Introduction

This project explores the use of MATLAB and Tableau for data visualization. Three problems are addressed:
(1) Sales data visualization using a nested treemap in MATLAB
(2) A heatmap representation of mean temperatures over the Tri-State Area in MATLAB
(3) ATP match data analysis using Tableau

Problem 1: Sales by Branch (MATLAB)

Analysis

The goal of this problem is to visualize the "Sales by Branch" dataset using a **nested treemap**. A nested treemap is a hierarchical visualization that uses rectangles to represent data, where the size of each rectangle corresponds to a quantitative value (e.g., revenue) and the nesting represents categorical relationships (e.g., branches and product groups).

Dataset Structure

The dataset consists of:

- **Branches**: Categorical and nominal (e.g., Brighton & Hove, Chichester).
- **Product Groups**: Categorical and nominal (e.g., Books, Electronics, Hardware).
- **Total Revenue**: Quantitative and continuous.

The dataset has a **hierarchical structure**, where each branch contains multiple product groups, and each product group has an associated revenue value. This structure makes a nested treemap an ideal choice for visualization.

Why a Nested Treemap is Suitable

1. **Hierarchical Data Representation**:
   - The dataset has a clear hierarchy: **Branches → Product Groups → Revenue**. A nested treemap is specifically designed to visualize hierarchical data, where each level of the hierarchy is represented by nested rectangles.
   - **For example**, the top-level rectangles represent branches, and within each branch, smaller rectangles represent product groups. This nesting makes it easy to see the relationship between branches and their respective product groups.

2. **Proportional Representation of Revenue**:
   - The size of each rectangle in the treemap is proportional to the **total revenue** of the corresponding branch or product group. This allows for quick identification of which branches or product groups contribute the most to overall sales.
   - **For instance**, the large rectangle for "Brighton & Hove" immediately indicates that this branch has the highest total revenue, while the smaller rectangles within it show the contribution of each product group (e.g., Software, Hardware).

3. **Space Efficiency**:
   - A nested treemap is highly space-efficient, as it uses the entire plot area to display all categories and subcategories. This is particularly useful for datasets with multiple branches and product groups, as it avoids clutter and ensures that all data points are visible.

4. **Cumulative Insights**:
    - The treemap provides **cumulative insights** by showing both the total revenue for each branch and the breakdown of revenue by product group within each branch. This makes it easy to compare the performance of different branches and identify which product groups drive sales in each branch.

5. **Visual Clarity and Intuitiveness**:
    - The use of color and size in a treemap makes it visually intuitive. Viewers can quickly grasp the relative importance of each branch and product group without needing to interpret complex axes or labels.
    - **For example**, the treemap clearly highlights that "Software" is the top-performing product group in "Brighton & Hove," while "DIY" is the top-performing product group in "Chichester."

How the Nested Treemap Visualizes the Data

1. **Top-Level Rectangles (Branches)**:
    - Each branch (e.g., Brighton & Hove, Chichester) is represented by a large rectangle. The size of the rectangle corresponds to the **total revenue** of the branch.
    - **For example**, "Brighton & Hove" has the largest rectangle because it has the highest total revenue across all branches.

2. **Nested Rectangles (Product Groups)**:
    - Within each branch, smaller rectangles represent the **product groups** (e.g., Books, Electronics, Hardware). The size of each nested rectangle corresponds to the revenue of the product group.
    - **For example**, within "Brighton & Hove," the "Software" rectangle is the largest, indicating that it generates the most revenue for this branch.

3. **Color Encoding**:
    - Colors can be used to differentiate between branches or product groups, making it easier to distinguish between categories at a glance.
    - **For example**, each branch could be assigned a unique color, with shades of that color used for its product groups.

4. **Labels**:
    - Labels are added to each rectangle to identify the branch and product group. This ensures that the visualization is informative and easy to interpret.

Implementation Steps

1. **Prepared the Dataset**:
    - Entered "Sales by Branch" data into a MATLAB cell array with columns: Branch, Product Group, and Total Revenue.
2. **Checked for Missing/Inconsistent Values**:
    - Verified all rows for valid entries (no missing or negative values).

3. **Extracted Values and Labels**:
   - Extracted **Total Revenue** into a numeric array (values).
   - Created labels by combining **Branch** and **Product Group** for each row.
4. **Normalized Revenue Values**:
   - Divided each revenue value by the sum of all revenues to ensure proportional representation.
5. **Generated Random Colors**:
   - Created random RGB colors for each rectangle using the rand function.
6. **Calculated Rectangle Positions and Sizes**:
   - Used the treemap function to recursively divide the plot area into proportional rectangles.
7. **Plotted the Rectangles**:
   - Drew rectangles using the plotRectangles function.
   - Added labels and applied colors to each rectangle.
8. **Added Outlines**:
   - Drew borders around each rectangle using the outline function for better readability.
9. **Formatted the Treemap**:
   - Added a title: "Nested Treemap: Branch vs Product Group".
   - Adjusted colors and layout for clarity.

Script: *Problem_1.m*

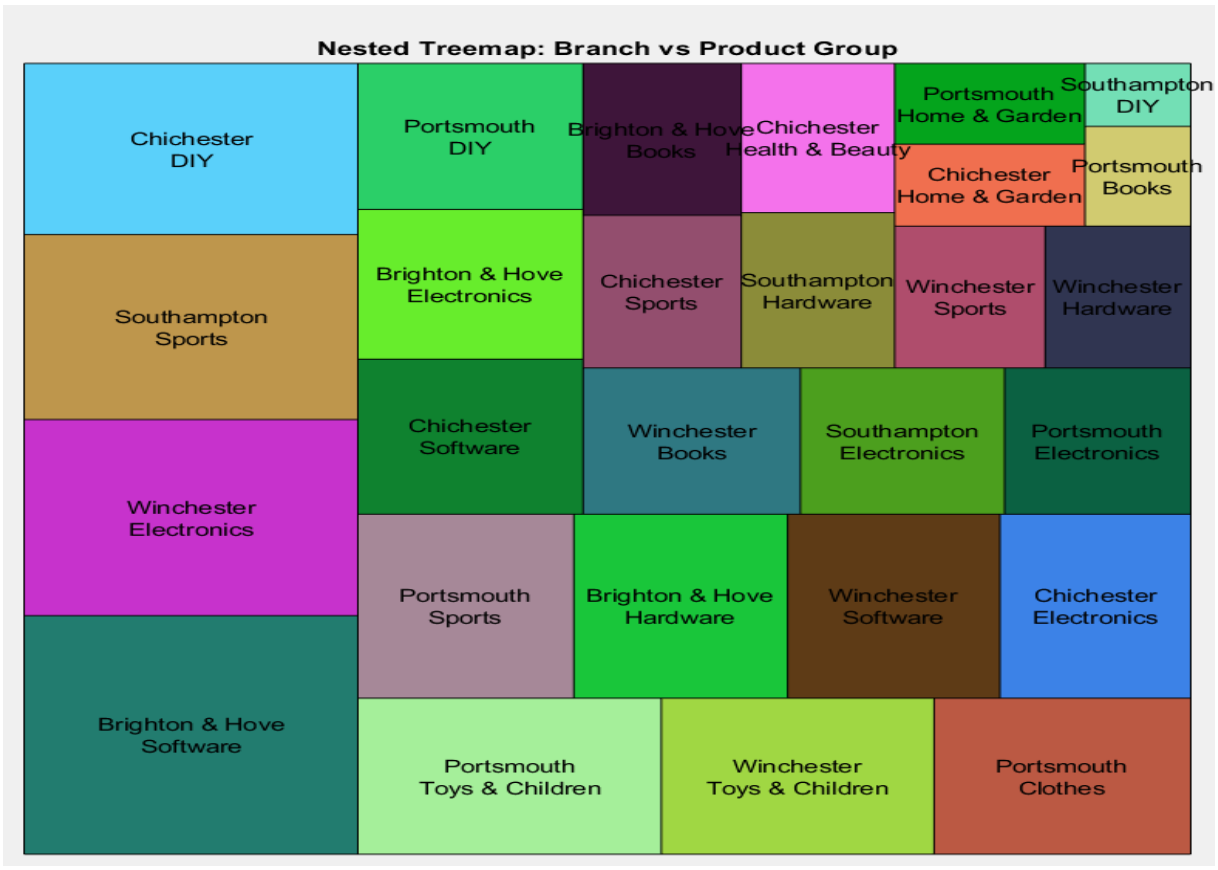
```
close all;
clear all;
clc;

data = {
    'Brighton & Hove', 'Books', 610.00;
    'Brighton & Hove', 'Electronics', 855.00;
    'Brighton & Hove', 'Hardware', 998.00;
    'Brighton & Hove', 'Software', 2020.00;
    'Chichester', 'Home & Garden', 396.00;
    'Chichester', 'Health & Beauty', 580.00;
    'Chichester', 'Sports', 613.00;
    'Chichester', 'Software', 885.00;
    'Chichester', 'Electronics', 891.00;
    'Chichester', 'DIY', 1449.00;
    'Portsmouth', 'Books', 268.00;
    'Portsmouth', 'Home & Garden', 390.00;
    'Portsmouth', 'Electronics', 690.00;
    'Portsmouth', 'DIY', 834.00;
    'Portsmouth', 'Sports', 1008.00;
    'Portsmouth', 'Clothes', 1016.00;
    'Portsmouth', 'Toys & Children', 1201.00;
    'Southampton', 'DIY', 169.00;
    'Southampton', 'Hardware', 604.00;
    'Southampton', 'Electronics', 757.00;
    'Southampton', 'Sports', 1567.00;
    'Winchester', 'Hardware', 524.00;
    'Winchester', 'Sports', 541.00;
    'Winchester', 'Books', 806.00;
    'Winchester', 'Software', 991.00;
    'Winchester', 'Toys & Children', 1079.00;
    'Winchester', 'Electronics', 1661.00;
};
```

```
values = [data{:, 3}];
labels = arrayfun(@(i) sprintf('%s\n%s', data{i, 1}, data{i, 2}), 1:size(data,
1), 'UniformOutput', false)';

values = values / sum(values);

colors = rand(length(values), 3);

rectangles = treemap(values);

plotRectangles(rectangles, labels, colors);

outline(rectangles);

title('Nested Treemap: Branch vs Product Group');
```

Script Explanation

- Ensures a clean environment by closing all figures, clearing all variables, and clearing the command window.
- Stores the sales data in a structured format with columns for **Branch**, **Product Group**, and **Total Revenue**.
- Extracts the revenue values into a numeric array.
- Combines the branch and product group into a single label for each data point.
- Divides each revenue value by the sum of all revenues to ensure proportional representation in the treemap.
- Creates a random RGB color for each rectangle to distinguish between different branches and product groups.
- Uses a recursive algorithm to calculate the positions and sizes of the rectangles based on the normalized revenue values.
- Draws each rectangle on the plot, applies the corresponding color, and adds labels to identify the branch and product group.
- Draws borders around each rectangle to improve readability and clearly separate adjacent rectangles.
- Includes a title to describe the purpose of the visualization: "Nested Treemap: Branch vs Product Group".

Results



Challenges & Solutions

Issue 1: Label Overlapping and Readability

- When plotting the treemap, labels for smaller rectangles overlapped, making it difficult to read the Branch and Product Group names.
- Some color choices made it hard to distinguish between adjacent rectangles.

Solution:

- Adjusted label placement by centering text within each rectangle and reducing font size for smaller rectangles.
- Used contrasting colors for adjacent rectangles to improve visual distinction.
- Added a legend to map colors to Branches for better clarity.

Issue 2: Normalization of Revenue Values

- Initially, the rectangles in the treemap were not proportional to the revenue values, leading to incorrect visual representation.
- Some rectangles were too small or too large compared to their actual revenue contribution.

Solution:

- Normalized the revenue values by dividing each value by the sum of all revenues.
- Ensured that the total area of the treemap was 1 (or 100%), making the rectangles proportional to the revenue contributions.

Problem 2: Electric Vehicle Sales Data (MATLAB)

Analysis

The goal of Problem 2 was to create a **heatmap** of the mean temperature for each month in the Greater Cincinnati Tri-State Area (Ohio, Kentucky, and Indiana). The heatmap needed to be overlaid on a map of the region to provide geographical context.

Implementation Steps

1. **Data Preparation**:
   - The temperature data was stored in an Excel file (area.xlsx), with cities as rows and monthly temperatures as columns.
   - The data was imported into MATLAB using the readtable function.

2. **Map Preparation**:
   - A map of the Tri-State Area was created using the shaperead function to load US state boundaries.
   - The map was customized to focus on the Greater Cincinnati region by setting latitude and longitude limits.

3. **Heatmap Creation**:
   - The mean temperature for each city was calculated by averaging the monthly temperatures.
   - A scatter plot was overlaid on the map, with the color of each point representing the mean temperature.
   - A color bar was added to provide a reference for the temperature values.

4. **Visual Enhancements**:
   - State names (Ohio, Kentucky, Indiana) were added to the map for better geographical context.
   - The heatmap was titled "Mean Temperature Heatmap (°F)" for clarity.

Script: *Problem_2.m*

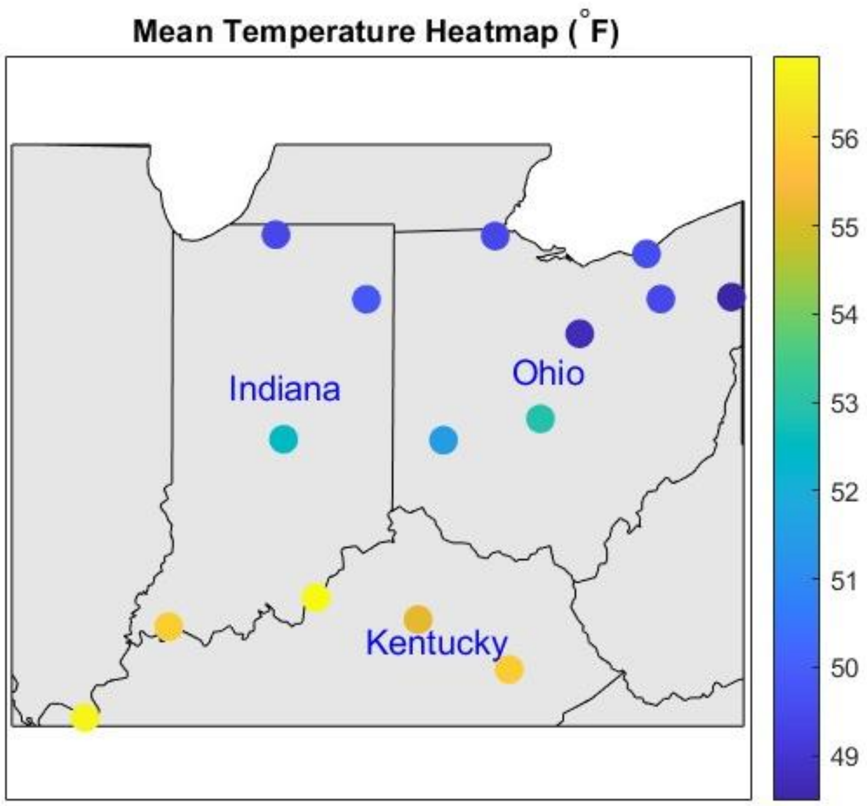
```
data = readtable('area.xlsx');


cities = data.CITIES;
tempData = data{:, 2:end};


lat = [41.0814, 41.4993, 39.9612, 39.7589, 40.7584, 41.6639, 41.0998, ...
       37.5556, 38.0406, 38.2527, 37.0834, 37.9716, 41.0793, 39.7684, 41.6764];
% Latitude
lon = [-81.5190, -81.6944, -82.9988, -84.1916, -82.5154, -83.5552, -80.6495,
...
       -83.3839, -84.5037, -85.7585, -88.6001, -87.5711, -85.1394, -86.1581, -
86.2520]; % Longitude
```

```
latLim = [37.0, 42.5];
lonLim = [-89.5, -80.5];

figure;
axesm('mercator', 'MapLatLimit', latLim, 'MapLonLimit', lonLim);
states = shaperead('usastatelo', 'UseGeoCoords', true);
geoshow(states, 'FaceColor', [0.9 0.9 0.9], 'EdgeColor', 'black');
title('Greater Cincinnati Tri-State Area');

stateNames = {'Ohio', 'Kentucky', 'Indiana'};
stateCentroids = [
    40.4173, -82.9071;
    37.8393, -84.2700;
    40.2672, -86.1349;
];

for i = 1:length(stateNames)
    textm(stateCentroids(i, 1), stateCentroids(i, 2), stateNames{i}, ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle', ...
        'FontSize', 12, ...
        'Color', 'blue');
end


hold on;
scatterm(lat, lon, 100, mean(tempData, 2), 'filled');
colorbar;
title('Mean Temperature Heatmap (^{\circ}F)');
hold off;
```

Script Explanation

- Imports the temperature data from the file area.xlsx using readtable().
- Extracts the city names and temperature data for each month.
- Specifies the latitude and longitude coordinates for each city in the dataset.
- Defines the latitude and longitude limits (latLim and lonLim) to focus on the Greater Cincinnati Tri-State Area.
- Uses the axesm() function to create a map with the Mercator projection.
- Loads US state boundaries using shaperead() and displays them on the map with geoshow().
- Adds a title to the map: "Greater Cincinnati Tri-State Area".
- Defines the names of the states (Ohio, Kentucky, Indiana) and their centroid coordinates.
- Uses textm() to display the state names at their respective centroids.
- Uses scatterm() to plot the cities on the map, with the color of each point representing the mean temperature for that city.
- Adds a color bar to provide a reference for the temperature values.
- Adds a title to the heatmap: "Mean Temperature Heatmap (°F)".
- Uses hold on and hold off to overlay the heatmap on the map.
- Ensures the visualization is clear and easy to interpret.

Results



Challenges & Solutions

Issue 1: Data Import and Formatting

- The temperature data was stored in an Excel file, and importing it into MATLAB required proper formatting.
- Some cities had missing or inconsistent temperature values.

Solution:

- Used the readtable function to import the data and ensured that all missing values were handled.
- Verified the data for consistency before proceeding with the analysis.

Issue 2: Map Customization

The default map included the entire US, making it difficult to focus on the Tri-State Area.

Solution:

- Set custom latitude and longitude limits using MapLatLimit and MapLonLimit properties.
- Added state names to the map for better geographical context.

Issue 3: Heatmap Visualization

- The scatter plot points were too small or too large, making it difficult to interpret the heatmap.

Solution:

- Adjusted the size of the scatter plot points to ensure they were clearly visible.
- Added a color bar to provide a reference for the temperature values.

Problem 3: Visualizations (TABLEAU)

1. Analysis of the Visualization: "Top 10 Countries with Most Wins"

How This Visualization Was Created in Tableau:

1. **Data Source:** The dataset used is atp_matches_2024.csv, which contains ATP tennis match data.
2. **Selecting Relevant Columns:** The key column used here is Winner_ioc (Winner's Country Code) and Winner_name.
3. **Measure Used:** Count of Winner_name (Number of matches won).
4. **Creating the Visualization:**

    - Drag Winner_ioc to the **Rows** shelf.
    - Drag Winner_name to **Columns** and change it to **COUNT(Winner_name)** to aggregate the number of wins.
    - Use the **Filter** panel to show only the **Top 10** countries based on the number of wins.
    - Change the visualization type to **Treemap** (from "Show Me" panel).

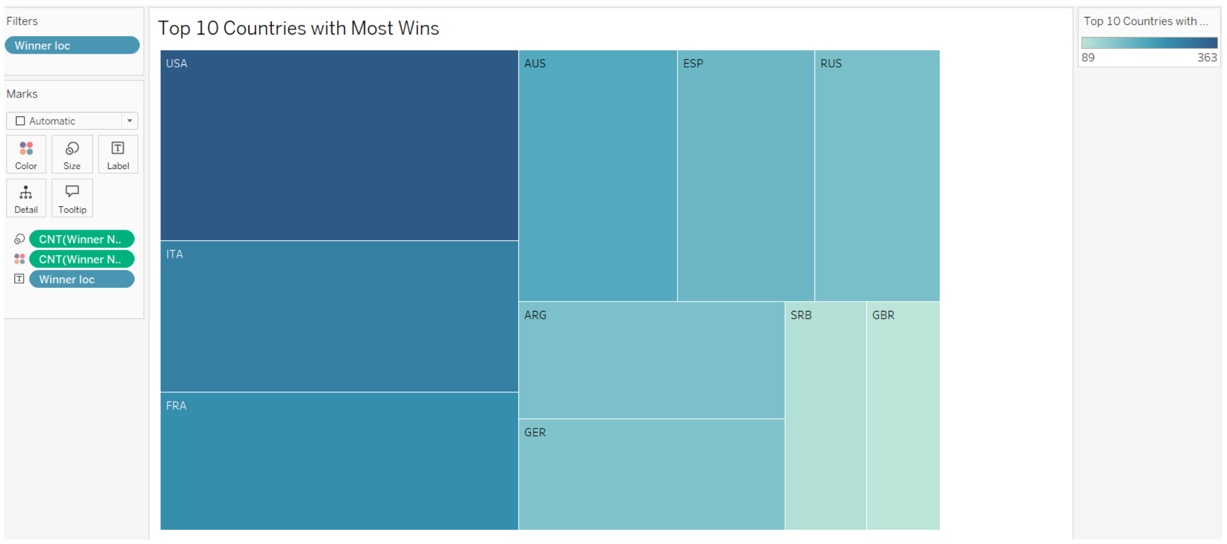
5. **Formatting Adjustments:**

    - Added **color gradient** based on the number of wins.
    - Adjusted labels to display country codes directly on the treemap.
    - Added a meaningful title: **"Top 10 Countries with Most Wins"**.

Observations & Insights:

- **USA dominates** in the number of wins, indicating a strong performance by American players.
- **Australia (AUS) and Spain (ESP)** also have a significant number of wins, showing their consistent competitiveness in ATP matches.
- Countries like **Russia (RUS), Italy (ITA), and France (FRA)** are close in rankings, suggesting strong tennis talent from these regions.
- The **size of the blocks represents the dominance** of certain nations in terms of match victories.

Output:



Challenges & Solutions

Issue : Labeling & Readability

- Smaller country blocks in the treemap had overlapping or unreadable labels.

Solution:

- Adjust the **font size** or enable **tooltips** to display information on hover.
- Use **color contrast** to improve visibility.

2. Analysis of the Visualization: "Pie Chart for Distribution of Matches by Tournament Name"

1. Data Used

- The dataset **"atp_matches_2024.csv"** contains match data for ATP tournaments.
- The **Tourney Name** field represents different tennis tournaments.
- The **Match Number (Count)** field is used to determine the number of matches played in each tournament.

2. Steps Taken to Create the Pie Chart

1. **Load Data into Tableau**

    - Connected to **atp_matches_2024.csv** in Tableau.

- Checked the **Tourney Name** and **Match Number** fields to ensure they were properly interpreted.

2. **Creating the Pie Chart**

   - Dragged **Tourney Name** to the **Color** shelf to differentiate tournaments.
   - Dragged **Match Number (CNT)** to the **Angle** shelf to determine the slice size.
   - Changed the **Marks** type to **Pie Chart** for a proportional visual representation.

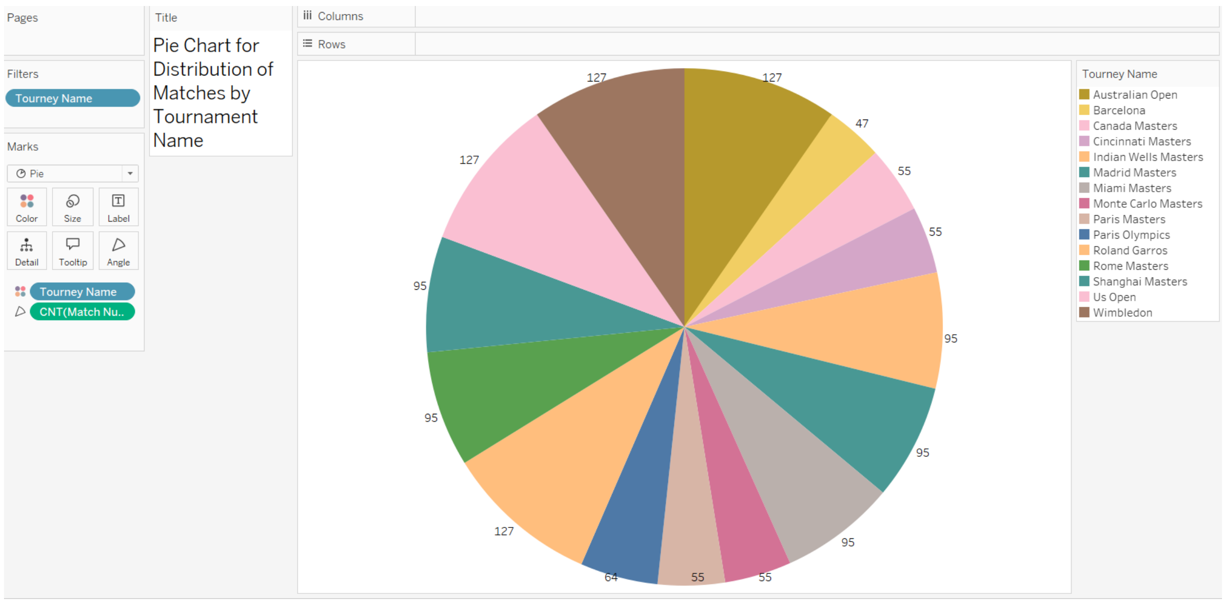
3. **Enhancing the Chart**

   - Applied distinct **colors** to each tournament for better differentiation.
   - Added **labels** showing the match count for each tournament.
   - Positioned the **legend** to the right for clarity.
   - Ensured proper **title formatting**: *"Pie Chart for Distribution of Matches by Tournament Name"*.

3. Insights from the Visualization

- Larger slices (e.g., Wimbledon, US Open, and Monte Carlo Masters) indicate tournaments with a higher number of matches.
- Smaller slices (e.g., Paris Olympics, Shanghai Masters) suggest fewer matches were played.
- The pie chart clearly shows the distribution of matches across tournaments, helping to identify which events had the most/least games.

Output:


Pages

Filters: Tourney Name

Marks: Pie

Color: Tourney Name

Angle: CNT(Match Nu..

Title: Pie Chart for Distribution of Matches by Tournament Name

Challenges & Solutions

Issue : Data Cleaning & Preparation

- Some tournaments had missing or inconsistent names, leading to incorrect match counts.

Solution:

- Used **data cleaning techniques** in Tableau and ensured **consistent naming conventions** by standardizing tournament names before visualization.

3. **Analysis of the Visualization: "Player's Age vs Performance in Different Tournaments"**

How This Visualization Was Created in Tableau

1. Data Source:

The dataset used is **atp_matches_2024.csv**, which contains ATP tennis match data.

2. Selecting Relevant Columns:

The key columns used in this visualization are:

- **Dimensions**: tourney_name, winner_name, winner_age
- **Measures**: w_ace, w_1stWon, w_df

3. Creating the Visualization:

1. **Drag tourney_name to Columns** – This will create separate bars for each tournament on the X-axis.
2. **Drag winner_name to Rows** – This will differentiate players and display their performance per tournament.
3. **Drag winner_age to the Color Shelf** – This helps color-code bars based on the player's age, allowing us to see patterns in performance by age.
4. **Drag w_ace, w_1stWon, and w_df to Columns** – Tableau will automatically aggregate them (e.g., SUM). You can adjust the aggregation (e.g., AVG) if needed.
5. **Change the Chart Type to Grouped Bar Chart** –

    - Go to the **"Show Me"** panel and select **Grouped Bar Chart**.
    - This will create a comparison of performance metrics for each player in different tournaments.

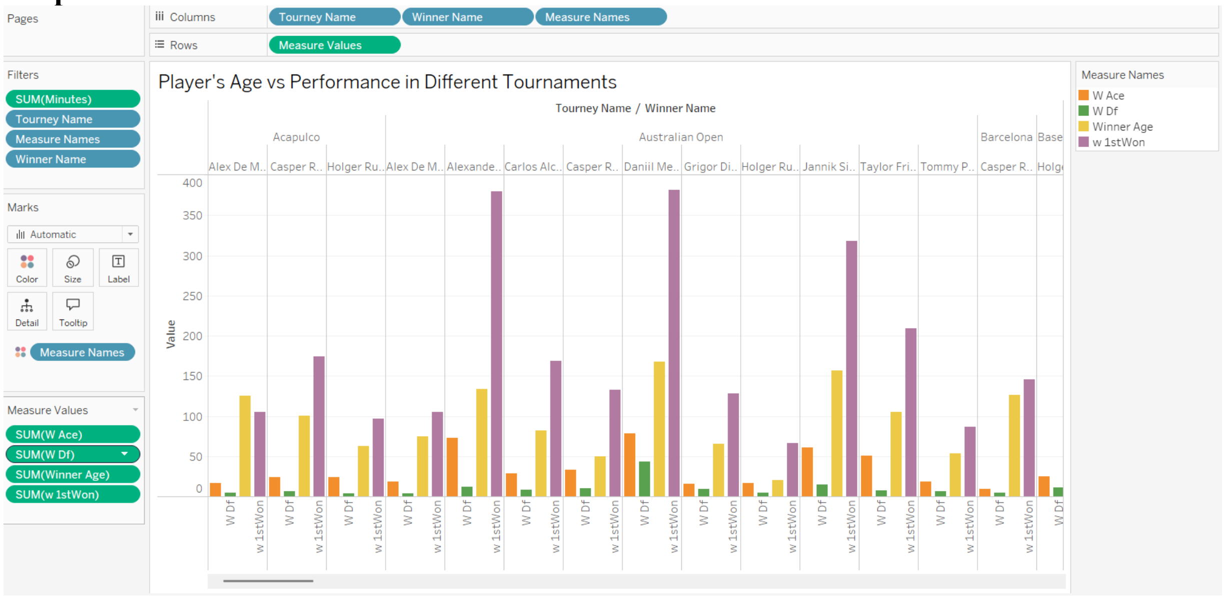
6. **Ensure the Grouping is Correct** –

    - The bars should be grouped by tournament and color-coded by **winner_age** to highlight age-based trends.

Observations & Insights:

- **Younger players (e.g., under 25) may have a higher number of aces** but also **more double faults**, indicating aggressive but risky serves.
- **Older players (30+) may have a higher first-serve win percentage**, suggesting more consistent and strategic play.
- **Certain tournaments may favor specific playing styles**, influencing serve performance across different age groups.
- If a player's **performance improves with age**, it could indicate experience and adaptability play a bigger role than raw power.

Output:



Columns: Tourney Name | Winner Name | Measure Names

Rows: Measure Values

Filters: SUM(Minutes), Tourney Name, Measure Names, Winner Name

Marks: Automatic; Measure Names

Measure Values: SUM(W Ace), SUM(W Df), SUM(Winner Age), SUM(w 1stWon)

Challenges & Solutions

Issue: Overlapping Bars and Cluttered Visualization

- When there are too many tournaments and players, bars may overlap, making the chart difficult to read.
- If age-based color coding is applied, it may create too many shades, making interpretation harder.

Solution:

- Use a **Top N filter** to show only the **Top 10 players or tournaments** based on w_1stWon or w_ace.
- Allow users to interact with filters to select specific players or age groups.