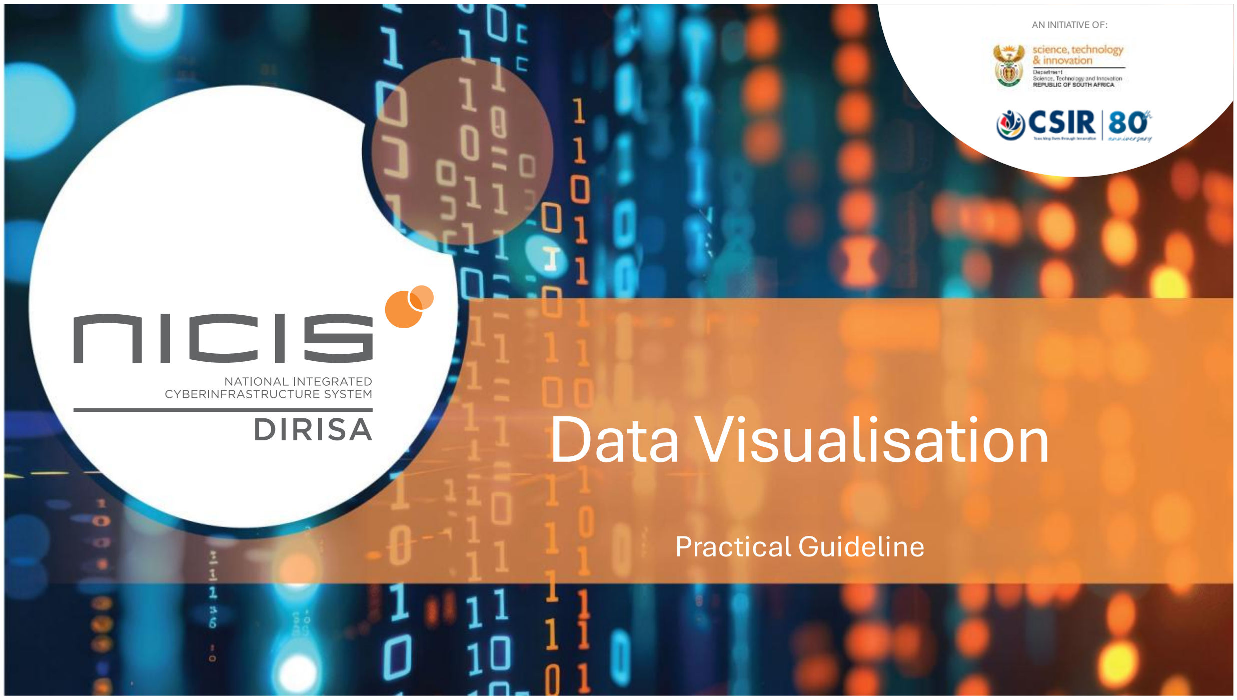
nicis
NATIONAL INTEGRATED CYBERINFRASTRUCTURE SYSTEM
DIRISA
AN INITIATIVE OF:
science, technology & innovation
Department: Science, Technology and Innovation
REPUBLIC OF SOUTH AFRICA
CSIR 80

Data Visualisation

Practical Guideline

Line Graph

Load Seaborn Datasets

```
import seaborn as sns
```

➤ Imports the Seaborn library for statistical data visualization.

```
sns.get_dataset_names()
```

➤ Returns a list of built-in Seaborn datasets you can load for testing or demos.

Load the Penguins Dataset

```
df = sns.load_dataset("penguins")
```

➤ Loads the "penguins" dataset into a DataFrame named df.

```
display(df)
```

➤ Displays the full DataFrame (in Jupyter, similar to print() but cleaner).

Plot Bill Length and Flipper Length

```
import matplotlib.pyplot as plt
```

➤ Imports Matplotlib’s Pyplot module.

```
plt.plot(df['bill_length_mm'], label='Bill length')
```

➤ Plots a line graph of **bill length**.

```
plt.plot(df['flipper_length_mm'], label='Flipper length (mm)')
```

➤ Adds another line graph for **flipper length**.

```
plt.xlabel('Penguin')
```

➤ Sets x-axis label.

```
plt.ylabel('Millimeter')
```

➤ Sets y-axis label.

```
plt.title('Plotting 2 variables.')
```

➤ Adds a title to the plot.

```
plt.legend()
```

➤ Displays a legend for both lines.

```
plt.grid()
```

➤ Adds a grid for better readability.

Plot Body Mass and Flipper Length

```
plt.plot(df['body_mass_g'], label='body mass (grams)')
```

➤ Line plot of body mass in grams.

```
plt.plot(df['flipper_length_mm'], label='Flipper length (mm)')
```

➤ Line plot of flipper length.

```
plt.grid()
```

➤ Displays gridlines.

```
plt.xlabel('Penguin')

plt.ylabel('a.u.')
```

➤ Sets labels; "a.u." stands for **arbitrary units**.

```
plt.title('Plotting 2 variables.')
```

➤ Title of the plot.

```
plt.legend()
```

➤ Shows the legend.

Dual Y-Axis Plot

```
fig, ax = plt.subplots()
```

➤ Creates a figure and axes object.

```
ax.plot(df['body_mass_g'], color="blue", label='Body Mass (grams)')
```

➤ First line plot on the **left y-axis**.

```
ax.set_xlabel('Penguin')

ax.set_ylabel('Body mass (g)', color="blue")
```

➤ Labels for the x-axis and left y-axis.

```
ax2 = ax.twinx()
```

➤ Creates a **second y-axis** that shares the same x-axis.

```
ax2.plot(df['flipper_length_mm'], color="orange", label='Flipper length (mm)')
```

➤ Plots on the **right y-axis**.

```
ax2.set_ylabel('Flipper length (mm)', color="orange")
```

➤ Right y-axis label.

```
ax2.set_title('Plotting 2 variables with different ranges.')

ax2.grid()
```

➤ Adds title and gridlines.

Scatter Plot

```
plt.scatter(df['body_mass_g'], df['flipper_length_mm'])
```

➤ Creates a scatter plot showing the relationship between **body mass** and **flipper length**.

```
plt.xlabel('Body mass (grams)')

plt.ylabel('Flipper length (mm)')

plt.title('Weight vs Flipper length')

plt.grid()
```

➤ Adds axis labels, title, and grid.

Histogram

```
plt.hist(df['body_mass_g'], bins=80, rwidth=0.8)
```

➤ Plots histogram of body mass with **80 bins**, bar width is 80% of available space.

```
plt.ylabel('Nuber of samples within\nsample value range.')

plt.xlabel('Body mass (grams)')

plt.title('Body mass distribution (80 bins)')

plt.grid()
```

➤ Labels and grid; note spelling mistake: "Nuber" should be "Number".

Histogram - 8 Bins

```
plt.hist(df['body_mass_g'], bins=8, rwidth=0.95)
```

➤ Same as before but with **8 bins** and wider bars (95%).

```
plt.ylabel('Nuber of samples within\nsample value range.')

plt.xlabel('Body mass (grams)')

plt.title('Body mass distribution with 8 bins')
plt.grid()
```

➤ Histogram labels and title.

Seaborn Distribution Plot

```
sns.set_theme()
```

➤ Applies the current Seaborn theme to future plots.

```
p = sns.displot(df['body_mass_g'], bins=8, kde=True)
```

➤ Distribution plot (histogram + **KDE curve** for smooth density).

```
p.set(xlabel='Boday mass range', ylabel='Nuber of samples within value range.', title='Body mass dist.')
```

➤ Sets plot labels and title. ("Boday" = typo, should be "Body").

Load Diamonds Dataset

```
data = sns.load_dataset('diamonds')
```

➤ Loads the diamonds dataset into data.

```
display(data)
```

➤ Displays the dataset.

Barplot (Carat by Cut & Color)

```
sns.barplot(x='cut', y='carat', hue='color', data=data)
```

➤ Grouped bar plot: carat on y-axis, cut on x-axis, color as hue.

```
plt.legend(bbox_to_anchor=(1,0.5), loc="center left")
```

➤ Moves legend outside the plot on the right side.

```
plt.grid()
```

➤ Adds gridlines.

Heatmap

Cut & Color Group Counts

```
df_m = data.groupby(["cut", "color"]).size().unstack(level=0)
```

➤ Groups data by **cut and color**, counts occurrences, then reshapes (pivot table).

```
print(df_m)
```

➤ Prints the result (a matrix of frequencies).

```
sns.heatmap(df_m, annot=False)
```

➤ Draws a heatmap to show counts. annot=False = no number labels.

Pie Chart (“Fair” cut)

```
plot = df_m.plot.pie(y='Fair')
```

➤ Creates a pie chart of **color distribution within "Fair" cut**.

```
plt.legend(bbox_to_anchor=(1.1,0.5), loc="center left")
```

➤ Moves legend outside to the right of the chart.

Pie Chart of Color G

```
df_m_trans = df_m.transpose()
```

➤ Transposes the DataFrame so colors become columns and cuts become rows.

```
display(df_m_trans)
```

➤ Shows the transposed table.

```
plot = df_m_trans.plot.pie(y='G')
```

➤ Creates a pie chart of **cut distribution within color G**.

```
plt.legend(bbox_to_anchor=(1.1,0.5), loc="center left")
```

➤ Positions the legend.