Multi-Level Sunburst Charts using Plotly and Python

Multi-level sunburst charts represent one of the most elegant and effective ways to visualize hierarchical data structures in Python. These circular, radial visualizations excel at displaying complex tree-like data relationships through concentric rings, where each level of the hierarchy is represented by a distinct ring extending outward from the center. With Plotly's comprehensive API, creating interactive and customizable sunburst charts has become increasingly accessible to data scientists and analysts working with hierarchical datasets.

Understanding Sunburst Charts

What Are Sunburst Charts?

Sunburst charts are a specialized form of data visualization that displays hierarchical data using a radial layout[1]. They consist of concentric circles or rings, where the innermost circle represents the root level of the hierarchy, and each subsequent ring represents deeper levels of categorization. The angular width of each segment is proportional to the value it represents, making it easy to compare relative sizes across different categories and levels.

These charts are particularly effective for visualizing **part-to-whole relationships** at multiple levels simultaneously[2]. Unlike traditional tree diagrams or hierarchical charts that use linear layouts, sunburst charts utilize circular space more efficiently, allowing for better representation of complex hierarchical structures without consuming excessive screen real estate.

Key Characteristics and Benefits

Sunburst charts offer several distinct advantages over traditional hierarchical visualizations:

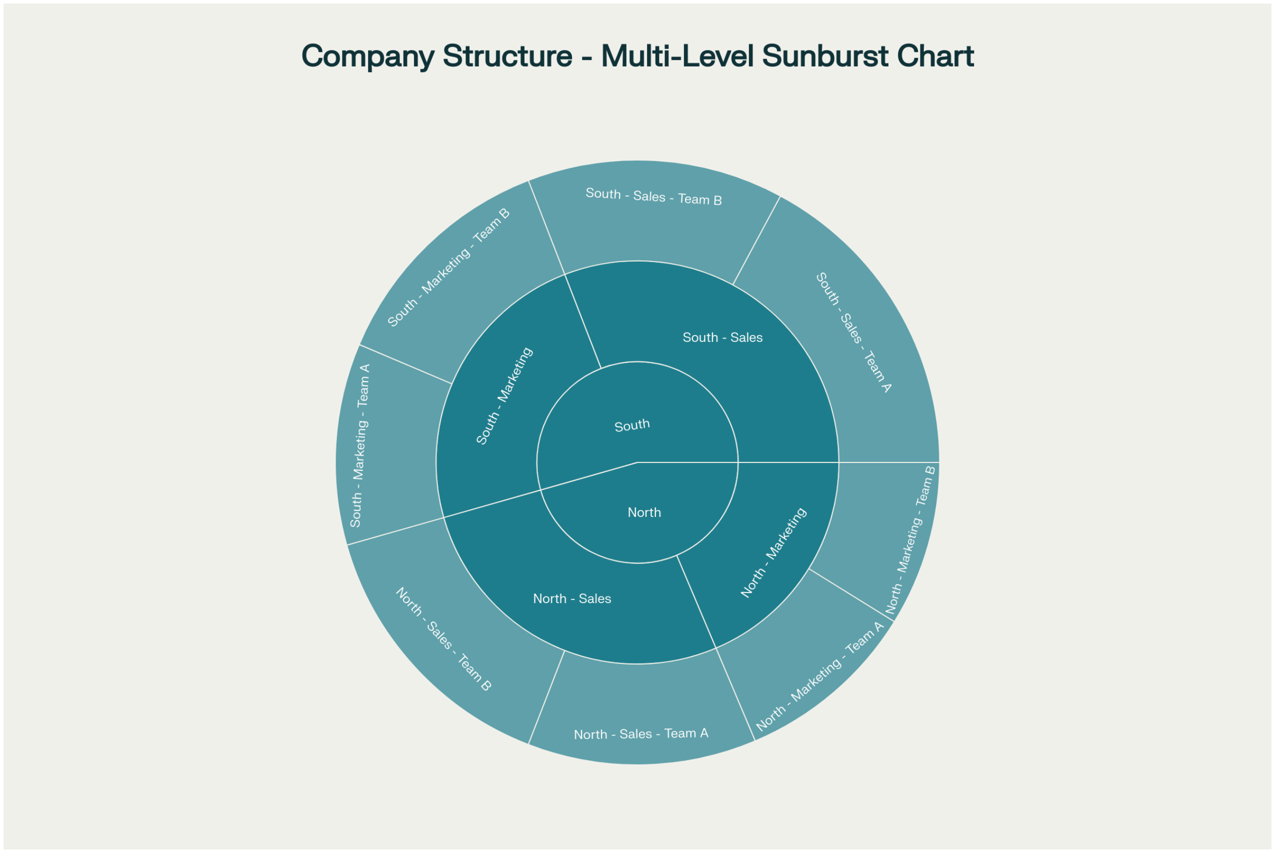
- **Space Efficiency**: The radial layout maximizes the use of available space, particularly beneficial when dealing with deep hierarchies
- **Intuitive Navigation**: Users can naturally follow the hierarchy from center to periphery
- **Proportional Representation**: Segment sizes accurately reflect the underlying data values
- **Interactive Capabilities**: Built-in zoom, hover, and drill-down functionality enhance user engagement
- **Multi-dimensional Analysis**: Support for color coding enables additional data dimensions to be displayed simultaneously

Basic Implementation Approaches

Using Plotly Express with Path Parameter

The most straightforward approach to creating sunburst charts involves using **Plotly Express** with the `path` parameter. This method is ideal for rectangular dataframes where different columns represent different levels of the hierarchy[3].

```
import plotly.express as px
import pandas as pd

# Create sample hierarchical data
data = {
    'region': ['North', 'North', 'North', 'North', 'South', 'South', 'South', 'South'],
    'department': ['Sales', 'Sales', 'Marketing', 'Marketing', 'Sales', 'Sales', 'Marketi
    'team': ['Team A', 'Team B', 'Team A', 'Team B', 'Team A', 'Team B', 'Team A', 'Team
    'employees': [25, 30, 20, 18, 35, 28, 22, 26]
}

df = pd.DataFrame(data)

# Create basic sunburst chart
fig = px.sunburst(
    df,
    path=['region', 'department', 'team'],
    values='employees',
    title='Company Structure - Multi-Level Sunburst'
)

fig.show()
```



Multi-level sunburst chart showing company structure by region, department, and team

This basic implementation demonstrates the fundamental structure of a three-level hierarchy, showing how regions branch into departments, which further subdivide into teams. The segment sizes accurately reflect the number of employees in each organizational unit.

Using Graph Objects for Advanced Control

For more sophisticated customization requirements, **Plotly Graph Objects** provides greater control over chart appearance and behavior[4]. This approach is particularly useful when working with explicit parent-child relationships or when advanced styling is required.

```
import plotly.graph_objects as go

# Create data with explicit parent-child structure
data = dict(
    labels=["Root", "Branch A", "Branch B", "A1", "A2", "B1", "B2", "B3"],
    parents=["", "Root", "Root", "Branch A", "Branch A", "Branch B", "Branch B", "Branch
    values=[100, 50, 50, 25, 25, 15, 15, 20]
)

# Create advanced sunburst chart
fig = go.Figure(go.Sunburst(
    labels=data['labels'],
    parents=data['parents'],
    values=data['values'],
    branchvalues="total",
    maxdepth=3,
    insidetextorientation='radial'
))

fig.update_layout(
    title="Advanced Sunburst Chart",
    margin=dict(t=50, l=0, r=0, b=0)
)

fig.show()
```

Advanced Features and Customization

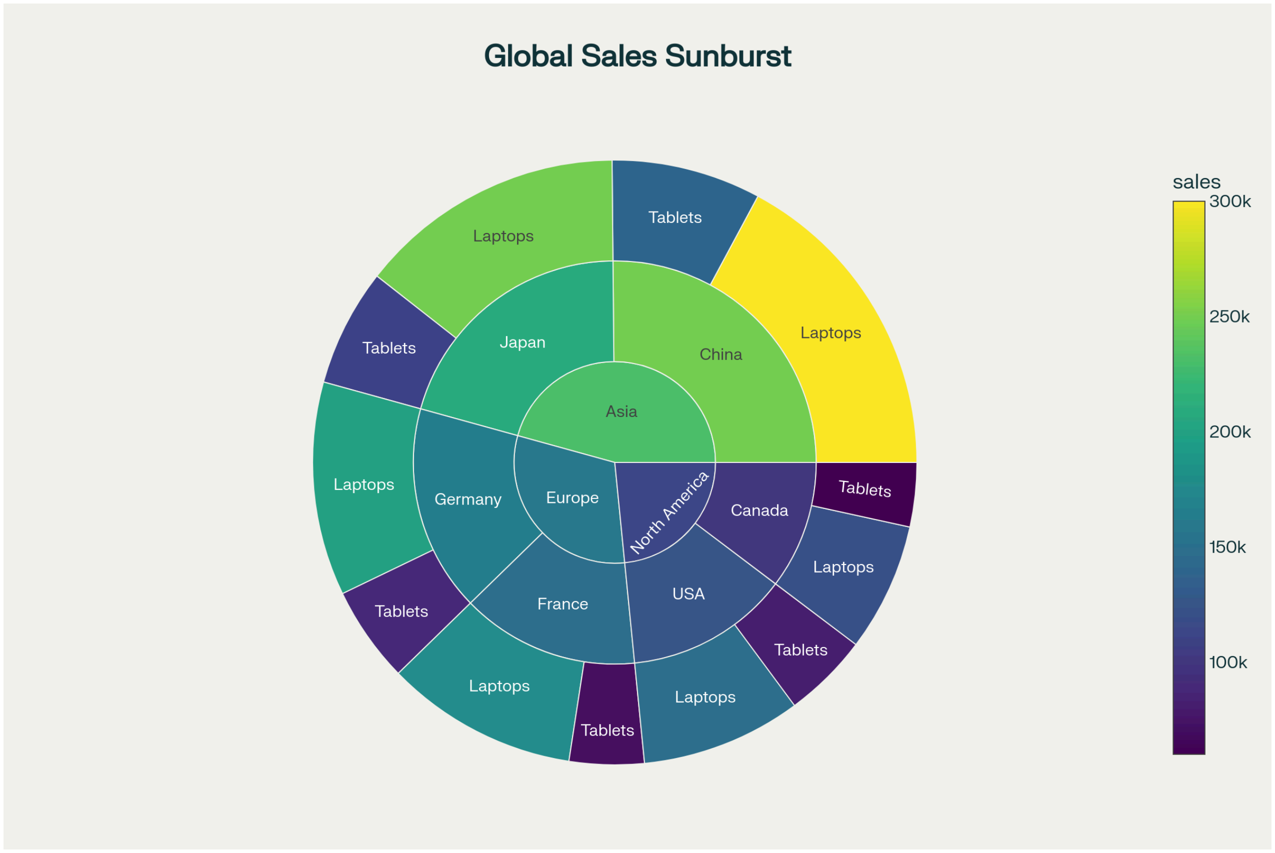
Color Mapping Strategies

Effective color mapping significantly enhances the analytical value of sunburst charts. Plotly supports both **continuous** and **discrete** color mapping approaches, each serving different analytical purposes.

Continuous Color Mapping

Continuous color mapping is ideal for representing numerical metrics such as sales performance, profit margins, or growth rates[1]. The color intensity provides immediate visual feedback about the relative performance of different segments.

```
import plotly.express as px
import pandas as pd
import numpy as np

# Create sales data with performance metrics
sales_data = {
    'continent': ['North America', 'North America', 'Europe', 'Europe', 'Asia', 'Asia'],
    'country': ['USA', 'Canada', 'Germany', 'France', 'Japan', 'China'],
    'product': ['Laptops', 'Tablets', 'Laptops', 'Tablets', 'Laptops', 'Tablets'],
    'sales': [150000, 80000, 200000, 90000, 250000, 110000]
}

df = pd.DataFrame(sales_data)

# Create sunburst with continuous color scale
fig = px.sunburst(
    df,
    path=['continent', 'country', 'product'],
    values='sales',
    color='sales',
    color_continuous_scale='RdBu',
    color_continuous_midpoint=np.average(df['sales']),
    title='Global Sales Performance'
)

fig.show()
```



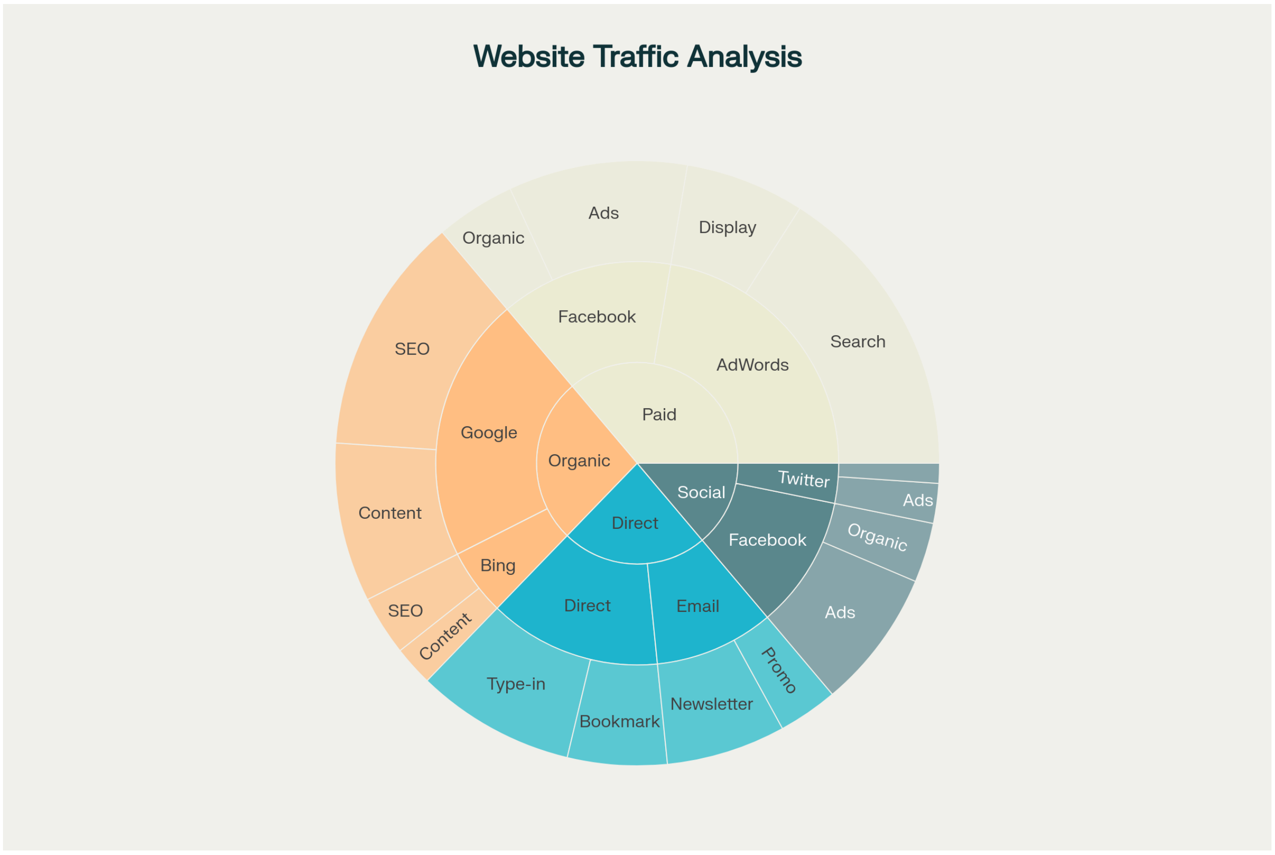
Multi-level sunburst chart showing global sales performance by continent, country, and product

Discrete Color Mapping

Discrete color mapping excels at distinguishing between categorical variables, making it easier to identify patterns and relationships within the hierarchy[1].

```python
import plotly.express as px
import pandas as pd

# Create website traffic data
traffic_data = {
    'source': ['Organic', 'Organic', 'Paid', 'Paid', 'Social', 'Social'],
    'channel': ['Google', 'Bing', 'AdWords', 'Facebook', 'Facebook', 'Twitter'],
    'campaign': ['SEO', 'Content', 'Search', 'Ads', 'Organic', 'Ads'],
    'visitors': [12000, 3000, 15000, 9000, 7000, 2000]
}

df = pd.DataFrame(traffic_data)

# Create sunburst with discrete colors
fig = px.sunburst(
    df,
    path=['source', 'channel', 'campaign'],
    values='visitors',
    color='source',
    color_discrete_sequence=px.colors.qualitative.Set3,
    title='Website Traffic Analysis'
)
```

```
fig.show()
```



Multi-level sunburst chart showing website traffic analysis by source, channel, and campaign

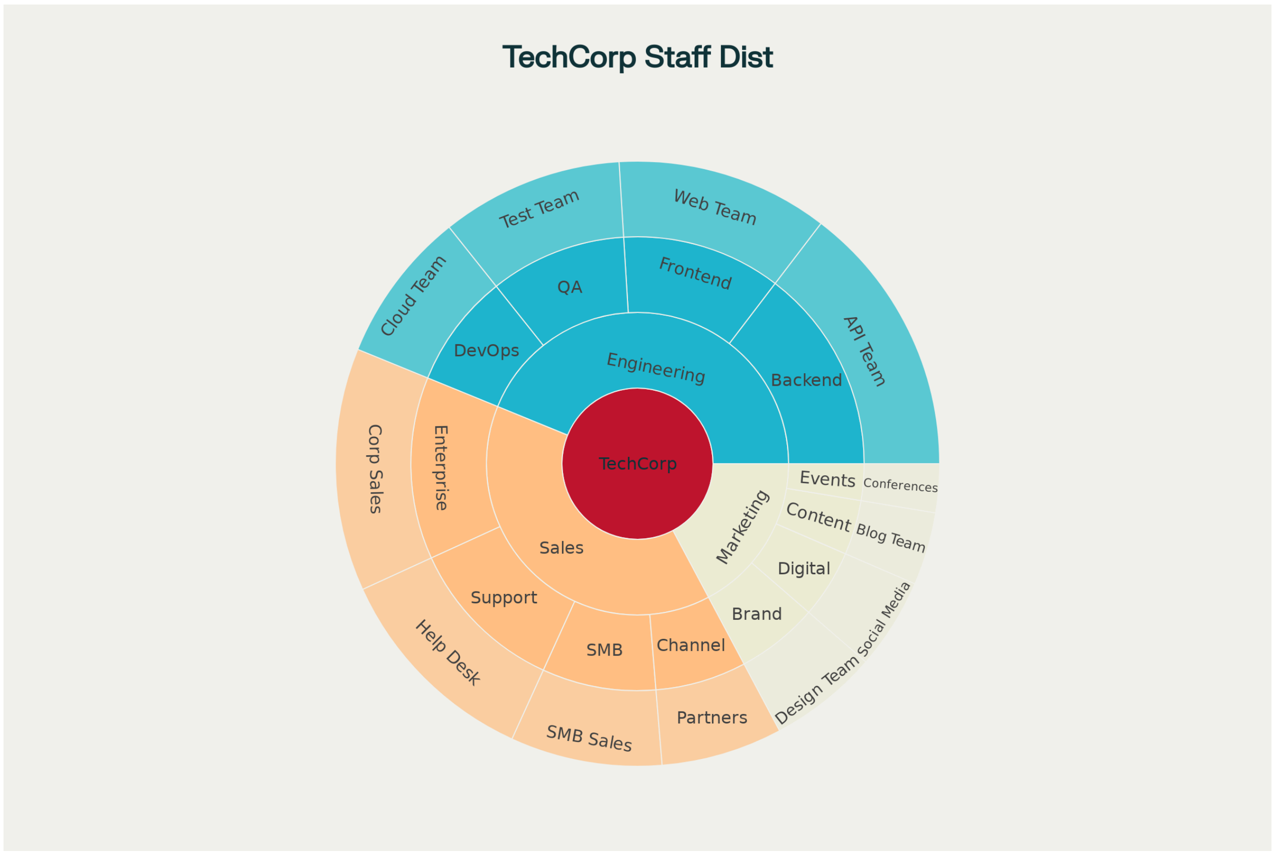
Enhanced Interactivity Features

Modern sunburst charts benefit significantly from enhanced interactivity features that improve user engagement and analytical capabilities[5]. These features include custom hover templates, drill-down functionality, and dynamic filtering.

Custom Hover Templates

Custom hover templates provide users with additional context without cluttering the main visualization. They can display multiple metrics, percentages, and contextual information that enhances the analytical value of the chart.

```
import plotly.express as px
import pandas as pd

# Create detailed product data
product_data = {
    'category': ['Electronics', 'Electronics', 'Electronics', 'Clothing', 'Clothing', 'Cl
    'subcategory': ['Phones', 'Laptops', 'Tablets', 'Mens', 'Womens', 'Kids'],
    'product': ['iPhone', 'MacBook', 'iPad', 'Shirts', 'Dresses', 'Toys'],
    'sales': [50000, 80000, 30000, 25000, 35000, 15000],
    'units_sold': [500, 200, 300, 1000, 700, 500],
    'profit_margin': [0.4, 0.3, 0.35, 0.6, 0.5, 0.45]
}

df = pd.DataFrame(product_data)

# Create sunburst with enhanced hover information
fig = px.sunburst(
    df,
    path=['category', 'subcategory', 'product'],
    values='sales',
    color='profit_margin',
    hover_data=['units_sold', 'profit_margin'],
    color_continuous_scale='Viridis',
    title='Product Sales Analysis with Enhanced Interactivity'
)

fig.update_traces(
    hovertemplate='<b>%{label}</b><br>' +
                  'Sales: $%{value:,.0f}<br>' +
                  'Profit Margin: %{color:.1%}<br>' +
                  '<extra></extra>'
)

fig.show()
```



Advanced multi-level sunburst chart showing company structure with custom styling and hover templates

Data Structure Requirements and Best Practices

Hierarchical Data Organization

Effective sunburst charts require **properly structured hierarchical data**[6]. The data structure should maintain clear parent-child relationships and logical grouping of elements. Two primary data organization approaches are commonly used:

Rectangular Data Format

For datasets where hierarchy levels are represented as separate columns, the rectangular format provides the most straightforward approach:

```
# Rectangular format example
data = {
    'level1': ['A', 'A', 'A', 'B', 'B', 'B'],
    'level2': ['A1', 'A1', 'A2', 'B1', 'B1', 'B2'],
    'level3': ['Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6'],
    'values': [10, 20, 15, 25, 30, 18]
}
```

Explicit Parent-Child Format

For complex hierarchies or when working with tree-like data structures, the explicit parent-child format offers more flexibility:

```
# Parent-child format example
data = {
    'labels': ['Root', 'A', 'B', 'A1', 'A2', 'B1', 'B2'],
    'parents': ['', 'Root', 'Root', 'A', 'A', 'B', 'B'],
    'values': [100, 50, 50, 25, 25, 20, 30]
}
```

Handling Missing Values and Incomplete Hierarchies

Real-world datasets often contain missing values or incomplete hierarchical structures[1]. Plotly handles these situations gracefully when missing values are represented as `None`:

```
import plotly.express as px
import pandas as pd

# Data with missing values
data = {
    'region': ['North', 'North', 'North', 'South', 'South', 'South'],
    'state': ['NY', 'CA', None, 'TX', 'FL', None],
    'city': ['NYC', 'LA', None, 'Austin', 'Miami', None],
    'population': [8000000, 4000000, 2000000, 1000000, 500000, 800000]
}
```

```
df = pd.DataFrame(data)

fig = px.sunburst(
    df,
    path=['region', 'state', 'city'],
    values='population',
    title='Population Data with Missing Values'
)

fig.show()
```

Performance Optimization and Scalability

Managing Large Datasets

When working with large hierarchical datasets, several optimization strategies can improve performance and readability:

Data Aggregation Techniques

For datasets with numerous small segments, aggregation can improve both performance and visual clarity:

```
# Aggregate small segments into "Others" category
def aggregate_small_segments(df, threshold=0.05):
    total = df['values'].sum()
    small_segments = df[df['values'] < total * threshold]
    large_segments = df[df['values'] >= total * threshold]

    if len(small_segments) > 0:
        others_row = {
            'category': 'Others',
            'values': small_segments['values'].sum()
        }
        return pd.concat([large_segments, pd.DataFrame([others_row])], ignore_index=True)

    return df
```

Controlling Display Depth

The `maxdepth` parameter allows control over the number of hierarchy levels displayed simultaneously, preventing visual clutter:

```
fig = px.sunburst(
    df,
    path=['level1', 'level2', 'level3', 'level4'],
    values='values',
    maxdepth=3,  # Show only first 3 levels
    title='Controlled Depth Sunburst'
)
```

Memory and Rendering Optimization

For applications requiring real-time updates or handling large datasets, consider these optimization approaches:

- **Lazy Loading**: Load hierarchy levels progressively based on user interaction
- **Data Sampling**: Use representative samples for initial visualization
- **Caching**: Cache computed layouts for frequently accessed hierarchies
- **Incremental Updates**: Update only changed segments rather than regenerating entire charts

Common Use Cases and Industry Applications

Business Intelligence and Analytics

Sunburst charts excel in business intelligence applications where understanding hierarchical relationships is crucial[7]:

Organizational Analysis

Companies use sunburst charts to visualize organizational structures, showing employee distribution across regions, departments, and teams. This application helps identify imbalances, planning resource allocation, and understanding organizational complexity.

Financial Performance Analysis

Financial analysts employ sunburst charts to break down revenue or expenses by product lines, geographic regions, and customer segments. The proportional representation makes it easy to identify the most significant contributors to overall performance.

Market Segmentation

Marketing teams utilize sunburst charts to analyze customer segments, showing how different demographic groups contribute to overall revenue or engagement metrics.

Web Analytics and Digital Marketing

Digital marketing professionals leverage sunburst charts for comprehensive traffic analysis[7]:

Traffic Source Analysis

Website analytics benefit significantly from sunburst visualization, showing the hierarchy of traffic sources, channels, and specific campaigns. This approach provides immediate insights into which marketing channels drive the most valuable traffic.

User Journey Mapping

Understanding user behavior through multi-level funnel analysis becomes more intuitive with sunburst charts, showing how users progress through different stages of the customer journey.

Scientific and Research Applications

Research institutions and scientific organizations use sunburst charts for various analytical purposes:

Taxonomic Classification

Biological and ecological researchers use sunburst charts to visualize species distribution across different taxonomic levels, from kingdom to species.

Experimental Data Analysis

Scientific experiments often generate hierarchical data that benefits from sunburst visualization, particularly when analyzing factor interactions across multiple levels.

Troubleshooting and Common Issues

Label Readability Problems

One of the most common challenges with sunburst charts involves label readability, particularly for smaller segments[8]:

Solutions for Label Overlap

```
# Adjust label orientation and font size
fig.update_traces(
    textfont_size=10,
    insidetextorientation='radial'
)

# Use uniformtext for consistent sizing
fig.update_layout(
    uniformtext=dict(minsize=8, mode='hide')
)
```

Color Mapping Issues

Color mapping problems often arise when dealing with mixed data types or unexpected value ranges:

Debugging Color Scales

```
# Verify data types
print(df['color_column'].dtype)

# Check for null values
print(df['color_column'].isnull().sum())

# Examine value distribution
print(df['color_column'].describe())
```

Performance and Memory Issues

Large datasets can cause performance problems or memory issues:

Diagnostic Approaches

```
# Monitor data size
print(f"Dataset size: {df.memory_usage().sum() / 1024**2:.2f} MB")

# Check hierarchy complexity
levels = df.groupby('level1').size().describe()
print(f"Hierarchy complexity: {levels}")

# Profile rendering time
import time
start_time = time.time()
fig.show()
end_time = time.time()
print(f"Rendering time: {end_time - start_time:.2f} seconds")
```

Best Practices and Design Guidelines

Visual Design Principles

Effective sunburst charts require careful attention to visual design principles[6]:

Color Scheme Selection

- **Use colorblind-friendly palettes** for accessibility
- **Maintain sufficient contrast** between adjacent segments
- **Apply consistent color logic** throughout the hierarchy
- **Limit color variations** to prevent visual confusion

Typography and Labeling

- **Keep labels concise** and descriptive
- **Use appropriate font sizes** for readability
- **Consider label rotation** for better space utilization
- **Provide clear legends** when necessary

Interaction Design

Well-designed interactions enhance user experience and analytical capabilities:

Hover Behavior

```
fig.update_traces(
    hovertemplate='<b>%{label}</b><br>' +
                  'Value: %{value}<br>' +
                  'Percentage: %{percentParent}<br>' +
                  '<extra></extra>'
)
```

Click Interactions

Advanced applications can implement custom click handlers for drill-down functionality or data filtering.

Accessibility Considerations

Ensuring accessibility improves usability for all users:

- **Provide alternative text descriptions** for screen readers
- **Use patterns or textures** in addition to colors
- **Ensure keyboard navigation** support
- **Provide data table alternatives** for complex hierarchies

Advanced Customization Techniques

Custom Styling and Theming

Advanced users can create custom themes and styling for consistent branding:

```
# Custom theme configuration
custom_theme = {
    'layout': {
        'font': {'family': 'Arial, sans-serif', 'size': 12},
        'colorway': ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728'],
        'paper_bgcolor': '#f8f9fa',
        'plot_bgcolor': '#ffffff'
    }
}

fig.update_layout(custom_theme['layout'])
```

Dynamic Data Updates

For dashboard applications, implementing dynamic data updates enhances user experience:

```
# Example of dynamic data update function
def update_sunburst_data(fig, new_data):
    fig.data[^0].labels = new_data['labels']
    fig.data[^0].parents = new_data['parents']
    fig.data[^0].values = new_data['values']
    return fig
```

Integration with Dashboard Frameworks

Sunburst charts integrate well with popular dashboard frameworks:

Dash Integration

```
import dash
from dash import dcc, html, Input, Output
import plotly.express as px

app = dash.Dash(__name__)

app.layout = html.Div([
    dcc.Graph(id='sunburst-chart'),
    dcc.Dropdown(
        id='filter-dropdown',
        options=[{'label': 'All', 'value': 'all'}],
        value='all'
    )
])

@app.callback(
    Output('sunburst-chart', 'figure'),
    Input('filter-dropdown', 'value')
)
def update_chart(selected_filter):
    # Filter data based on selection
    filtered_data = filter_data(selected_filter)

    # Create updated sunburst chart
    fig = px.sunburst(
        filtered_data,
        path=['level1', 'level2', 'level3'],
        values='values'
    )
    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

Conclusion

Multi-level sunburst charts represent a powerful and versatile tool for visualizing hierarchical data structures. Through Plotly's comprehensive API, developers and analysts can create sophisticated, interactive visualizations that effectively communicate complex data relationships. The combination of intuitive radial layouts, flexible customization options, and robust interactivity features makes sunburst charts particularly valuable for business intelligence, web analytics, and scientific research applications.

Success with sunburst charts requires careful attention to data structure, thoughtful design choices, and consideration of user experience principles. By following the best practices outlined in this guide and leveraging the advanced features available in Plotly, practitioners can create compelling visualizations that enhance understanding and drive better decision-making.

The continued evolution of web-based visualization technologies ensures that sunburst charts will remain relevant and continue to offer new possibilities for hierarchical data exploration. As datasets grow in complexity and size, the importance of effective hierarchical visualization techniques like sunburst charts will only increase, making them an essential skill for modern data professionals.

⁂

1. https://www.geeksforgeeks.org/python/sunburst-plot-using-plotly-in-python/
2. https://community.plotly.com/t/multiple-leaves-on-sunburst-chart/53546
3. https://coderzcolumn.com/tutorials/data-science/how-to-create-sunburst-chart-in-python-plotly
4. https://www.youtube.com/watch?v=RdaV9dvC6sc
5. https://deephaven.io/core/plotly/0.16.0/docs/sunburst/
6. https://plotly.com/python/pie-charts/
7. https://www.youtube.com/watch?v=9jiynC5Vjto
8. https://www.youtube.com/watch?v=6OMKTI7RKjQ