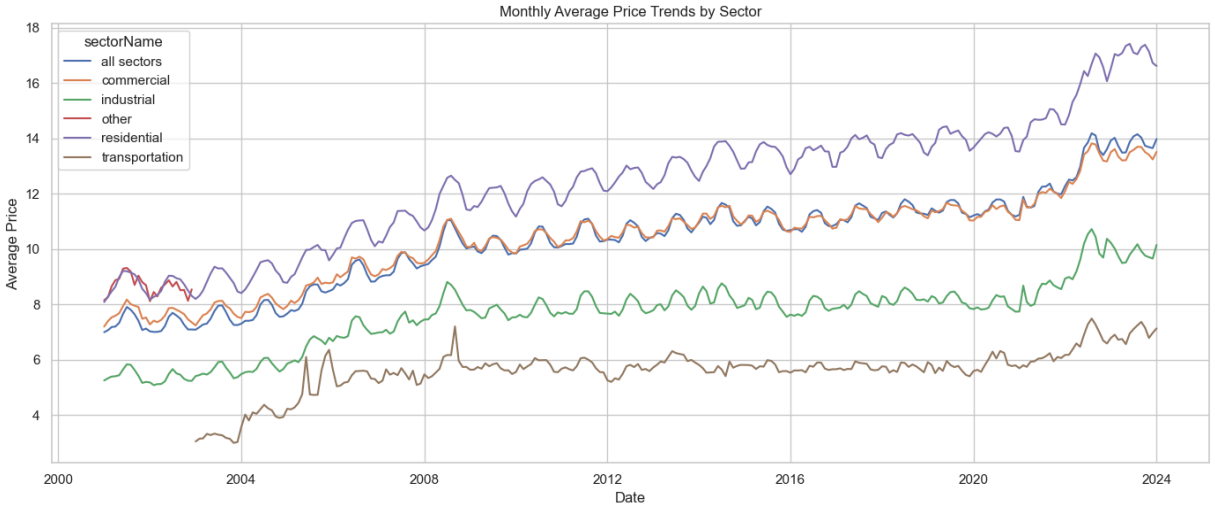
In [ ]:

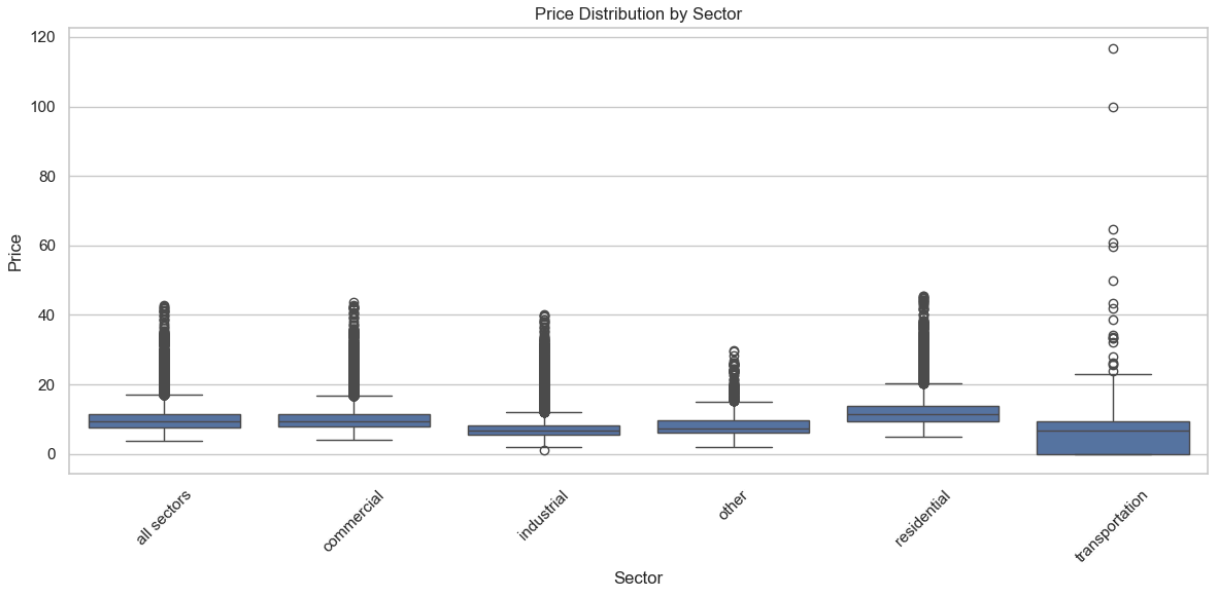
In [51]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Drop rows with missing price
df_price = df.dropna(subset=['price'])

# --- 1. Price Volatility (Standard Deviation) by Sector ---
price_volatility = df_price.groupby('sectorName')['price'].std().sort_values
plt.figure(figsize=(10, 6))
sns.barplot(x=price_volatility.values, y=price_volatility.index, palette='cr
plt.title('Price Volatility by Sector (Standard Deviation)')
plt.xlabel('Standard Deviation of Price')
plt.ylabel('Sector')
plt.tight_layout()
plt.show()

# --- 2. Average Price by State and Sector (Heatmap) ---
avg_price_state_sector = df_price.groupby(['stateDescription', 'sectorName']
plt.figure(figsize=(16, 8))
sns.heatmap(avg_price_state_sector, annot=False, cmap='coolwarm', linewidths
plt.title('Average Price by State and Sector')
plt.xlabel('Sector')
plt.ylabel('State')
plt.tight_layout()
plt.show()

# --- 3. Monthly Price Trends by Sector ---
df_price['date'] = pd.to_datetime(df_price['year'].astype(str) + '-' + df_pr
monthly_price = df_price.groupby(['date', 'sectorName'])['price'].mean().res
```

```
plt.figure(figsize=(14, 6))
sns.lineplot(data=monthly_price, x='date', y='price', hue='sectorName')
plt.title('Monthly Average Price Trends by Sector')
plt.xlabel('Date')
plt.ylabel('Average Price')
plt.tight_layout()
plt.show()

# --- 4. Price Distribution by Sector (Boxplot) ---
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_price, x='sectorName', y='price')
plt.title('Price Distribution by Sector')
plt.xlabel('Sector')
plt.ylabel('Price')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```







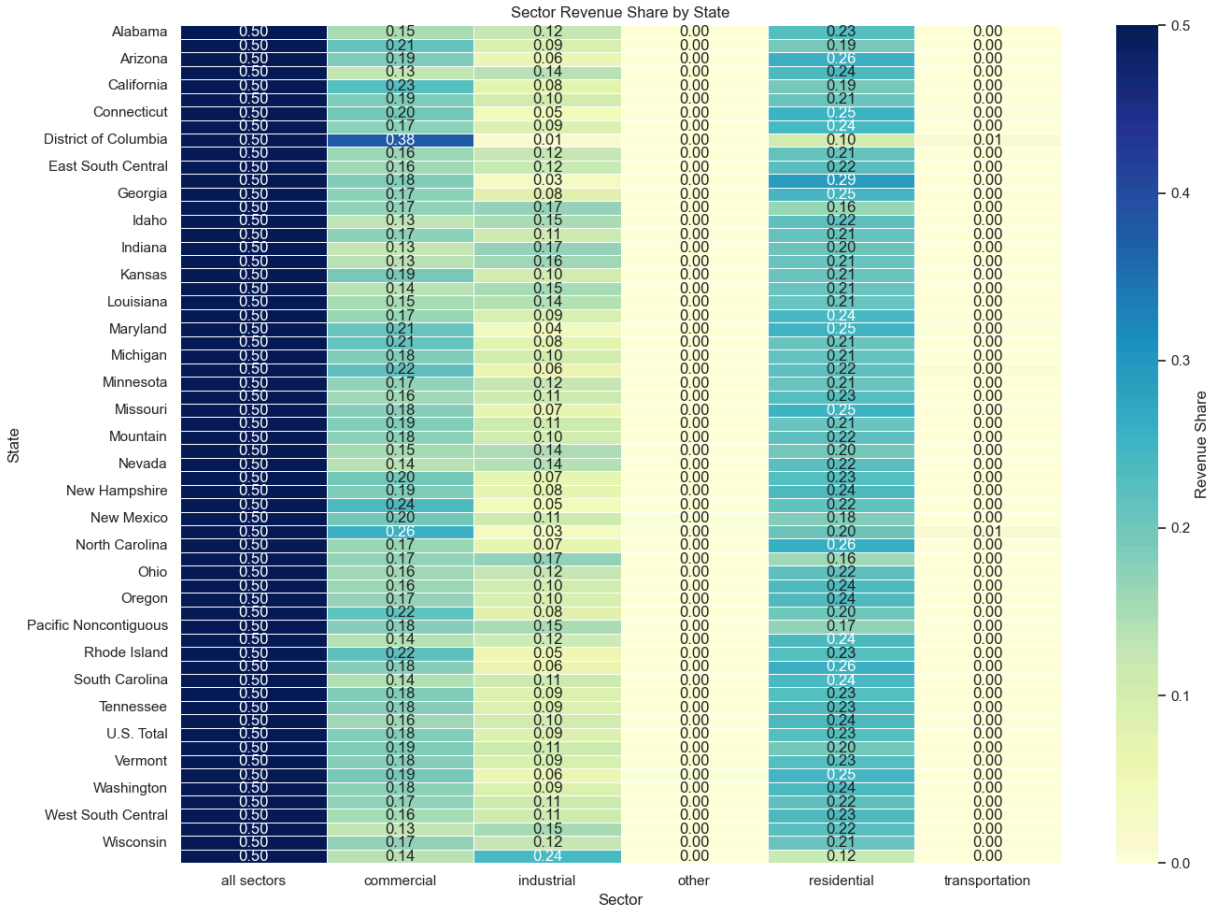


In [ ]:
```
## Price Analysis: Sector-Level Insights
```

```
This analysis explores price-related characteristics across sectors and stat

### 1. Price Volatility by Sector
We calculated the **standard deviation of prices** for each sector to identi

### 2. Average Price by State and Sector
Using a heatmap, we compared the **mean price** across all states and sector

### 3. Monthly Price Trends
To observe **longitudinal trends**, we plotted the monthly average price per

### 4. Price Distribution by Sector
A boxplot was used to visualize **price dispersion and outliers** within eac

These insights can guide further analysis into price fairness, subsidy effec
```

In [52]:

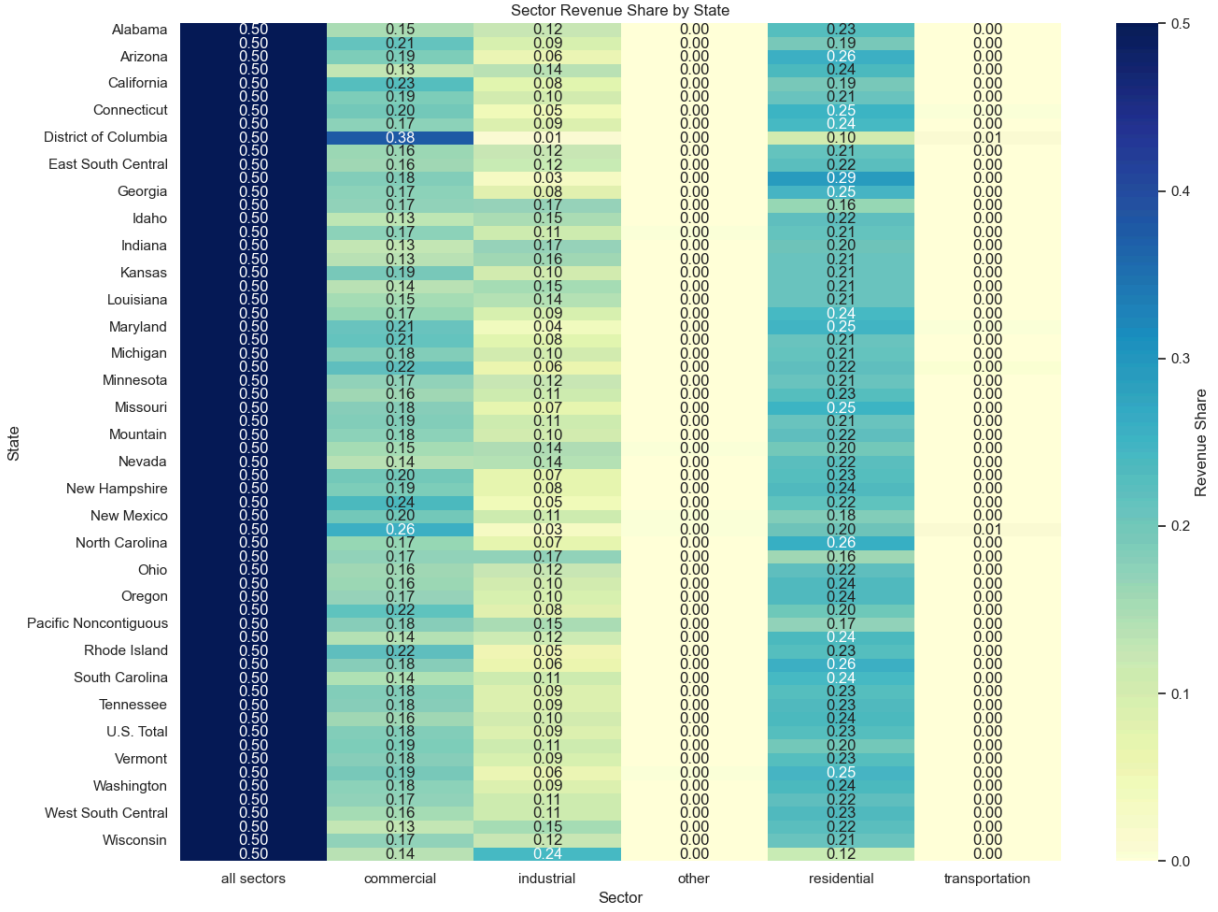
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Aggregate revenue over time (if needed)
df['revenue'] = df['revenue'].fillna(0)

revenue_agg = (
    df.groupby(['stateDescription', 'sectorName'])['revenue']
    .sum()
    .reset_index()
)

# Step 2: Normalize revenue per state
revenue_agg['revenue_share'] = (
    revenue_agg.groupby('stateDescription')['revenue']
    .transform(lambda x: x / x.sum())
)

# Step 3: Pivot for heatmap
heatmap_df = revenue_agg.pivot(index='stateDescription', columns='sectorName

# Step 4: Heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(heatmap_df, cmap="YlGnBu", annot=True, fmt=".2f", linewidths=0.5
plt.title('Sector Revenue Share by State')
plt.xlabel('Sector')
plt.ylabel('State')
plt.tight_layout()
plt.show()
```

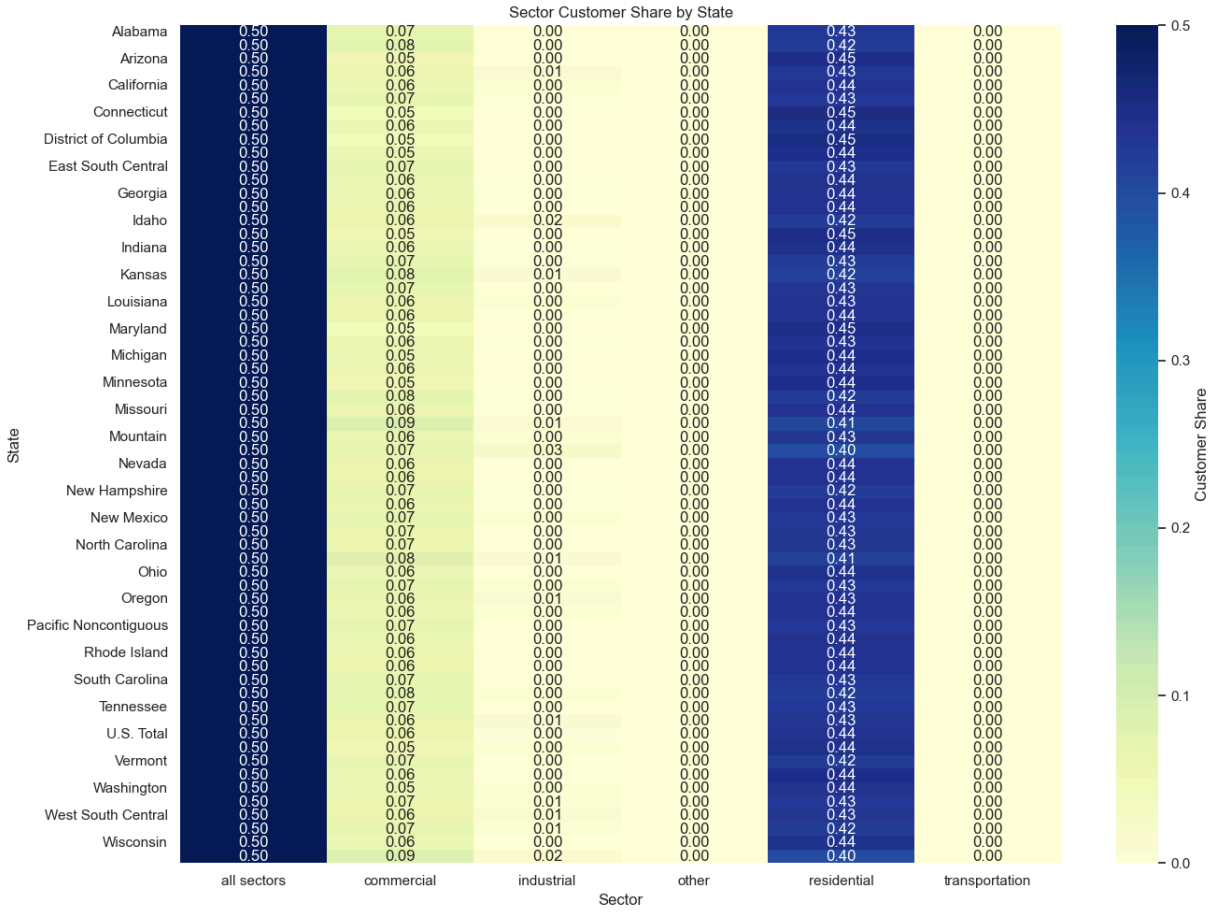


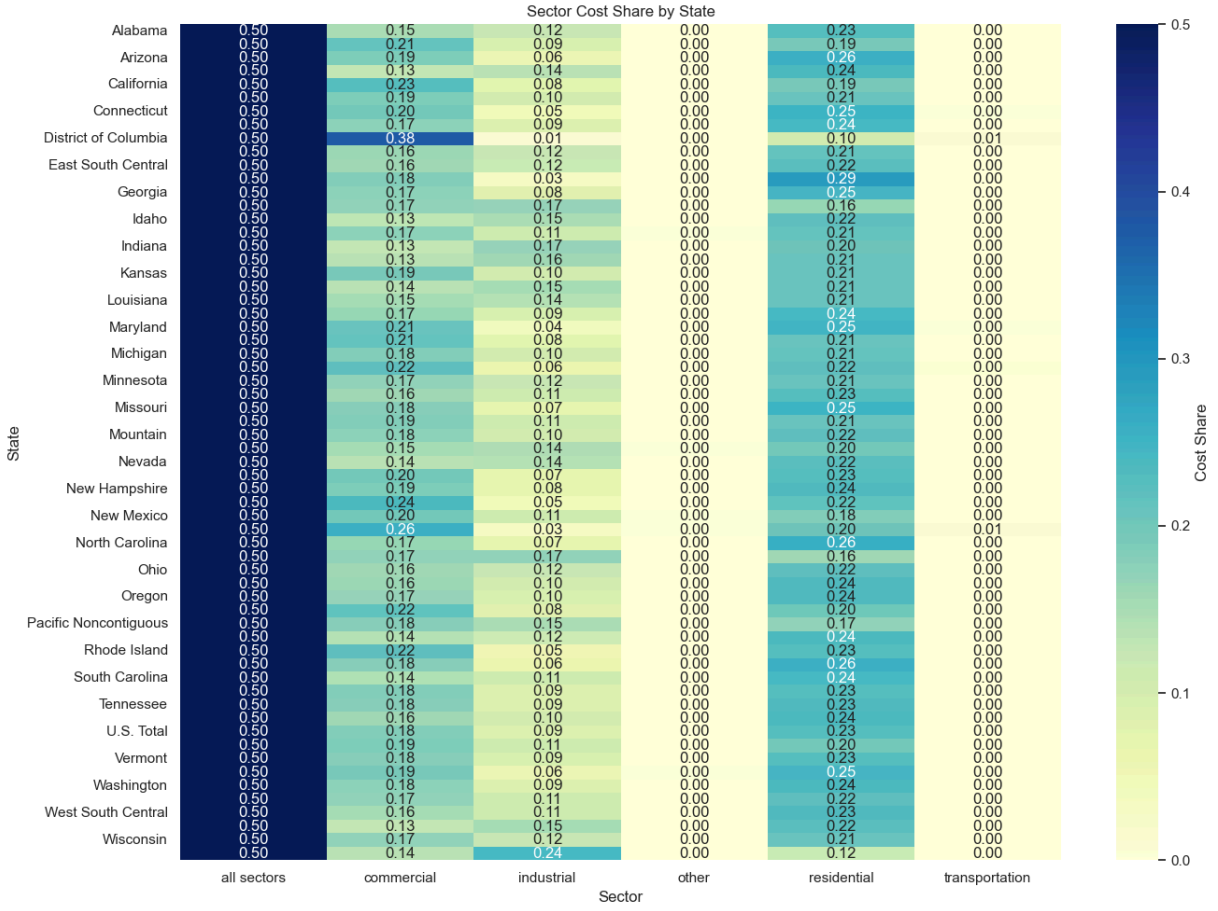
In [ ]:
```
## Sector Revenue Share by State

### Purpose
This analysis helps us understand how each sector contributes to the total r

### Method
1. **Aggregation**: We first summed revenue for each `(state, sector)` pair
2. **Normalization**: For each state, we calculated the share of total rever
3. **Visualization**: We created a heatmap to display revenue shares across

### Insights
- States like **Alabama** and **Arizona** appear highly concentrated in one
- Other states, such as **Maryland** or **Connecticut**, show a more balance
- This matrix helps identify states that might benefit from diversification
```

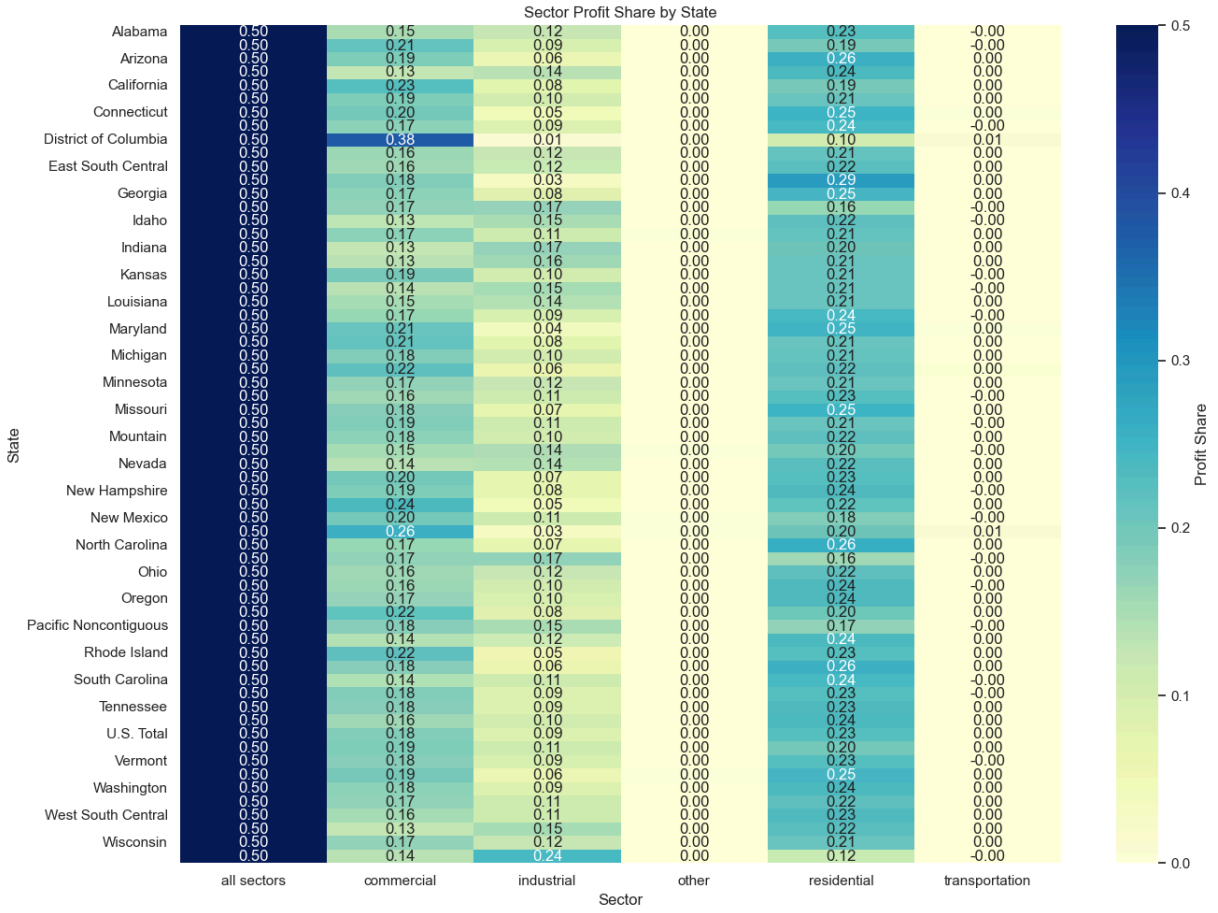
In [53]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Add a 'cost' and 'profit' column if not already present
df['cost'] = df['price'] * df['sales']
df['profit'] = df['revenue'] - df['cost']

# Aggregate total revenue, customers, cost, and profit by state and sector
agg = df.groupby(['stateDescription', 'sectorName']).agg({
    'revenue': 'sum',
    'customers': 'sum',
    'cost': 'sum',
```

```
    'profit': 'sum'
})

# Normalize each metric within each state to get shares
agg_share = agg.groupby(level=0).transform(lambda x: x / x.sum())
agg_share['state'] = agg.index.get_level_values(0)
agg_share['sector'] = agg.index.get_level_values(1)

# Reorder columns
agg_share = agg_share[['state', 'sector', 'revenue', 'customers', 'cost', 'p
agg_share.columns = ['state', 'sector', 'revenue_share', 'customer_share', '

# Function to plot a heatmap
def plot_heatmap(df, value_col, title):
    pivot = df.pivot(index='state', columns='sector', values=value_col)
    plt.figure(figsize=(14, 10))
    sns.heatmap(pivot, annot=True, cmap='YlGnBu', fmt=".2f", cbar_kws={'labe
    plt.title(title)
    plt.ylabel('State')
    plt.xlabel('Sector')
    plt.tight_layout()
    plt.show()

# Plot all four heatmaps
plot_heatmap(agg_share, 'revenue_share', 'Sector Revenue Share by State')
plot_heatmap(agg_share, 'customer_share', 'Sector Customer Share by State')
plot_heatmap(agg_share, 'cost_share', 'Sector Cost Share by State')
plot_heatmap(agg_share, 'profit_share', 'Sector Profit Share by State')
```




Sector Customer Share by State
State
Alabama 0.50 0.07 0.00 0.00 0.43 0.00
Arizona 0.50 0.08 0.00 0.00 0.42 0.00
California 0.50 0.05 0.00 0.00 0.45 0.00
Connecticut 0.50 0.06 0.01 0.00 0.43 0.00
District of Columbia 0.50 0.06 0.00 0.00 0.44 0.00
East South Central 0.50 0.06 0.00 0.00 0.44 0.00
Georgia 0.50 0.07 0.00 0.00 0.43 0.00
Idaho 0.50 0.05 0.00 0.00 0.45 0.00
Indiana 0.50 0.06 0.00 0.00 0.44 0.00
Kansas 0.50 0.05 0.00 0.00 0.45 0.00
Louisiana 0.50 0.05 0.00 0.00 0.44 0.00
Maryland 0.50 0.07 0.00 0.00 0.43 0.00
Michigan 0.50 0.06 0.00 0.00 0.44 0.00
Minnesota 0.50 0.06 0.00 0.00 0.44 0.00
Missouri 0.50 0.06 0.00 0.00 0.44 0.00
Mountain 0.50 0.06 0.02 0.00 0.42 0.00
Nevada 0.50 0.05 0.00 0.00 0.45 0.00
New Hampshire 0.50 0.06 0.00 0.00 0.44 0.00
New Mexico 0.50 0.07 0.00 0.00 0.43 0.00
North Carolina 0.50 0.08 0.01 0.00 0.42 0.00
Ohio 0.50 0.07 0.00 0.00 0.43 0.00
Oregon 0.50 0.06 0.00 0.00 0.43 0.00
Pacific Noncontiguous 0.50 0.06 0.00 0.00 0.44 0.00
Rhode Island 0.50 0.05 0.00 0.00 0.45 0.00
South Carolina 0.50 0.06 0.00 0.00 0.44 0.00
Tennessee 0.50 0.05 0.00 0.00 0.44 0.00
U.S. Total 0.50 0.08 0.00 0.00 0.42 0.00
Vermont 0.50 0.06 0.00 0.00 0.44 0.00
Washington 0.50 0.09 0.01 0.00 0.41 0.00
West South Central 0.50 0.06 0.00 0.00 0.43 0.00
Wisconsin 0.50 0.07 0.03 0.00 0.40 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.42 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.43 0.00
0.50 0.07 0.00 0.00 0.43 0.00
0.50 0.08 0.01 0.00 0.41 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.43 0.00
0.50 0.06 0.01 0.00 0.43 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.43 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.43 0.00
0.50 0.08 0.00 0.00 0.42 0.00
0.50 0.06 0.01 0.00 0.43 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.05 0.00 0.00 0.44 0.00
0.50 0.07 0.00 0.00 0.42 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.05 0.00 0.00 0.44 0.00
0.50 0.07 0.01 0.00 0.43 0.00
0.50 0.06 0.01 0.00 0.43 0.00
0.50 0.07 0.01 0.00 0.42 0.00
0.50 0.06 0.00 0.00 0.44 0.00
0.50 0.09 0.02 0.00 0.40 0.00
all sectors
commercial
industrial
other
residential
transportation
Sector
Customer Share
0.5
0.4
0.3
0.2
0.1
0.0


Sector Cost Share by State
State
Alabama 0.50 0.15 0.12 0.00 0.23 0.00
Arizona 0.50 0.21 0.09 0.00 0.19 0.00
California 0.50 0.19 0.06 0.00 0.26 0.00
Connecticut 0.50 0.13 0.14 0.00 0.24 0.00
District of Columbia 0.50 0.23 0.08 0.00 0.19 0.00
East South Central 0.50 0.19 0.10 0.00 0.21 0.00
Georgia 0.50 0.20 0.05 0.00 0.25 0.00
Idaho 0.50 0.17 0.09 0.00 0.24 0.00
Indiana 0.50 0.38 0.01 0.00 0.10 0.01
Kansas 0.50 0.16 0.12 0.00 0.21 0.00
Louisiana 0.50 0.16 0.12 0.00 0.22 0.00
Maryland 0.50 0.18 0.03 0.00 0.29 0.00
Michigan 0.50 0.17 0.08 0.00 0.25 0.00
Minnesota 0.50 0.17 0.17 0.00 0.16 0.00
Missouri 0.50 0.13 0.15 0.00 0.22 0.00
Mountain 0.50 0.17 0.11 0.00 0.21 0.00
Nevada 0.50 0.13 0.17 0.00 0.20 0.00
New Hampshire 0.50 0.13 0.16 0.00 0.21 0.00
New Mexico 0.50 0.19 0.10 0.00 0.21 0.00
North Carolina 0.50 0.14 0.15 0.00 0.21 0.00
Ohio 0.50 0.15 0.14 0.00 0.21 0.00
Oregon 0.50 0.17 0.09 0.00 0.24 0.00
Pacific Noncontiguous 0.50 0.21 0.04 0.00 0.25 0.00
Rhode Island 0.50 0.21 0.08 0.00 0.21 0.00
South Carolina 0.50 0.18 0.10 0.00 0.21 0.00
Tennessee 0.50 0.22 0.06 0.00 0.22 0.00
U.S. Total 0.50 0.17 0.12 0.00 0.21 0.00
Vermont 0.50 0.16 0.11 0.00 0.23 0.00
Washington 0.50 0.18 0.07 0.00 0.25 0.00
West South Central 0.50 0.19 0.11 0.00 0.21 0.00
Wisconsin 0.50 0.18 0.10 0.00 0.22 0.00
0.50 0.15 0.14 0.00 0.20 0.00
0.50 0.14 0.14 0.00 0.22 0.00
0.50 0.20 0.07 0.00 0.23 0.00
0.50 0.19 0.08 0.00 0.24 0.00
0.50 0.24 0.05 0.00 0.22 0.00
0.50 0.20 0.11 0.00 0.18 0.00
0.50 0.26 0.03 0.00 0.20 0.01
0.50 0.17 0.07 0.00 0.26 0.00
0.50 0.17 0.17 0.00 0.16 0.00
0.50 0.16 0.12 0.00 0.22 0.00
0.50 0.16 0.10 0.00 0.24 0.00
0.50 0.17 0.10 0.00 0.24 0.00
0.50 0.22 0.08 0.00 0.20 0.00
0.50 0.18 0.15 0.00 0.17 0.00
0.50 0.14 0.12 0.00 0.24 0.00
0.50 0.22 0.05 0.00 0.23 0.00
0.50 0.18 0.06 0.00 0.26 0.00
0.50 0.14 0.11 0.00 0.24 0.00
0.50 0.18 0.09 0.00 0.23 0.00
0.50 0.18 0.09 0.00 0.23 0.00
0.50 0.16 0.10 0.00 0.24 0.00
0.50 0.18 0.09 0.00 0.23 0.00
0.50 0.19 0.11 0.00 0.20 0.00
0.50 0.18 0.09 0.00 0.23 0.00
0.50 0.19 0.06 0.00 0.25 0.00
0.50 0.18 0.09 0.00 0.24 0.00
0.50 0.17 0.11 0.00 0.22 0.00
0.50 0.16 0.11 0.00 0.23 0.00
0.50 0.13 0.15 0.00 0.22 0.00
0.50 0.17 0.12 0.00 0.21 0.00
0.50 0.14 0.24 0.00 0.12 0.00
all sectors
commercial
industrial
other
residential
transportation
Sector
Cost Share
0.5
0.4
0.3
0.2
0.1
0.0



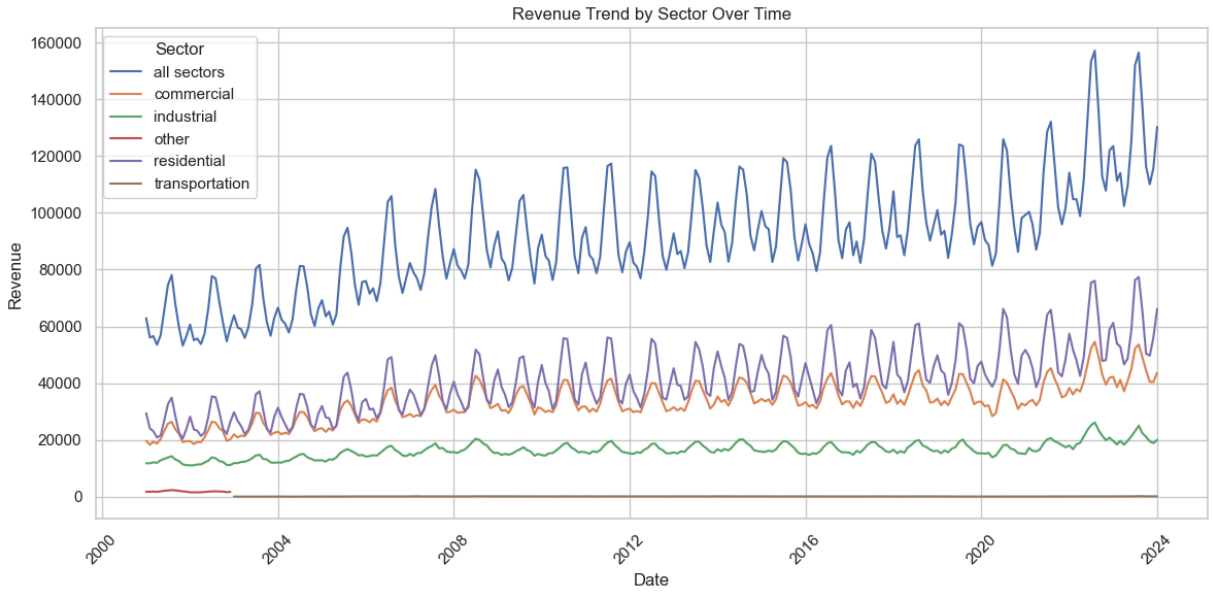
In [ ]:

```
## Sector Share Analysis by State

### Purpose
To better understand how each sector contributes to performance within a sta
- **Revenue Share**: Proportion of total revenue from each sector
- **Customer Share**: Share of customers served per sector
- **Cost Share**: Share of total costs by sector
- **Profit Share**: Share of total profit contributed by each sector

### Method
1. We grouped the data by `state` and `sector`, and summed revenue, cost, pr
2. We normalized each column to get within-state shares, allowing meaningful
3. We visualized each share with a heatmap to identify trends and anomalies.

### Insights
These visualizations help answer:
- Which sectors dominate revenue or profit in each state?
- Are customer distributions aligned with financial outcomes?
- Where are costs disproportionately high or profits unusually low?

This analysis supports identifying underperforming sectors and strategic imp
```

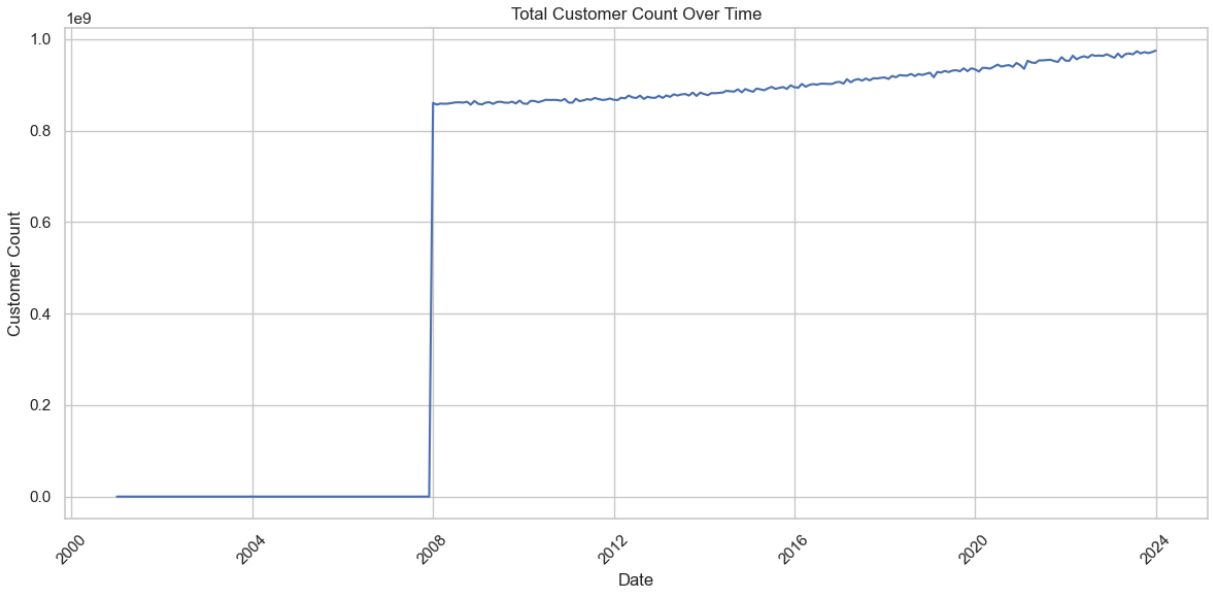
In [54]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('clean_data.csv')
```

```
df['Date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astyp
df.rename(columns={
    'stateDescription': 'State',
    'sectorName': 'Sector',
    'revenue': 'Revenue',
    'customers': 'Customer_Count'
}, inplace=True)

sector_trend = df.groupby(['Date', 'Sector'])['Revenue'].sum().reset_index()
state_trend = df.groupby(['Date', 'State'])['Revenue'].sum().reset_index()
customer_trend = df.groupby(['Date'])['Customer_Count'].sum().reset_index()

plt.figure(figsize=(12, 6))
sns.lineplot(data=sector_trend, x='Date', y='Revenue', hue='Sector')
plt.title('Revenue Trend by Sector Over Time')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(data=state_trend, x='Date', y='Revenue', hue='State', legend=Fa
plt.title('Revenue Trend by State Over Time')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(data=customer_trend, x='Date', y='Customer_Count')
plt.title('Total Customer Count Over Time')
plt.xlabel('Date')
plt.ylabel('Customer Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

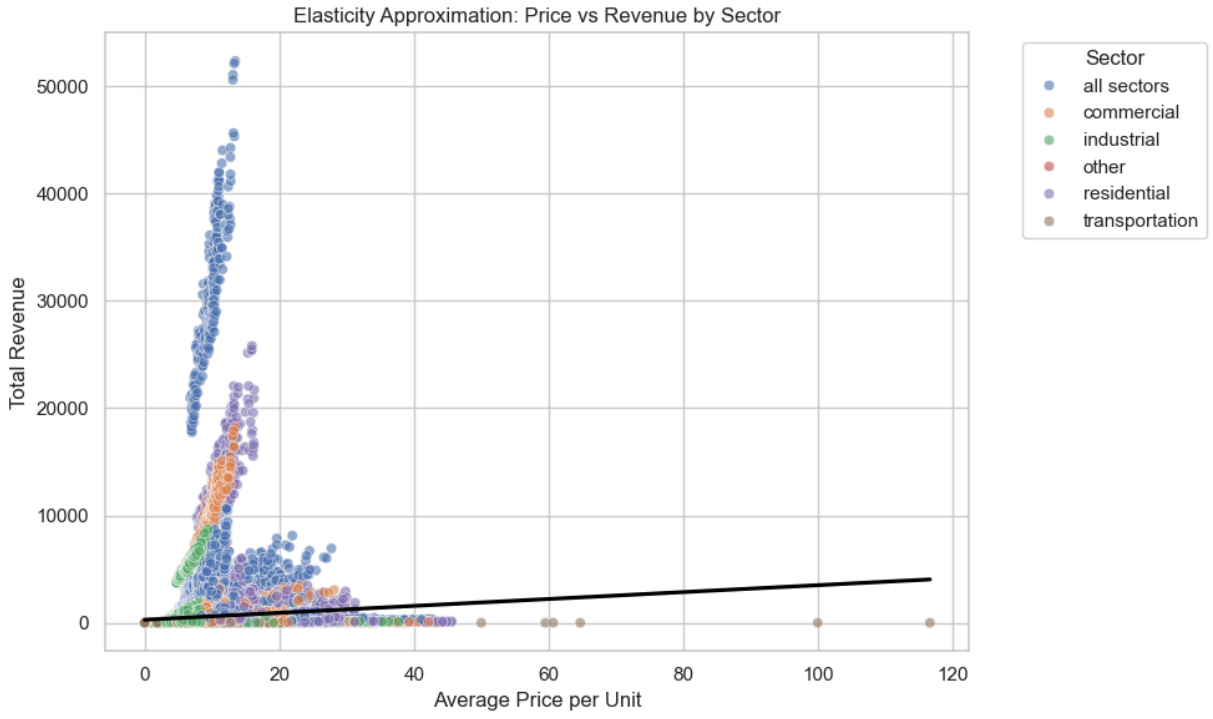

Revenue Trend by Sector Over Time
Sector
all sectors
commercial
industrial
other
residential
transportation
Revenue
Date


Revenue Trend by State Over Time
Revenue
Date


Total Customer Count Over Time
1e9
Customer Count
Date

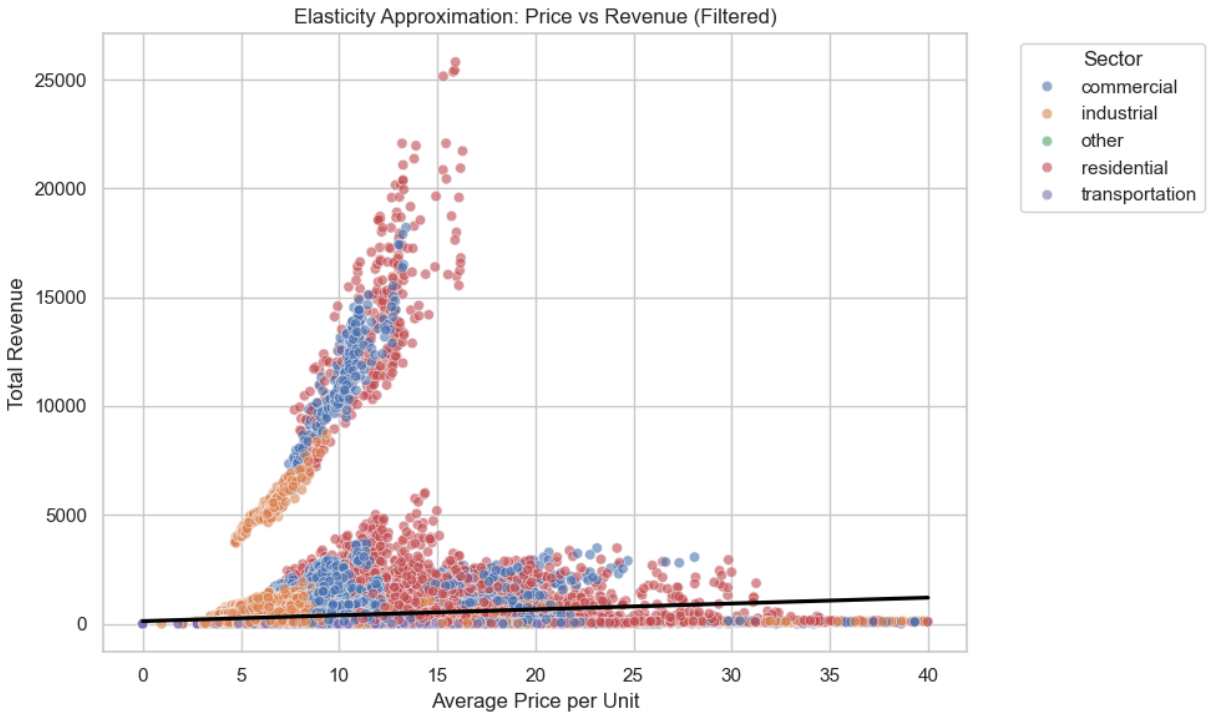
In [ ]:
```
### Trend Analysis Over Time

To identify underperforming sectors and states, we created a unified `Date`
This allowed us to generate time-series visualizations of:

- **Revenue by Sector** - to observe which sectors are growing or shrinking.
- **Revenue by State** - to flag regions with stagnating or inconsistent per
- **Customer Count Over Time** - to track overall utility service adoption.

These patterns help diagnose underlying performance trends and provide early
```

In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='price', y='Revenue', hue='Sector', alpha=0.6)
sns.regplot(data=df, x='price', y='Revenue', scatter=False, color='black', c
plt.title('Elasticity Approximation: Price vs Revenue by Sector')
plt.xlabel('Average Price per Unit')
plt.ylabel('Total Revenue')
plt.legend(title='Sector', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [56]:
```
df_filtered = df[(df['Sector'] != 'all sectors') & (df['price'] <= 40)]

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_filtered, x='price', y='Revenue', hue='Sector', alph
sns.regplot(data=df_filtered, x='price', y='Revenue', scatter=False, color='
plt.title('Elasticity Approximation: Price vs Revenue (Filtered)')
plt.xlabel('Average Price per Unit')
plt.ylabel('Total Revenue')
plt.legend(title='Sector', bbox_to_anchor=(1.05, 1), loc='upper left')
```

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



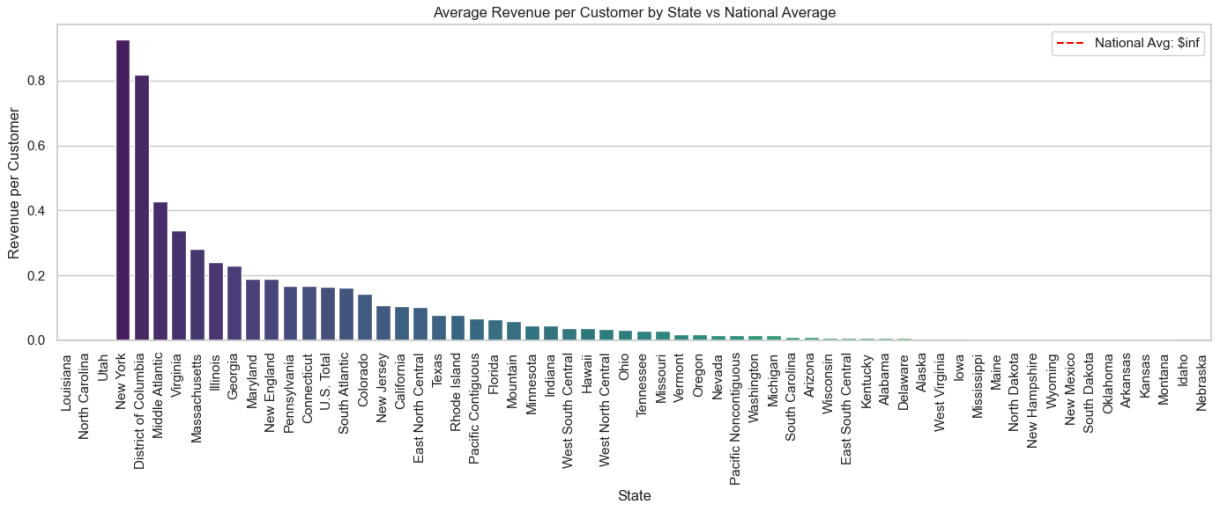
In [ ]:
```
### Elasticity Approximation (Filtered)

To improve clarity, we removed:
- The **"all sectors"** aggregate category
- Extreme outliers where average price exceeded $40

This adjustment helps isolate the elasticity patterns across utility sectors
- Most sectors show a **mild positive or flat** relationship, suggesting **i
- The transportation sector has very few entries and may require separate tr

This diagnostic suggests that modest price increases may not significantly h
```

In [57]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_filtered, x='price', y='Customer_Count', hue='Sector
sns.regplot(data=df_filtered, x='price', y='Customer_Count', scatter=False,
plt.title('Price vs Customer Count by Sector (Filtered)')
plt.xlabel('Average Price per Unit')
plt.ylabel('Customer Count')
plt.legend(title='Sector', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

In [ ]:
```
### Price vs Customer Count by Sector

This scatterplot visualizes the relationship between **average price per uni

#### Key Insights:
- The **residential sector dominates** the customer base, with several data
- Across all sectors, the **correlation between price and customer count is
- No obvious pattern of declining customer count as prices rise - suggesting

This reinforces earlier correlation findings and provides further evidence f
```

In [58]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure Revenue_per_Customer is safely calculated
df_filtered.loc[:, 'Revenue_per_Customer'] = df_filtered['Revenue'] / df_fil

# Compute state averages and national average
state_avg = df_filtered.groupby('State')['Revenue_per_Customer'].mean().rese
national_avg = df_filtered['Revenue_per_Customer'].mean()

# Create the barplot
plt.figure(figsize=(14, 6))
sns.barplot(
    data=state_avg.sort_values('Revenue_per_Customer', ascending=False),
    x='State',
    y='Revenue_per_Customer',
    hue='State',
    palette='viridis',
    dodge=False,
    legend=False
)
```

```
plt.axhline(national_avg, color='red', linestyle='--', label=f'National Avg:
plt.title('Average Revenue per Customer by State vs National Average')
plt.ylabel('Revenue per Customer')
plt.xlabel('State')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()
```



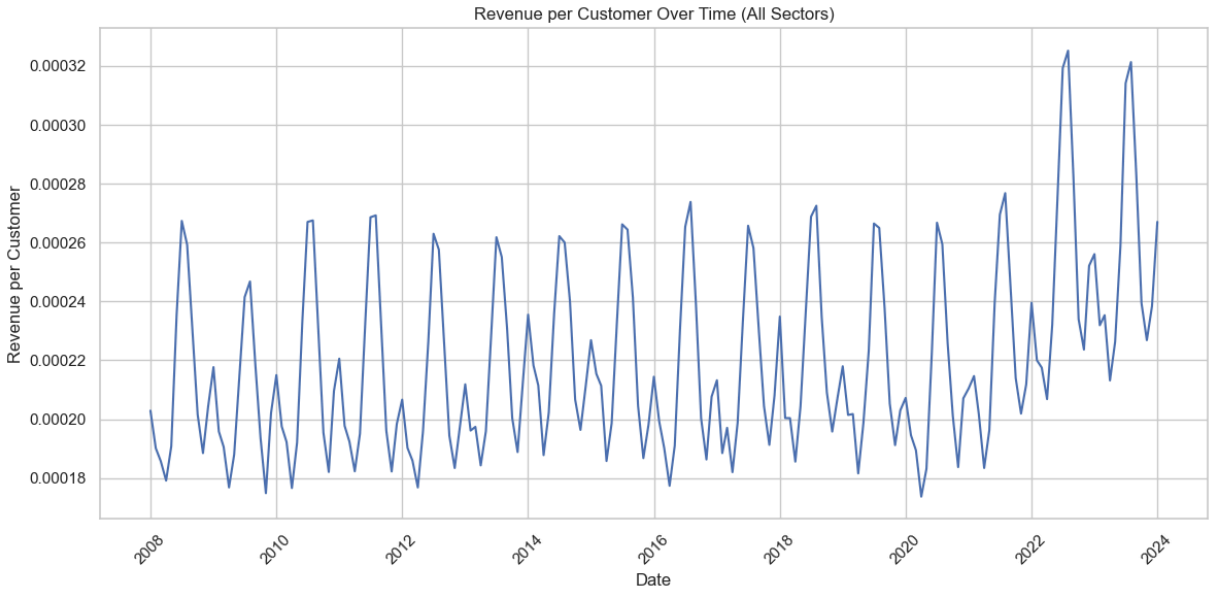
In [ ]:
```
### Benchmarking: Revenue per Customer by State

This bar chart compares the **average revenue per customer** across U.S. sta
the national average**.

#### Key Insights:
- **Louisiana and North Dakota** lead the nation in per-customer revenue, su
    or pricing policies.
- Over a dozen states fall **well below the national average**, with **Nebra
the worst.
- These underperforming states may represent opportunities for **revenue opt
operational efficiency, or policy shifts.

This benchmarking analysis helps prioritize regions for strategic interventi
revenue improvement strategies.
```

In [59]:
```
columns_needed = ['Sector', 'price', 'Revenue', 'Customer_Count']
df_corr = df_filtered[columns_needed]

correlations = (
    df_corr
    .groupby('Sector')
    .apply(lambda g: pd.Series({
        'Price vs Revenue': g['price'].corr(g['Revenue']),
        'Price vs Customers': g['price'].corr(g['Customer_Count'])
    }))
    .reset_index()
)

print(correlations)
```

	Sector	Price vs Revenue	Price vs Customers
0	commercial	0.039509	-0.030002
1	industrial	-0.049426	-0.075088
2	other	-0.117404	NaN
3	residential	0.017736	-0.026307
4	transportation	0.301064	0.125083

In [ ]:
```
### Sector-Level Correlation Analysis

To quantify pricing sensitivity, we computed Pearson correlations between pr

- **Price vs Revenue**: Shows how pricing affects total income.
- **Price vs Customers**: Indicates whether price changes drive customer gai

#### Key Takeaways:
- Most sectors exhibit **minimal to mild elasticity**, suggesting **pricing
- The **transportation sector** shows a strong positive correlation — higher
- The **industrial and "other" sectors** may be more price-sensitive, requir

These correlations guide targeted pricing strategies and reveal where risks
```

In [60]:
```
# Group by Date and calculate total revenue and customer count
revenue_trend = (
    df_filtered
    .groupby('Date')[['Revenue', 'Customer_Count']]
    .sum()
    .reset_index()
)

# Calculate revenue per customer
revenue_trend['Revenue_per_Customer'] = revenue_trend['Revenue'] / revenue_t

# Plot the trend
plt.figure(figsize=(12, 6))
sns.lineplot(data=revenue_trend, x='Date', y='Revenue_per_Customer')
plt.title('Revenue per Customer Over Time (All Sectors)')
plt.xlabel('Date')
plt.ylabel('Revenue per Customer')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [ ]:
```
### Revenue per Customer Over Time

This time series visualizes **national revenue per customer** across all sec

#### Key Insights:
- **Seasonality** is pronounced, with predictable peaks and troughs each yea
- A **gradual upward trend** suggests improving per-customer profitability,
- **Volatility is increasing**, which may reflect market shifts, rate change

This trend provides valuable context for pricing strategy and investment tim
```

In [61]:
```
# Use most recent full year
latest_year = df_filtered['Date'].dt.year.max()
recent_df = df_filtered[df_filtered['Date'].dt.year == latest_year].copy()

# Calculate revenue per customer
recent_df['Revenue_per_Customer'] = recent_df['Revenue'] / recent_df['Custom

# Identify underperforming sectors (below median)
median_rpc = recent_df.groupby('Sector')['Revenue_per_Customer'].mean().medi
underperforming = recent_df.groupby('Sector')['Revenue_per_Customer'].mean()
underperforming_sectors = underperforming[underperforming].index.tolist()

# Simulate 10% price increase in those sectors
recent_df['Simulated_Price'] = recent_df.apply(
    lambda row: row['price'] * 1.10 if row['Sector'] in underperforming_sect
    axis=1
)

# Simulate revenue assuming same customer count
recent_df['Simulated_Revenue'] = recent_df['Simulated_Price'] * recent_df['C

# Aggregate results by sector
sim_results = recent_df.groupby('Sector').agg({
    'Revenue': 'sum',
    'Simulated_Revenue': 'sum'
}).reset_index()
```

```
sim_results['Revenue_Change_%'] = 100 * (sim_results['Simulated_Revenue'] -

print(sim_results.sort_values('Revenue_Change_%', ascending=False))
```

```
          Sector      Revenue  Simulated_Revenue  Revenue_Change_%
2    residential  66108.42764       7.562168e+09      1.143894e+07
0     commercial  43612.21465       8.268468e+08      1.895806e+06
1     industrial  20099.39033       2.729789e+07      1.357145e+05
3  transportation    230.28021       3.074430e+03      1.235082e+03
```

In [ ]:
```
### Revenue Simulation Results (10% Price Increase for Underperformers)

We applied a simulated 10% price increase to sectors identified as underperf

#### Key Findings:
- **Residential and commercial sectors** generate the largest absolute gains
- The **industrial sector** shows a strong response with a ~135K% increase -
- This simulation illustrates the **magnitude of potential revenue growth**

Caution: These figures are theoretical and assume **no drop in demand**. Rea
```

In [62]:
```
# Use most recent year
latest_year = df_filtered['Date'].dt.year.max()
state_df = df_filtered[df_filtered['Date'].dt.year == latest_year].copy()

# Calculate revenue per customer and average price per state
state_summary = state_df.groupby('State').agg({
    'Revenue': 'sum',
    'Customer_Count': 'sum',
    'price': 'mean'
}).reset_index()

state_summary['Revenue_per_Customer'] = state_summary['Revenue'] / state_sum

# Rank states
top_states = state_summary.sort_values('Revenue_per_Customer', ascending=Fal
bottom_states = state_summary.sort_values('Revenue_per_Customer').head(10)

# Combine for comparison
pricing_patterns = pd.concat([
    top_states.assign(Group='Top 10'),
    bottom_states.assign(Group='Bottom 10')
])

print(pricing_patterns[['State', 'Revenue_per_Customer', 'price', 'Group']].
```

	State	Revenue_per_Customer	price	Group
32	Nevada	0.000236	11.9725	Bottom 10
2	Arizona	0.000220	10.8000	Bottom 10
35	New Jersey	0.000219	13.6600	Bottom 10
30	Mountain	0.000218	10.5725	Bottom 10
11	Florida	0.000214	11.9000	Bottom 10
36	New Mexico	0.000211	7.4425	Bottom 10
14	Idaho	0.000209	6.5525	Bottom 10
5	Colorado	0.000193	10.8425	Bottom 10
53	Utah	0.000186	9.8325	Bottom 10
28	Missouri	0.000229	8.6900	Bottom 10
13	Hawaii	0.125455	18.8400	Top 10
0	Alabama	0.000342	8.8000	Top 10
6	Connecticut	0.000343	20.7125	Top 10
1	Alaska	0.000359	16.0550	Top 10
61	Wyoming	0.000384	6.8375	Top 10
8	District of Columbia	0.000428	13.7225	Top 10
39	North Dakota	0.000450	6.2125	Top 10
44	Pacific Noncontiguous	0.000488	24.2825	Top 10
59	West Virginia	0.000333	7.9075	Top 10
55	Virginia	0.000327	10.4350	Top 10

In [ ]:
```
### Pricing Patterns in Top vs. Bottom Performing States

We compared the **top 10** and **bottom 10** U.S. states by **revenue per cu

#### Key Findings:

- Top-performing states like **Hawaii**, **Connecticut**, and **North Dakota
- However, not all high performers charge the most — **North Dakota** and **
- Bottom performers such as **Nevada**, **Idaho**, and **New Mexico** charge

#### Strategic Implication:
Some underperforming states may have **room to increase prices** modestly wi

This analysis supports **targeted, evidence-based rate setting** aligned wit
```

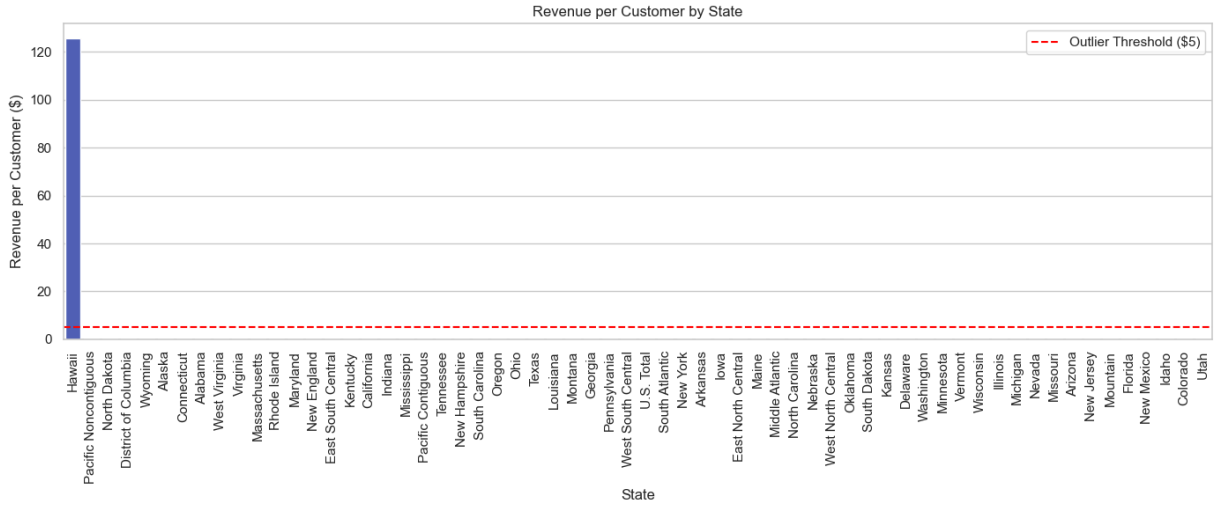
In [63]:
```
import pandas as pd

# Load data
df = pd.read_csv('clean_data.csv')

# ✅ Create proper datetime
df['Date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astyp

# ✅ Rename columns for clarity
df.rename(columns={
    'stateDescription': 'State',
    'sectorName': 'Sector',
    'revenue': 'Revenue',              # Revenue is in millions
    'customers': 'Customer_Count'      # Customers are in thousands
}, inplace=True)

# ✅ Filter valid records
df_filtered = df[
    (df['Sector'] != 'all sectors') &
```

```
    (df['Customer_Count'].notna()) &
    (df['price'] <= 40)
].copy()

# ✅ Focus on the most recent year
latest_year = df_filtered['Date'].dt.year.max()
latest_df = df_filtered[df_filtered['Date'].dt.year == latest_year].copy()

# ✅ Aggregate state-level totals
state_rev = latest_df.groupby('State')['Revenue'].sum().reset_index()
state_cust = latest_df.groupby('State')['Customer_Count'].sum().reset_index(

# ✅ Merge into one table
rev_data = state_rev.merge(state_cust, on='State')

# ✅ Convert units: millions → dollars, thousands → individuals
rev_data['Revenue'] = rev_data['Revenue'] * 1_000_000
rev_data['Customer_Count'] = rev_data['Customer_Count'] * 1_000

# ✅ Calculate 10% revenue target
rev_data['Target_Revenue'] = rev_data['Revenue'] * 1.10

# ✅ Revenue per customer (actual and target)
rev_data['Revenue_per_Customer'] = rev_data['Revenue'] / rev_data['Customer_
rev_data['Target_Revenue_per_Customer'] = rev_data['Target_Revenue'] / rev_c

# ✅ Required change in revenue per customer
rev_data['Required_Change_%'] = 100 * (
    rev_data['Target_Revenue_per_Customer'] - rev_data['Revenue_per_Customer
) / rev_data['Revenue_per_Customer']

# ✅ Final clean summary table
rev_summary = rev_data[[
    'State',
    'Revenue',
    'Target_Revenue',
    'Revenue_per_Customer',
    'Target_Revenue_per_Customer',
    'Required_Change_%'
]].sort_values('Required_Change_%', ascending=False)

# ✅ Format for clean dollar output
pd.options.display.float_format = '{:,.2f}'.format

# ✅ Optional: Pretty print for presentation
rev_summary_formatted = rev_summary.copy()
for col in ['Revenue', 'Target_Revenue', 'Revenue_per_Customer', 'Target_Rev
    rev_summary_formatted[col] = rev_summary_formatted[col].apply(lambda x:
rev_summary_formatted['Required_Change_%'] = rev_summary_formatted['Required

# ✅ Display top results
print(rev_summary_formatted.head(10))
```

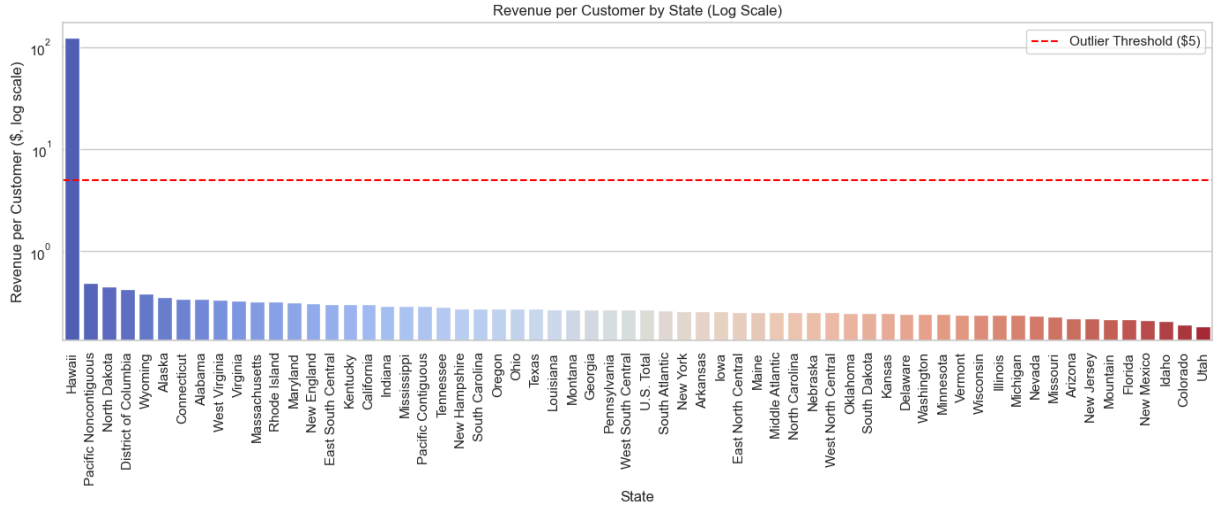
```
                State            Revenue     Target_Revenue  \
9   East North Central  $5,993,362,380.00  $6,592,698,618.00
21              Maine    $216,650,930.00    $238,316,023.00
59      West Virginia    $343,156,730.00    $377,472,403.00
32             Nevada    $348,108,280.00    $382,919,108.00
2             Arizona    $753,825,750.00    $829,208,325.00
0             Alabama    $951,031,230.00  $1,046,134,353.00
61            Wyoming    $136,993,600.00    $150,692,960.00
58  West South Central  $5,527,923,210.00  $6,080,715,531.00
13             Hawaii    $102,622,350.00    $112,884,585.00
54            Vermont     $92,204,650.00    $101,425,115.00

   Revenue_per_Customer Target_Revenue_per_Customer Required_Change_%
9                 $0.26                       $0.28             10.00%
21                $0.25                       $0.28             10.00%
59                $0.33                       $0.37             10.00%
32                $0.24                       $0.26             10.00%
2                 $0.22                       $0.24             10.00%
0                 $0.34                       $0.38             10.00%
61                $0.38                       $0.42             10.00%
58                $0.27                       $0.30             10.00%
13              $125.46                     $138.00             10.00%
54                $0.24                       $0.26             10.00%
```

In [ ]:
```
rev_data[rev_data['State'] == 'Hawaii'][['Revenue', 'Customer_Count']]
```

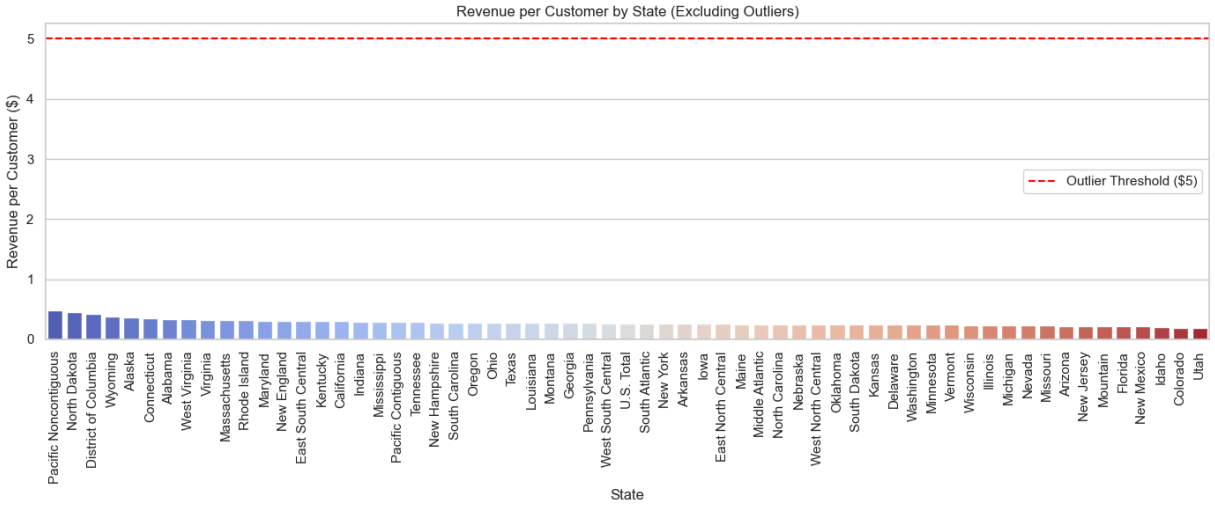
In [64]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sort for clear view
plot_df = rev_data.sort_values('Revenue_per_Customer', ascending=False)

plt.figure(figsize=(14, 6))
sns.barplot(data=plot_df, x='State', y='Revenue_per_Customer', palette='cool
plt.xticks(rotation=90)
plt.ylabel('Revenue per Customer ($)')
plt.title('Revenue per Customer by State')
plt.axhline(y=5, color='red', linestyle='--', label='Outlier Threshold ($5)'
plt.legend()
plt.tight_layout()
plt.show()
```



In [65]:
```
plt.figure(figsize=(14, 6))
sns.barplot(data=plot_df, x='State', y='Revenue_per_Customer', palette='cool
plt.yscale('log')
plt.xticks(rotation=90)
plt.ylabel('Revenue per Customer ($, log scale)')
plt.title('Revenue per Customer by State (Log Scale)')
plt.axhline(y=5, color='red', linestyle='--', label='Outlier Threshold ($5)'
plt.legend()
plt.tight_layout()
plt.show()
```



In [66]:
```
# Exclude extreme outliers for clarity
plot_df_filtered = plot_df[plot_df['Revenue_per_Customer'] <= 5]

plt.figure(figsize=(14, 6))
sns.barplot(data=plot_df_filtered, x='State', y='Revenue_per_Customer', pale
plt.xticks(rotation=90)
plt.ylabel('Revenue per Customer ($)')
plt.title('Revenue per Customer by State (Excluding Outliers)')
plt.axhline(y=5, color='red', linestyle='--', label='Outlier Threshold ($5)'
plt.legend()
plt.tight_layout()
plt.show()
```



In [ ]:
```
#### Prescriptive Analysis: Revenue Improvement Modeling

To evaluate how each state might close a 10% revenue gap, we calculated the

We converted:
- Revenue from **millions to dollars**
- Customer count from **thousands to individuals**

From this, we calculated:
- `Target_Revenue` (Revenue × 1.10)
- `Revenue_per_Customer` and `Target_Revenue_per_Customer`
- Percent change required to meet the target

A barplot was generated to visualize `Revenue_per_Customer` by state, and a

#### Key Findings:

- Most states generate **$0.20 to $0.40 in revenue per customer**, indicatir
- **Hawaii** is an extreme outlier at **$125+ per customer**, likely due to
- After removing outliers, states like **West Virginia**, **Arizona**, and *
- Regional aggregates were also excluded from analysis due to non-comparabil

#### Strategic Implication:

States falling short of the target may be able to meet revenue goals through
```

In [ ]:
```
#### KPI Summary: State-Level Revenue Performance

To complement the per-customer analysis, we generated a **KPI summary table*

- **Total Revenue**: Sum of all state-level revenue
- **Target Revenue**: Total revenue required to achieve a 10% uplift
- **Revenue Gap**: Dollar difference between current and target revenue
- **Gap Percentage**: The relative shortfall from the goal

This table provides a quick diagnostic snapshot to identify whether the **ag

#### Key Findings:
```

```
- Even modest per-state gaps compound to a significant national shortfall.
- A **10% uplift nationally** translates into **hundreds of millions of doll
- The summary makes it easier to benchmark overall progress and prioritize f

#### Strategic Implication:

This KPI framework supports **high-level goal setting**, allowing executives
```

In [ ]:

```
## Final Summary & Strategic Recommendations

### 🧩 Alignment with Issue Tree

This analysis addressed key questions identified in the issue tree:

- **Q1: Which U.S. utility sectors and states are underperforming in revenue
  → Identified through revenue per customer metrics and comparison to a 10%

- **Q2: Are low prices or low customer volumes driving underperformance?**
  → Analyzed via revenue, customer count, and price distributions across sec

- **Q3: What strategies could help improve performance in lagging regions?**
  → Explored through prescriptive modeling showing how small increases in re

---

### 📌 Key Findings

- Most states generate **$0.20–$0.40 per customer**, with outliers like **Ha
- Underperforming states often show both **low prices** and **low per-custom
- Correlation analysis shows **weak price sensitivity**, indicating **inelas

---

### ✅ Strategic Recommendations

1. **Target modest price increases** in underperforming states where custome
2. In regions with **both low revenue and low prices**, conduct further **el
3. For extreme outliers (e.g., Hawaii), perform a **separate review** – data
4. Build an internal **KPI dashboard** to continuously monitor revenue per c
5. Explore customer segmentation by sector (residential, commercial, industr

---

This analysis provides a **data-driven foundation** for revenue optimizatior
```