**Data Visualization: Unlocking insights of Data**

[October 23, 2023](https://codanics.com/data-visualization-in-data-science/)[Dr. Aammar Tufail](https://codanics.com/author/codanics/)

In today’s data-driven world, we are overwhelmed with vast amounts of information. To make sense of this deluge, *Data Visualization* has emerged as a critical tool. It transforms complex datasets into intuitive visuals, enabling better comprehension, decision-making, and insights. Dive in with us as we explore the power and potential of visualizing data!

In this post,

* First we will learn what is Data Visualization and why do we need to be an expert in data visualization in order to jump into the field of Data Science/Analytic?
* Secondly, we will learn about popular data visualization tools
* Thirdly, we will see about the types of plots
* Lastly, we will learn how to make those plots on dummy dataset using [python](https://codanics.com/data-visualization-in-data-science/) libraries (pandas, matplotlib, seaborn, [plotly](https://codanics.com/data-visualization-in-data-science/), etc.).

Let’s have a look into it:

What is Data Visualization?

Data Visualization is the representation of data in a graphical format. It allows users to see and understand patterns, trends, and insights in data. From simple bar charts to intricate 3D models, data visualization encompasses a wide range of techniques.

Why is Data Visualization Important?

1. **Quick Insights**: A well-crafted visual can communicate complex data in seconds, saving time and effort.
2. **Informed Decision Making**: Visual data aids in making evidence-based decisions, reducing risks.
3. **Engagement**: Visuals are more engaging than spreadsheets, making it easier to present and share findings.
4. **Data Exploration**: Visualization tools enable users to delve deep into data, uncovering hidden insights.

Key Principles of Effective Data Visualization:

1. **Simplicity**: Less is more. Avoid clutter and focus on the essentials.
2. **Consistency**: Use consistent colors, fonts, and symbols to avoid confusion.
3. **Accuracy**: Ensure the visual accurately represents the data.
4. **Interactivity**: Modern tools allow for interactive visuals, enhancing user engagement and understanding.

Popular Data Visualization Tools:

1. **Tableau**: A powerful tool offering a wide range of visualization options.
2. **Power BI**: Microsoft’s solution for data visualization and business intelligence.
3. **D3.js**: A JavaScript library for creating custom, dynamic visuals.
4. [**Python**](https://codanics.com/data-visualization-in-data-science/)**Libraries**: Libraries like Matplotlib, Seaborn, and [Plotly](https://codanics.com/data-visualization-in-data-science/) are popular among data scientists.
5. **R Programming:** ggplot2, [plotly](https://codanics.com/data-visualization-in-data-science/)

Data Visualization in Action:

Imagine you’re a retailer looking to understand sales trends. A spreadsheet with thousands of rows of sales data might be overwhelming. However, a heat map showing sales density or a line chart depicting monthly sales trends can provide clear insights in moments.

For our readers, here’s a glimpse of the kind of visuals we’re talking about:

**Different Plots-1**

**Geographical Maps**

**Bar plots**

**Dashboard**

**Different Plots-1**

**Geographical Maps**

**Bar plots**

**Dashboard**

**Different Plots-1**

**Geographical Maps**

**Bar plots**

**Dashboard**

Previous slide

Next slide

The images above perfectly encapsulate the essence of data visualization. From analyzing pie charts to understanding sales trends through heat maps and line charts, these visuals demonstrate the power and versatility of data visualization tools. Moreover, the intricacy of a comprehensive dashboard showcases how various visualizations can come together to provide a holistic view of data.

Incorporating such visuals into your content not only enhances the reader’s experience but also solidifies their understanding of the topic. As the saying goes, “A picture is worth a thousand words,” and in the realm of data, a well-crafted visualization might be worth a million data points!

Harness the potential of data visualization, and let your data come alive. Happy visualizing!

Key Terms used in Data Visualization

Here’s a list of key terms in data visualization, along with explanations and examples for each:

1. **Bar Chart**:
   * **Explanation**: A graphical representation of data using bars of varying heights or lengths.
   * **Example**: Comparing the sales of different products in a month.
2. **Pie Chart**:
   * **Explanation**: A circular chart divided into slices to illustrate numerical proportions.
   * **Example**: Showing the market share of various smartphone brands.
3. **Histogram**:
   * **Explanation**: A representation of the distribution of a dataset, similar to a bar chart but for frequency distribution.
   * **Example**: Displaying the age distribution of employees in a company.
4. **Scatter Plot**:
   * **Explanation**: A graph with points plotted to show the relationship between two sets of data.
   * **Example**: Comparing advertising spend with sales revenue.
5. **Line Chart**:
   * **Explanation**: A chart that displays information as a series of data points connected by straight line segments.
   * **Example**: Tracking stock market prices over a week.
6. **Heat Map**:
   * **Explanation**: A data visualization technique where values in a matrix are represented as colors.
   * **Example**: Showing website activity, where darker colors represent more clicks.
7. **Box Plot (or Whisker Plot)**:
   * **Explanation**: A standardized way of displaying the dataset based on a five-number summary: minimum, first quartile, median, third quartile, and maximum.
   * **Example**: Comparing exam scores of different classes.
8. **Area Chart**:
   * **Explanation**: Similar to a line chart, but the area between the axis and the line is filled with color or shading.
   * **Example**: Displaying the total revenue of a company over several years.
9. **Radar (or Spider) Chart**:
   * **Explanation**: A chart that displays multivariate data on axes starting from the same point.
   * **Example**: Comparing the features of different products.
10. **Treemap**:

* **Explanation**: A visualization of hierarchical data using nested rectangles.
* **Example**: Showing storage used by different types of files on a computer.

1. **Geographical Map**:

* **Explanation**: A map that displays data based on geographical areas or locations.
* **Example**: Highlighting areas with high crime rates in a city.

1. **Time Series**:

* **Explanation**: A sequence of data points indexed in time order.
* **Example**: Analyzing the daily temperature of a place over a year.

There are several other plots, such as bubble chart, violin plots and much more.

Plotting in Python

We can plot in [python](https://codanics.com/data-visualization-in-data-science/) using libraries such as [pandas](https://pandas.pydata.org/docs/user_guide/visualization.html), [matplotlib](https://matplotlib.org/), [seaborn](https://seaborn.pydata.org/), [plotly](https://plotly.com/python/), [altair](https://altair-viz.github.io/), [bokeh](https://bokeh.org/)

**1. Bar Chart**

# import pandas and matplotlib.pyplot libraries

import pandas as pd

import matplotlib.pyplot as plt

# create a dictionary with sample data

data = {'Products': ['Product A', 'Product B', 'Product C'],

'Sales': [100, 150, 80]}

# create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# plot the dataframe as a bar chart

df.plot(x='Products', y='Sales', kind='bar', legend=False)

# set the y-axis label

plt.ylabel('Sales')

# set the title of the plot

plt.title('Sales of different products')

# save the plot to a file

plt.savefig('bar\_chart.png', dpi=300, bbox\_inches='tight')

# display the plot

plt.show()

Copy

**2. Pie Chart**

# import pandas and matplotlib.pyplot libraries

import pandas as pd

import matplotlib.pyplot as plt

# create a dictionary with sample data

data = {'Products': ['Product A', 'Product B', 'Product C'],

'Sales': [100, 150, 80]}

# create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# plot the dataframe as a pie chart

df.set\_index('Products')['Sales'].plot(kind='pie', autopct='%1.1f%%', startangle=90)

# set the title of the plot

plt.title('Sales distribution of different products')

# save the plot to a file

plt.savefig('pie\_chart.png', dpi=300, bbox\_inches='tight')

# display the plot

plt.ylabel('') # This is to remove the 'Sales' ylabel which is unnecessary in a pie chart

plt.show()

Copy

**3. Histogram**

# import pandas and matplotlib.pyplot libraries

import pandas as pd

import matplotlib.pyplot as plt

# create a dictionary with sample data

data = {'Sales': [30,50,120,150,200,300,100, 150, 80, 120, 140, 130,

50,50,30,20,10,50,35,45,55,50,55,65,56,58,58,54,55,50,5,

110, 145, 105, 90, 115, 125, 135, 85, 95]}

# create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# plot the dataframe as a histogram

df['Sales'].plot(kind='hist', bins=5, edgecolor='black')

# set the x-axis and y-axis labels

plt.xlabel('Sales Value')

plt.ylabel('Number of Transactions')

# set the title of the plot

plt.title('Distribution of Sales Transactions')

# save the plot to a file

plt.savefig('histogram.png', dpi=300, bbox\_inches='tight')

# display the plot

plt.show()

Copy

**4. Scatter plot**

# import pandas and matplotlib.pyplot libraries

import pandas as pd

import matplotlib.pyplot as plt

# create a dictionary with sample data

data = {

'Sales': [100, 150, 80, 120, 140, 130, 110, 145, 105, 90],

'Profit': [50, 70, 40, 60, 75, 65, 55, 72, 52, 45]

}

# create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# plot the dataframe as a scatter plot

df.plot(x='Sales', y='Profit', kind='scatter')

# set the x-axis and y-axis labels

plt.xlabel('Sales Value')

plt.ylabel('Profit Value')

# set the title of the plot

plt.title('Relationship between Sales and Profit')

# save the plot to a file

plt.savefig('scatter\_plot.png', dpi=300, bbox\_inches='tight')

# display the plot

plt.show()

Copy

**5. Line Chart**

# import pandas and matplotlib.pyplot libraries

import pandas as pd

import matplotlib.pyplot as plt

# create a dictionary with sample data

data = {

'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],

'Monthly Sales': [100, 110, 105, 115, 120, 125, 130, 135, 140, 145, 150, 155]

}

# create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# plot the dataframe as a line chart

df.plot(x='Month', y='Monthly Sales', kind='line', marker='o')

# set the x-axis and y-axis labels

plt.xlabel('Month')

plt.ylabel('Sales Value')

# set the title of the plot

plt.title('Monthly Sales Trend over a Year')

# save the plot to a file

plt.savefig('line\_chart.png', dpi=300, bbox\_inches='tight')

# display the plot

plt.show()

Copy

**6. Heatmap**

# import required libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sample correlation matrix

data = {

'Product A': [1, 0.6, 0.8, 0.3],

'Product B': [0.6, 1, 0.5, 0.7],

'Product C': [0.8, 0.5, 1, 0.4],

'Product D': [0.3, 0.7, 0.4, 1]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Plotting the heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(df, annot=True, cmap='viridis', vmin=0, vmax=1)

# Set the title of the plot

plt.title('Product Correlation Heatmap')

# Save the plot to a file

plt.savefig('heatmap.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

**7. Boxplot**

# import required libraries

import pandas as pd

import matplotlib.pyplot as plt

# Sample data

data = {

'Product A': [95, 85, 90, 100, 105, 110],

'Product B': [100, 105, 95, 90, 115, 120],

'Product C': [105, 110, 115, 120, 125, 130]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Plotting the boxplot

df.boxplot(grid=False, vert=True, fontsize=10)

# Set the title of the plot

plt.title('Sales Distribution by Product')

# Set the y-axis label

plt.ylabel('Sales Value')

# Save the plot to a file

plt.savefig('boxplot.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

Box plot can also be created using seaborn library as follows

# we can do the same thing with seaborn library as follows

# import required libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sample data

data = {

'Product A': [95, 85, 90, 100, 105, 110],

'Product B': [100, 105, 95, 90, 115, 120],

'Product C': [105, 110, 115, 120, 125, 130]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Plotting the boxplot using seaborn

plt.figure(figsize=(10, 6))

sns.boxplot(data=df)

# Set the title of the plot

plt.title('Sales Distribution by Product')

# Set the y-axis label

plt.ylabel('Sales Value')

# Save the plot to a file

plt.savefig('sns\_boxplot.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

**8. Area Chart**

# import required libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sample data

data = {

'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May'],

'Sales': [100, 150, 180, 220, 270] # Accumulated sales for demonstration

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Plotting the area chart

plt.figure(figsize=(10, 6))

sns.set\_style("whitegrid")

plt.fill\_between(df['Month'], df['Sales'], color="skyblue", alpha=0.4)

plt.plot(df['Month'], df['Sales'], color="Slateblue", alpha=0.6)

# Set the title, x-axis and y-axis labels

plt.title('Monthly Accumulated Sales')

plt.xlabel('Month')

plt.ylabel('Accumulated Sales Value')

# Save the plot to a file

plt.savefig('area\_chart.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

**9. Spider or Radar Plot**

# import required libraries

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

# Sample data

data = {

'Metrics': ['Metric A', 'Metric B', 'Metric C', 'Metric D', 'Metric E'],

'Product 1': [3, 4, 5, 4, 2],

'Product 2': [4, 5, 4, 3, 3]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Number of variables

categories = list(df['Metrics'])

N = len(categories)

# Set the angle for each metric's axis

angles = [n / float(N) \* 2 \* np.pi for n in range(N)]

angles += angles[:1]

# Initialize the spider plot

plt.figure(figsize=(8, 6))

ax = plt.subplot(111, polar=True)

# Plotting for Product 1

values = df['Product 1'].tolist()

values += values[:1] # Repeat the first value to close the circular graph

ax.plot(angles, values, linewidth=2, linestyle='solid', label='Product 1')

ax.fill(angles, values, 'b', alpha=0.1)

# Plotting for Product 2

values = df['Product 2'].tolist()

values += values[:1] # Repeat the first value to close the circular graph

ax.plot(angles, values, linewidth=2, linestyle='solid', label='Product 2')

ax.fill(angles, values, 'r', alpha=0.1)

# Add the axis labels

plt.xticks(angles[:-1], categories, color='grey', size=12)

ax.set\_rlabel\_position(30)

plt.yticks([1,2,3,4,5], ["1","2","3","4","5"], color="grey", size=8)

plt.ylim(0, 5)

# Add title and legend

plt.title('Comparison between Product 1 and Product 2 on various metrics')

ax.legend(loc='upper right', bbox\_to\_anchor=(0.1, 0.1))

# Save the plot to a file

plt.savefig('radar\_chart.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

**10. Tree Map**

# Import required libraries

import pandas as pd

import matplotlib.pyplot as plt

import squarify # install if you do not have it

# Sample data

data = {

'File Type': ['Documents', 'Videos', 'Images', 'Audio', 'Other'],

'Storage Used (GB)': [50, 150, 80, 60, 10]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Plotting the treemap

plt.figure(figsize=(12, 8))

colors = ['red', 'green', 'blue', 'yellow', 'grey']

squarify.plot(sizes=df['Storage Used (GB)'], label=df['File Type'], color=colors, alpha=0.6)

# Add title

plt.title('Storage Used by File Type')

plt.axis('off') # Turn off the axis

# Save the plot to a file

plt.savefig('treemap.png', dpi=300, bbox\_inches='tight')

# Display the plot

plt.show()

Copy

**11. Geographical Maps**

# Import required libraries

import plotly.graph\_objects as go

# Data for cities and their coordinates

cities = {

'New York': [40.7128, -74.0060],

'London': [51.5074, -0.1278],

'Sydney': [-33.8688, 151.2093],

'Tokyo': [35.6895, 139.6917],

'Rio de Janeiro': [-22.9068, -43.1729]

}

# Extracting latitudes, longitudes, and city names for plotting

lats = [coords[0] for coords in cities.values()]

longs = [coords[1] for coords in cities.values()]

names = list(cities.keys())

# Creating the Scattergeo plot

fig = go.Figure(data=go.Scattergeo(

lon = longs,

lat = lats,

text = names,

mode = 'markers',

marker = dict(

size = 10,

opacity = 0.8,

reversescale = True,

autocolorscale = False,

symbol = 'circle',

line = dict(

width=1,

color='rgba(102, 102, 102)'

),

)

))

# Setting the layout for the map

fig.update\_layout(

title = 'Cities on World Map using Plotly',

geo = dict(

scope='world',

showland = True,

)

)

# Display the map

fig.show()

# Save the map to an HTML file

fig.write\_html("world\_map\_plotly.html")

Copy

**12. Time Series plots**

# Import required libraries

import plotly.graph\_objects as go

import pandas as pd

# Sample time series data

data = {

'Date': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04', '2022-01-05'],

'Value': [10, 15, 7, 12, 10]

}

# Create a pandas dataframe from the dictionary

df = pd.DataFrame(data)

# Convert the 'Date' column to a datetime object for better handling

df['Date'] = pd.to\_datetime(df['Date'])

# Creating the time series plot

fig = go.Figure(data=go.Scatter(x=df['Date'], y=df['Value'], mode='lines+markers'))

# Setting the layout for the plot

fig.update\_layout(

title='Time Series Plot using Plotly',

xaxis\_title='Date',

yaxis\_title='Value',

xaxis=dict(showline=True, showgrid=False, showticklabels=True),

yaxis=dict(zeroline=False, showgrid=True, showline=False, showticklabels=True),

)

# Display the plot

fig.show()

# Save the plot to an HTML file

fig.write\_html("time\_series\_plotly.html")

# save as png

fig.write\_image("time\_series\_plotly.png")

Copy

**13. Bubble Chart**

# Import required libraries

import plotly.graph\_objects as go

# Sample data

data = {

'X': [10, 20, 30, 40, 50],

'Y': [15, 25, 35, 45, 55],

'Size': [30, 60, 90, 120, 150], # Determines the size of bubbles

'Labels': ['A', 'B', 'C', 'D', 'E']

}

# Creating the bubble chart

fig = go.Figure(data=go.Scatter(

x=data['X'],

y=data['Y'],

mode='markers',

text=data['Labels'],

marker=dict(

size=data['Size'],

sizemode='diameter', # 'diameter' ensures that the size values correspond to the diameter of the bubbles

opacity=0.6,

color=data['Y'], # Coloring the bubbles based on Y values

colorscale='Viridis',

colorbar=dict(title='Colorbar')

)

))

# Setting the layout for the plot

fig.update\_layout(

title='Bubble Chart using Plotly',

xaxis\_title='X Values',

yaxis\_title='Y Values',

showlegend=False

)

# Display the plot

fig.show()

# Save the plot to an HTML file

fig.write\_html("bubble\_chart\_plotly.html")

# save png file with 300 dpi

fig.write\_image("bubble\_chart\_plotly.png", scale=3)

Copy

**14. Violin plots**

A violin plot is similar to a box plot but also includes a rotated kernel density plot on each side. It provides a visualization of the distribution of the data, its probability density, and its cumulative distribution.

Here’s how to create a violin plot using Plotly:

# Import required libraries

import plotly.express as px

# Sample data

data = {

'Category': ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C'],

'Values': [10, 15, 5, 25, 20, 10, 30, 25, 5, 20, 15, 10]

}

# Creating the violin plot using plotly express

fig = px.violin(data, y="Values", x="Category", box=True, points="all")

# if box=False then no Boxplots will be there inside violins

# Setting the layout for the plot

fig.update\_layout(

title='Violin Plot using Plotly',

yaxis\_title='Values',

xaxis\_title='Category'

)

# Display the plot

fig.show()

# Save the plot to an HTML file

fig.write\_html("violin\_plot\_plotly.html")

#save png file with 300 dpi

fig.write\_image("violin\_plot\_plotly.png", scale=3)

Copy