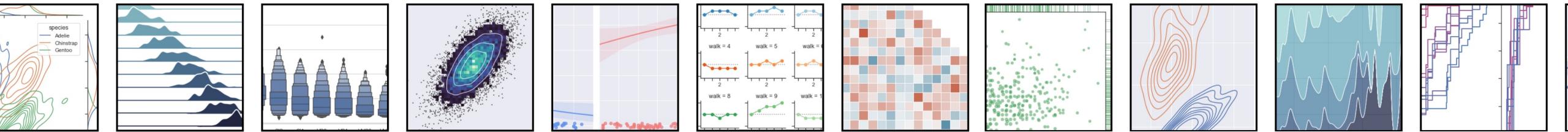


# DATA MANIPULATION & PUBLICATION- GRADE PLOTS

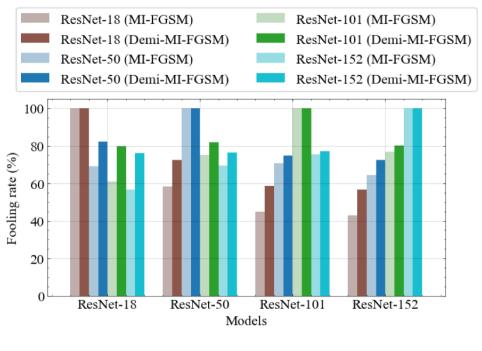
武上博

Feb 20, 2023

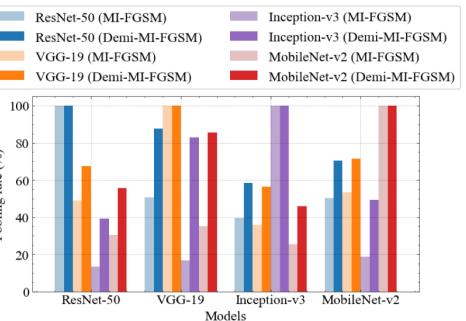
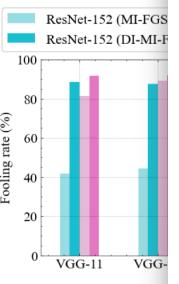


# Publication-grade Plots?

Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence



(a) Transferability among ResNet models



(b) Transferability among models with different architectures

Figure 6: We testify if in MI-FGSM ( $\ell_\infty$ ) can further our final attack: Demiguise attacks' fooling rates, is more than 5%, reaching over 90%.

Figure 3: Defense evaluation against seven adversarially trained models. Our proposed approach successfully improves every attack's transferability, achieving as high as almost 40% transfer fooling rate on models with defenses (rightmost bar).

Table 2: We compare the attack, Demiguise-HSJA, The fooling rate of ColorFool, while Demiguise- $\ell_2$ , reaching fooling rate

to be of great concern FGSM. [Xie et al., 20152, and transfer gen

### 4.3 Evading defenses

Being able to generalize across various model architectures is what especially makes ABD superior, which is only achievable as ABD successfully exploits the structural traits of ViTs. By dropping blocks of the transformer at a designated probability, our approach is able to structurally diversify local surrogate ViTs to a great extent, breaking structure or even architecture-wise connections between model gradients and adversarial perturbations, thereby avoiding overfitting to a specific model architecture, boosting cross-architectural transferability. Succinctly, we prove our proposed approach's comprehensive superiority and universality even against models with different architectures.

The advent of ViTs has brought a new class of adversarial attacks that significantly outperform traditional ones. However, the success of methods that actually exploit the structural traits of ViTs is very limited as we have shown in Figure 6. Specifically, for ViT, our proposed approach is designed to be used in combination with MIM. As we mentioned earlier, MIM is used as the baseline attack, while ABD is used as the target attack.

We attack 2 CNNs, 2 ViTs, and 2 DeiT models respectively. Results show that ABD always manages to generate adversarial examples that are more diverse than those generated by MIM. Specifically, for ResNet-50, the fooling rate of ABD is 94.9%, while that of MIM is 48.1%. For T2T-ViT-24, the fooling rate of ABD is 94.9%, while that of MIM is 75.0%. For DeiT-B/16, the fooling rate of ABD is 94.9%, while that of MIM is 75.0%. For DeiT-small, the fooling rate of ABD is 94.9%, while that of MIM is 75.0%.

**Diversifying local models.** It is as expected that the classification accuracy of different ViTs is almost always negatively correlated to probability threshold  $P$  (first row in Figure 6). As  $P$  increases, the number of dropped blocks in the transformer also increases, reducing the stability and accuracy of the ViT. However, this is the very act that enriches

that exploit these architectural features. Our proposed approach involving dropping partial blocks is novel and flexible, achieving significant performance gains. Specifically, we find that ignoring the attention gradient generated by partial blocks at each iteration prevents overfitting from interfering with the self-attention mechanism, thereby improving

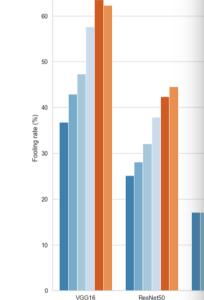
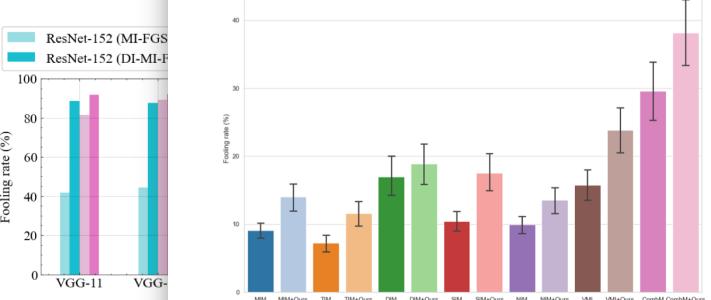


Figure 4: Comparing ABD vs. other attacks. ABD's fooling rate is higher than others. ABD's fooling rate is higher when used in combination with MIM.



Figure 5: GradCAM visualizations of the model's attention on adversarial examples generated by compared methods (baseline attack MIM, MIM + SE, MIM + PNA, MIM + Ours). Without exploiting ViT-specific architectural characteristics, the model's attention is not diverted, resulting in a failed transfer attack.

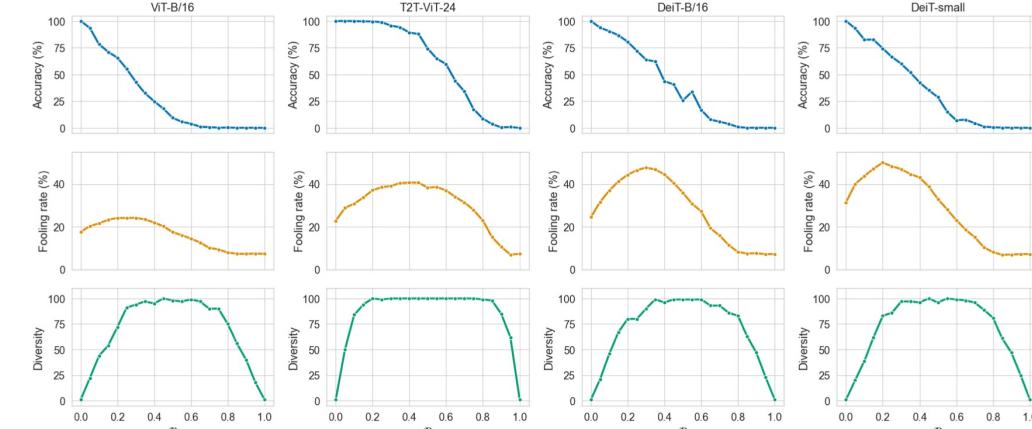


Figure 6: Relationship of  $P$  vs. (1) model accuracy (top row), (2) transfer fooling rate (middle row), and (3) model structure diversity (bottom row) across all four surrogate ViTs.  $P$  is grid searched within range  $[0, 1]$  with a step size of 0.05.

# Data Manipulation

- Logging data: CSV, Excel, even plain text files.

| Model      | Attack       | MIM    | MIM+Ours | TIM    | TIM+Ours | DIM    | DIM+Ours |
|------------|--------------|--------|----------|--------|----------|--------|----------|
| T2T-ViT-24 | Inc-V3       | 10.50% | 13.50%   | 8.10%  | 12.50%   | 17.70% | 18.10%   |
| T2T-ViT-24 | Inc-V4       | 12.50% | 18.20%   | 9.00%  | 16.60%   | 28.40% | 30.20%   |
| T2T-ViT-24 | IncRes-v2    | 4.70%  | 6.40%    | 4.20%  | 5.50%    | 8.50%  | 8.80%    |
| T2T-ViT-24 | Res-v2       | 6.70%  | 11.50%   | 3.30%  | 7.20%    | 17.20% | 20.00%   |
| T2T-ViT-24 | Inc-V3ens3   | 10.90% | 13.80%   | 9.60%  | 11.30%   | 16.90% | 17.60%   |
| T2T-ViT-24 | Inc-V3ens4   | 10.90% | 14.20%   | 10.90% | 12.70%   | 17.50% | 18.10%   |
| T2T-ViT-24 | IncRes-v2ens | 4.70%  | 6.40%    | 4.20%  | 5.50%    | 8.50%  | 8.80%    |
| ViT-B/16   | Inc-V3       | 9.50%  | 11.30%   | 7.50%  | 9.40%    | 12.50% | 13.40%   |
| ViT-B/16   | Inc-V4       | 7.80%  | 11.90%   | 4.60%  | 7.10%    | 10.10% | 12.90%   |
| ViT-B/16   | IncRes-v2    | 5.20%  | 7.10%    | 3.60%  | 4.80%    | 7.00%  | 8.20%    |
| ViT-B/16   | Res-v2       | 3.10%  | 4.40%    | 0.60%  | 1.60%    | 4.00%  | 5.00%    |
| ViT-B/16   | Inc-V3ens3   | 8.80%  | 10.00%   | 8.50%  | 10.00%   | 12.10% | 13.50%   |
| ViT-B/16   | Inc-V3ens4   | 9.70%  | 12.00%   | 10.10% | 11.80%   | 14.00% | 15.00%   |
| ViT-B/16   | IncRes-v2ens | 5.20%  | 7.10%    | 3.60%  | 4.80%    | 7.00%  | 8.20%    |
| DeiT-B/16  | Inc-V3       | 13.10% | 20.00%   | 9.70%  | 15.90%   | 18.30% | 22.00%   |
| DeiT-B/16  | Inc-V4       | 11.50% | 24.10%   | 6.60%  | 16.70%   | 20.10% | 23.00%   |
| DeiT-B/16  | IncRes-v2    | 6.20%  | 11.60%   | 5.00%  | 9.20%    | 10.90% | 14.10%   |
| DeiT-B/16  | Res-v2       | 6.00%  | 16.30%   | 1.00%  | 8.40%    | 12.00% | 14.40%   |
| DeiT-B/16  | Inc-V3ens3   | 10.10% | 19.00%   | 9.10%  | 15.50%   | 17.20% | 22.70%   |

defense.xlsx

```
outputs/True_DIM_MI_ResNet50_20_8_100_False
model_name succ_rate misclass_rate lpips
MyImageNetFastAT 0 314 0.2501748852990568
MyImageNetFreeAT 1 247 0.2501748852990568
Densenet121 90 406 0.2501748852990568
Densenet161 89 331 0.2501748852990568
Densenet169 101 351 0.2501748852990568
Densenet201 84 345 0.2501748852990568
InceptionV3 16 417 0.2501748852990568
VGG13 38 476 0.2501748852990568
VGG16 39 461 0.2501748852990568
VGG19 29 421 0.2501748852990568
Resnet18 67 484 0.2501748852990568
Resnet34 92 473 0.2501748852990568
Resnet50 998 999 0.2501748852990568
Resnet101 124 451 0.2501748852990568
Resnet152 94 374 0.2501748852990568
WideResnet50 105 403 0.2501748852990568
MyAdvEfficientnetb0 1 169 0.2501748852990568
MyAdvEfficientnetb1 2 140 0.2501748852990568
MyAdvEfficientnetb2 5 170 0.2501748852990568
```

outputs.txt

# Data Manipulation

## Recommended:

- File format: **CSV**.
- Use type-friendly:  
`csv.DictWriter`

## Relevant Documentation

- [Writing CSV File From a Dictionary With csv](#)
- [Python 3 - csv.DictReader/DictWriter](#)

```
import csv

header_fields = ["attack", "model", "eps", "fooling_rate"]
with open("results.csv", "w+") as f:
    writer = csv.DictWriter(f, fieldnames=header_fields)
    writer.writeheader()

    # Iterate over all experiments
    for some_exp in enum_experiments:
        writer.writerow(
            {
                "attack": attack,
                "model": model,
                "eps": eps,
                "fooling_rate": fooling_rate,
            }
        )
```

# Transforming Data

## Data Manipulation

### pandas – Python Data Analysis Library

- import pandas as pd
- Provide high-performance data structures and analysis tools.
- pd.DataFrame: for representing tabular data.
- Why? Most plotting libraries build on pandas' data structures.



Note: Looking for optimized speed? Try polars (rust-based).

# Transforming Data

## Data Manipulation

### Reading raw data:

- `df = pd.read_csv()`
- `df = pd.read_excel()` (pip install openpyxl)

### Initial read-out of the data:

- `df.head()`

### Access index and columns:

- `df.columns`
- `df.index`

```
import pandas as pd

df = pd.read_excel("../plot_ijcai23_abd/defense.xlsx")
df.head()
```

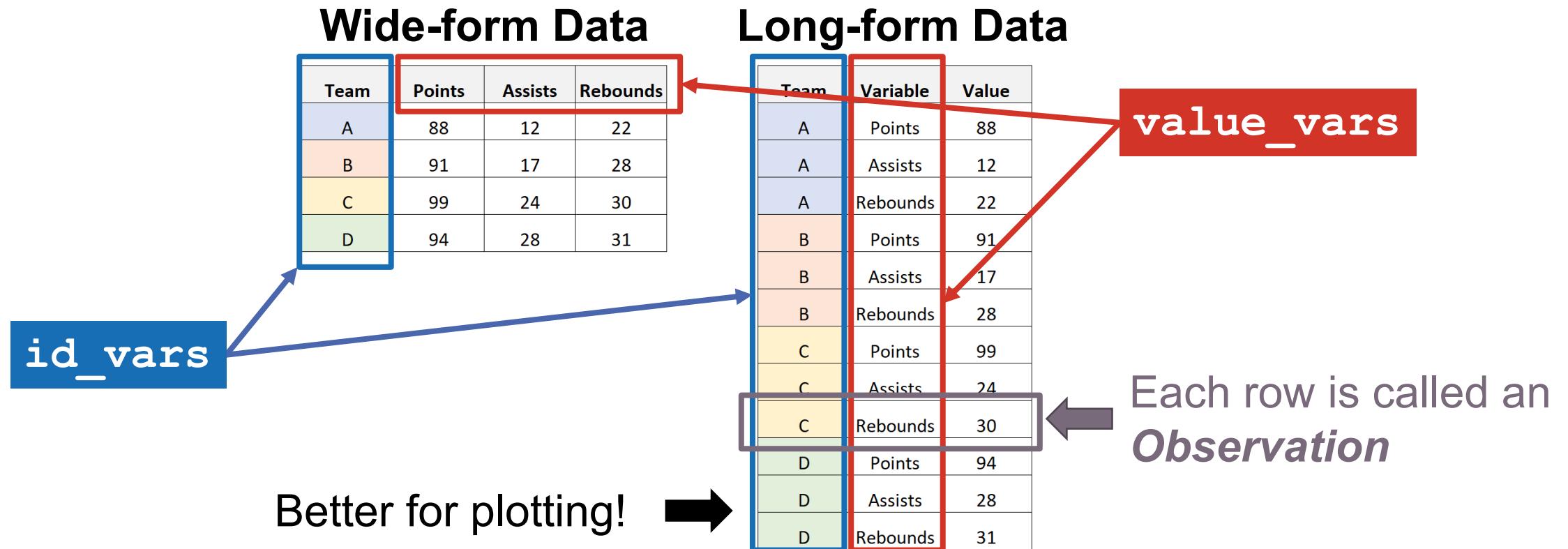
✓ 0.4s

|   | Model      | Attack     | MIM   | MIM+Ours | TIM   | TIM+Ours | DIM   | DIM+Ours | SIM   | SIM+Ours | NIM   |
|---|------------|------------|-------|----------|-------|----------|-------|----------|-------|----------|-------|
| 0 | T2T-ViT-24 | Inc-V3     | 0.105 | 0.135    | 0.081 | 0.125    | 0.177 | 0.181    | 0.115 | 0.152    | 0.106 |
| 1 | T2T-ViT-24 | Inc-V4     | 0.125 | 0.182    | 0.090 | 0.166    | 0.284 | 0.302    | 0.135 | 0.226    | 0.148 |
| 2 | T2T-ViT-24 | IncRes-v2  | 0.047 | 0.064    | 0.042 | 0.055    | 0.085 | 0.088    | 0.044 | 0.066    | 0.044 |
| 3 | T2T-ViT-24 | Res-v2     | 0.067 | 0.115    | 0.033 | 0.072    | 0.172 | 0.200    | 0.081 | 0.160    | 0.079 |
| 4 | T2T-ViT-24 | Inc-V3ens3 | 0.109 | 0.138    | 0.096 | 0.113    | 0.169 | 0.176    | 0.102 | 0.144    | 0.105 |

# Wide-form vs. Long-form Data

## Data Manipulation

- Data: *long* (or tidy) form, or *wide* form.



# Wide-form vs. Long-form Data

## Data Manipulation

Wide to long:

- `df.melt(id_vars=..., var_name=..., value_name=...)`

| Team | Points | Assists | Rebounds |
|------|--------|---------|----------|
| A    | 88     | 12      | 22       |
| B    | 91     | 17      | 28       |
| C    | 99     | 24      | 30       |
| D    | 94     | 28      | 31       |

df = df.melt(id\_vars="Team", var\_name="Stats", value\_name="Value")  
df

✓ 0.0s

|   | Team | Stats   | Value |
|---|------|---------|-------|
| 0 | A    | Points  | 88    |
| 1 | B    | Points  | 91    |
| 2 | C    | Points  | 99    |
| 3 | D    | Points  | 94    |
| 4 | A    | Assists | 12    |
| 5 | B    | Assists | 17    |
| 6 | C    | Assists | 24    |

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.melt.html>

Long to wide: `df.pivot().reset_index()`

# Aggregate?

## Data Manipulation

- Drop NaN: `df.dropna()` ([pandas.DataFrame.dropna](#))
- Replace: `df.replace()` ([pandas.DataFrame.replace](#))
- Sort: `df.sort_values()` ([pandas.DataFrame.sort\\_values](#))
- Rename: `df.rename()` ([pandas.DataFrame.rename](#))
- Apply lambda function: `df.apply()` ([pandas.DataFrame.apply](#))

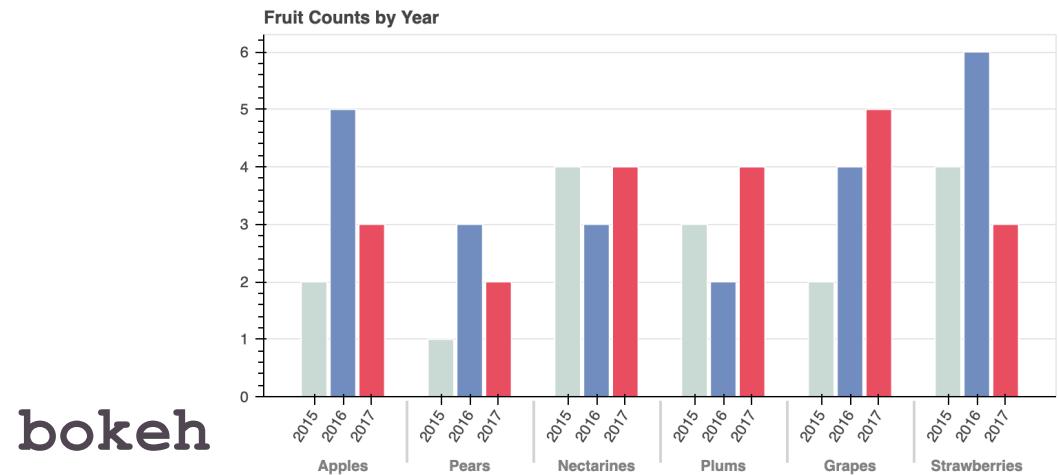
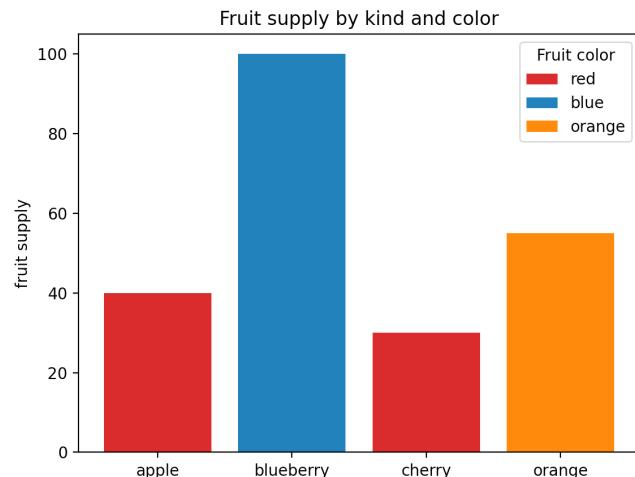


More on this: [pandas user guide - 10 minutes to pandas](#).



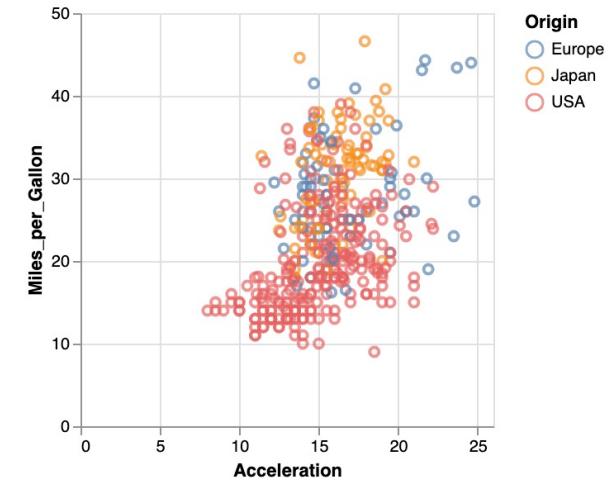
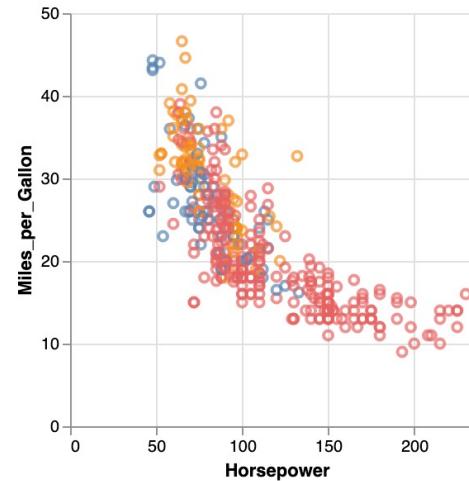
# Plot

Matplotlib  
plotly.py



bokeh

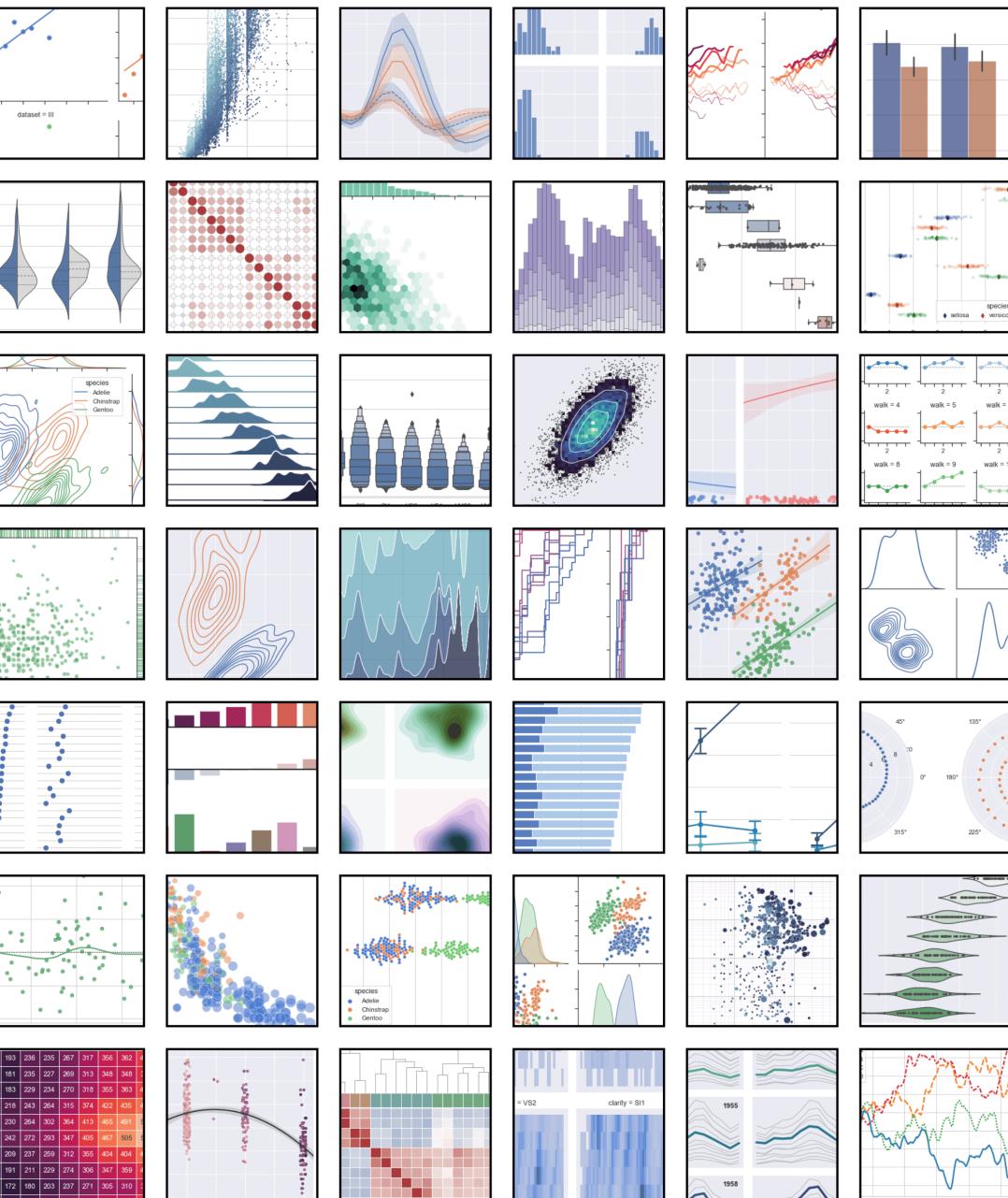
Altair (Vega / Vega-Lite)



# Plot

Use seaborn!

- Seaborn, statistical data visualization:
  - <https://seaborn.pydata.org>
- import seaborn as sns
- Good amount of abstraction over matplotlib.
- Great color/style defaults.



# Seaborn Styles

## Plots

```
sns.set_theme(context=..., style=..., palette=...,  
font=..., font_scale=...)
```

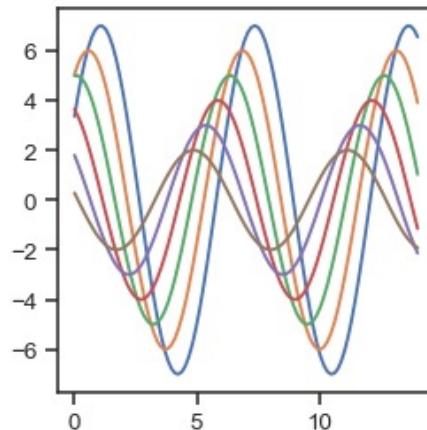
|                              |                               |                                     |
|------------------------------|-------------------------------|-------------------------------------|
| <b>context</b>               | Plotting context.             | notebook paper talk poster          |
| <b>style</b>                 | Axes style.                   | darkgrid whitegrid dark white ticks |
| <b>palette</b>               | Color palette.                | sns.color_palette()                 |
| <b>font &amp; font_scale</b> | Controls font style and size. |                                     |

# Seaborn Styles

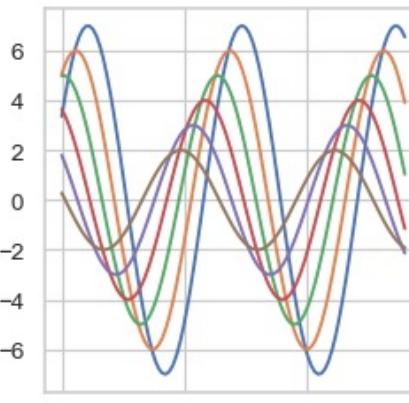
## Plots

For plots in publications, use:

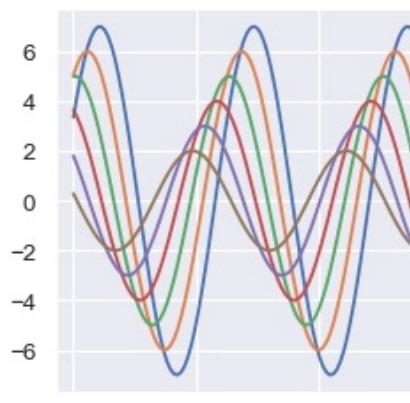
```
import seaborn as sns  
sns.set_theme(context="paper", style="whitegrid", font_scale=1.5)
```



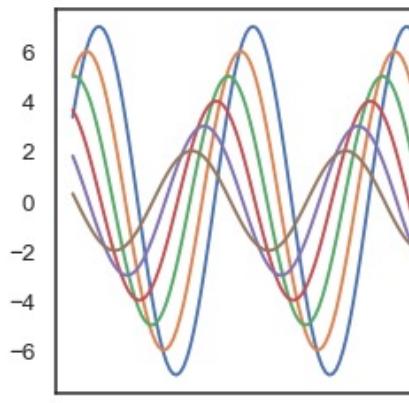
ticks



whitegrid



darkgrid



white

# Data Types

## Plots

### Attribute Type

- Categorical:



- Ordered:

- Ordinal



- Quantitative



### Ordering Direction

- Sequential:



- Diverging:



- Cyclic:



# Data Encoding

## Plots – Data Marks

### Data Marks:

- Points: scatter plots, sns.scatterplot(data, \*)
- Line: line plots, sns.lineplot(data, \*)
- Bar: bar plots, sns.barplot(data, \*)
- \*Area: heatmaps, sns.heatmap(data, \*)

# Data Encoding

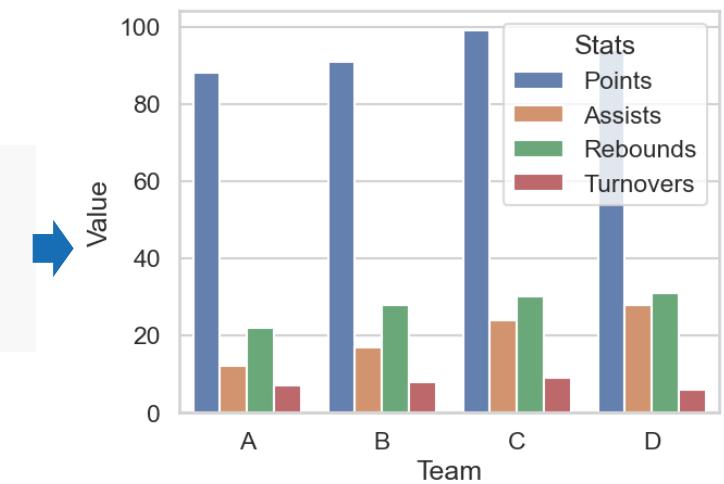
## Plots – Data Marks

`sns.barplot(data, x, y, hue, palette, ..., ax, ...)`

- `x, y, hue`: Some column name in the DataFrame.
- `palette`: Color palette name.
  - `deep, muted, bright, pastel, dark, colorblind`

|   | Team | Stats   | Value |
|---|------|---------|-------|
| 0 | A    | Points  | 88    |
| 1 | B    | Points  | 91    |
| 2 | C    | Points  | 99    |
| 3 | D    | Points  | 94    |
| 4 | A    | Assists | 12    |
| 5 | B    | Assists | 17    |
| 6 | C    | Assists | 24    |

```
import seaborn as sns  
sns.set_theme(context="talk", style="whitegrid")  
sns.barplot(df, x="Team", y="Value", hue="Stats")
```



# Data Encoding

## Plots – Color Palettes

Color Palettes – Chosen based on *data type*.

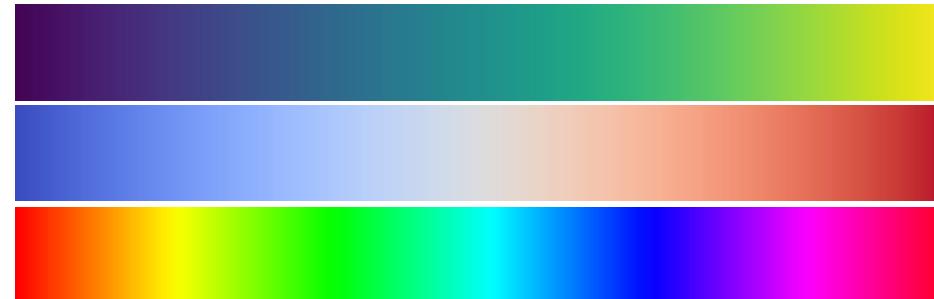
- Categorical:



`sns.color_palette("deep")`

- Ordinal:

- Sequential
- Diverging
- Cyclic



`sns.color_palette("viridis")`

`sns.color_palette("coolwarm")`

`sns.color_palette("hsv")`

[Tutorial: Choosing seaborn color palettes](#)

[Tutorial: Choosing Colormaps in Matplotlib](#)

# Data Encoding

## Plots – Color Palettes

### Categorical

- What if there are more categories than the default color cycle?

```
sns.color_palette("husl", 12)
```

✓ 0.0s



```
palette = sns.color_palette("husl", 4)
```

```
sns.barplot(df, x="Team", y="Value", hue="Stats", palette=palette)
```



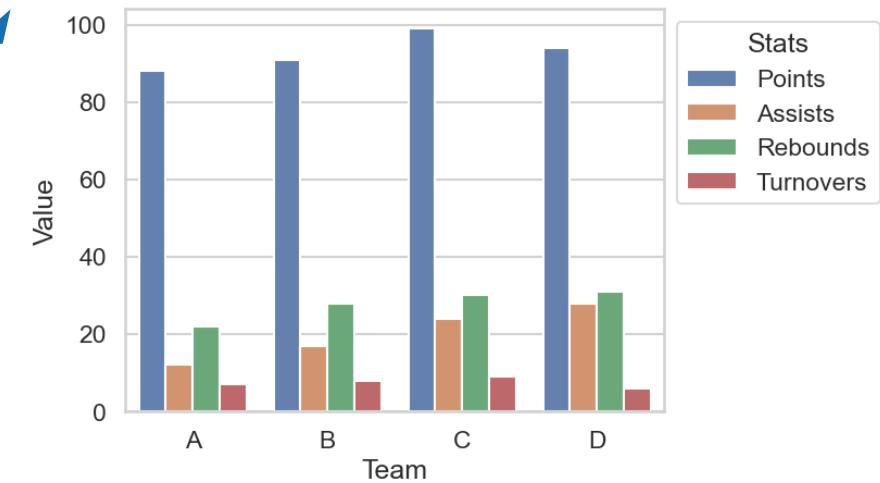
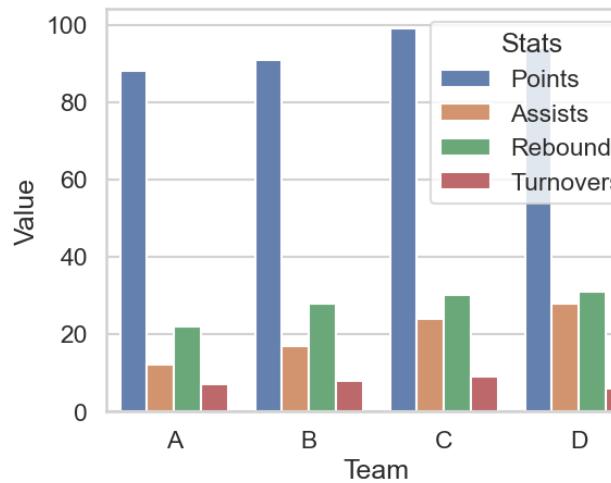
# Legend

## Plots – Legend

`sns.move_legend(ax, ...)`

- Example:

```
ax = sns.barplot(df, x="Team", y="Value", hue="Stats")  
sns.move_legend(ax, "upper left", bbox_to_anchor=(1, 1))
```



- [https://seaborn.pydata.org/generated/seaborn.move\\_legend.html](https://seaborn.pydata.org/generated/seaborn.move_legend.html)

# Extend Seaborn with Matplotlib

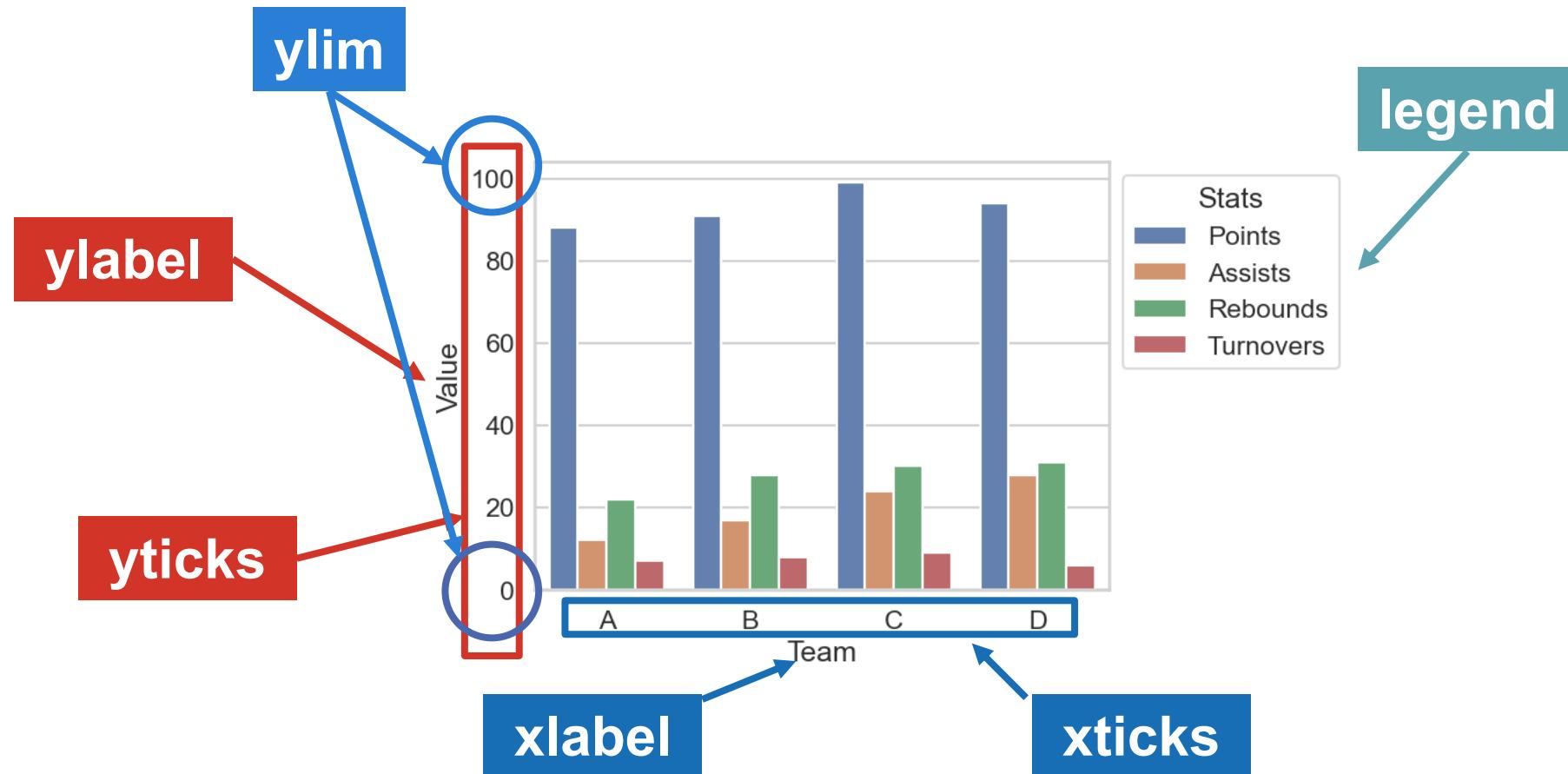
```
import matplotlib.pyplot as plt
```

**Used for:**

- Defining figure size: `plt.figure(figsize=(10, 6))`
- Setting axis attribute:
  - `set_xlabel`, `set_ylabel`, `set_xticks`, `set_yticks`,  
`set_xlim`, `set_ylim`
  - Subplots and more general control over the plot.

# Extend Seaborn with Matplotlib

## Matplotlib Figure

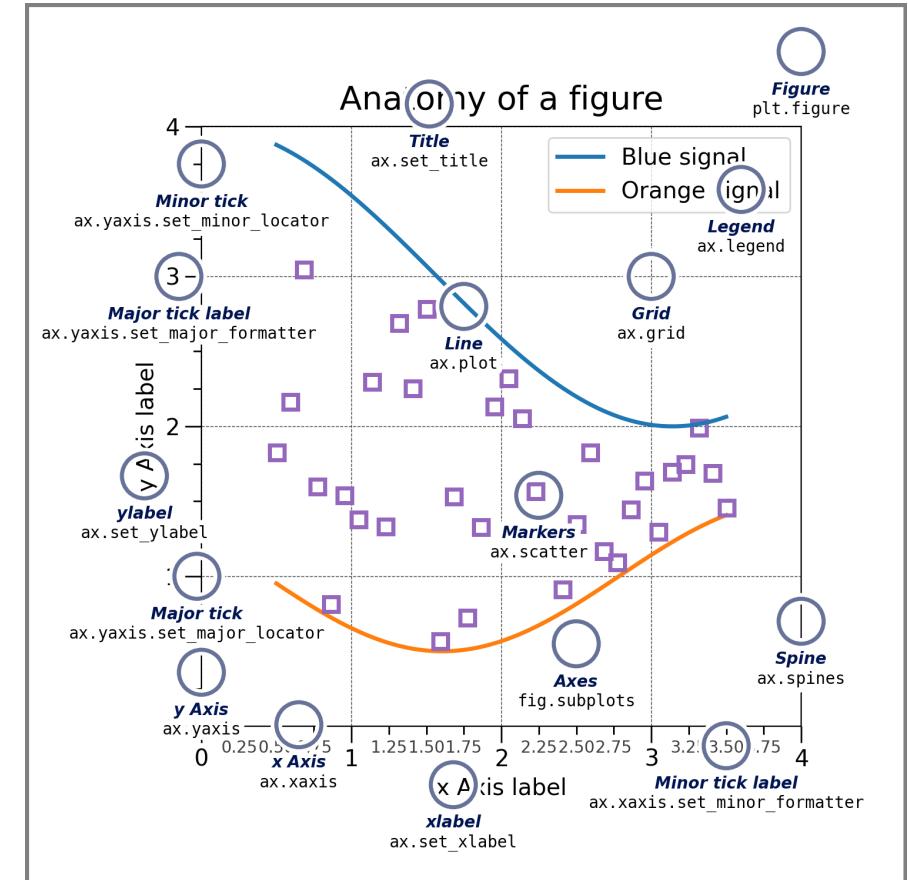


# Extend Seaborn with Matplotlib

## Matplotlib Figure

### Anatomy of a Matplotlib Figure

- [https://matplotlib.org/stable/tutorials/introductory/quick\\_start.html#parts-of-a-figure](https://matplotlib.org/stable/tutorials/introductory/quick_start.html#parts-of-a-figure)



# Extend Seaborn with Matplotlib

## Subplots and layout

General workflow:

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

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

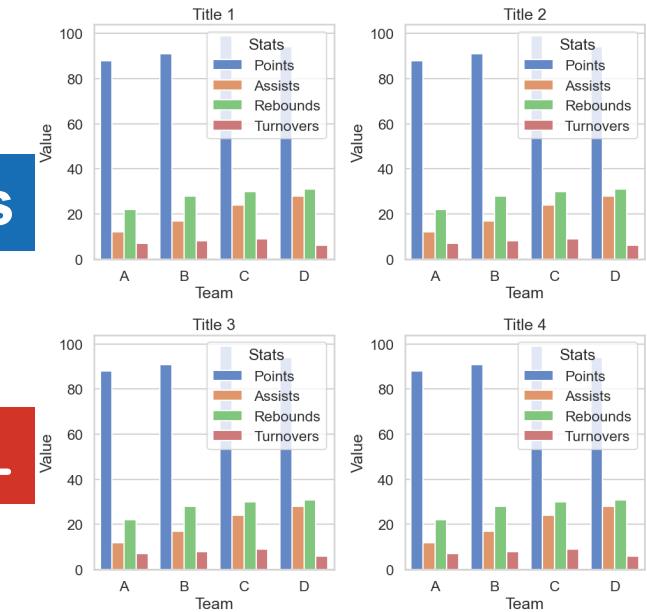
for i, axi in enumerate(axs.flat):
    sns.barplot(df, x="Team", y="Value", hue="Stats", palette="muted", ax=axi)
    axi.set_title(f"Title {i + 1}")

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

Two rows, two cols

Seaborn plot on axi

Set axi attribute  
(label, ticks, title...)



# Extend Seaborn with Matplotlib

## Subplots and layout

General workflow:

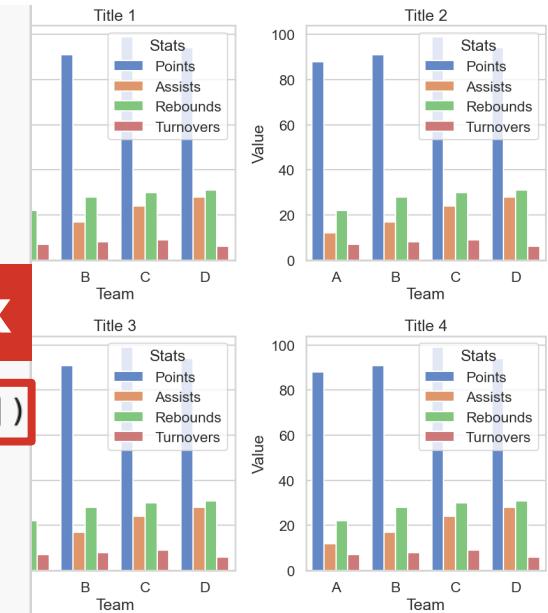
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

for i in range(2):
    for j in range(2):
        sns.barplot(df, x="Team", y="Value", hue="Stats", palette="muted", ax=axs[i, j])
        axs[i, j].set_title(f"Title {i + 1}")

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

Access axes by index

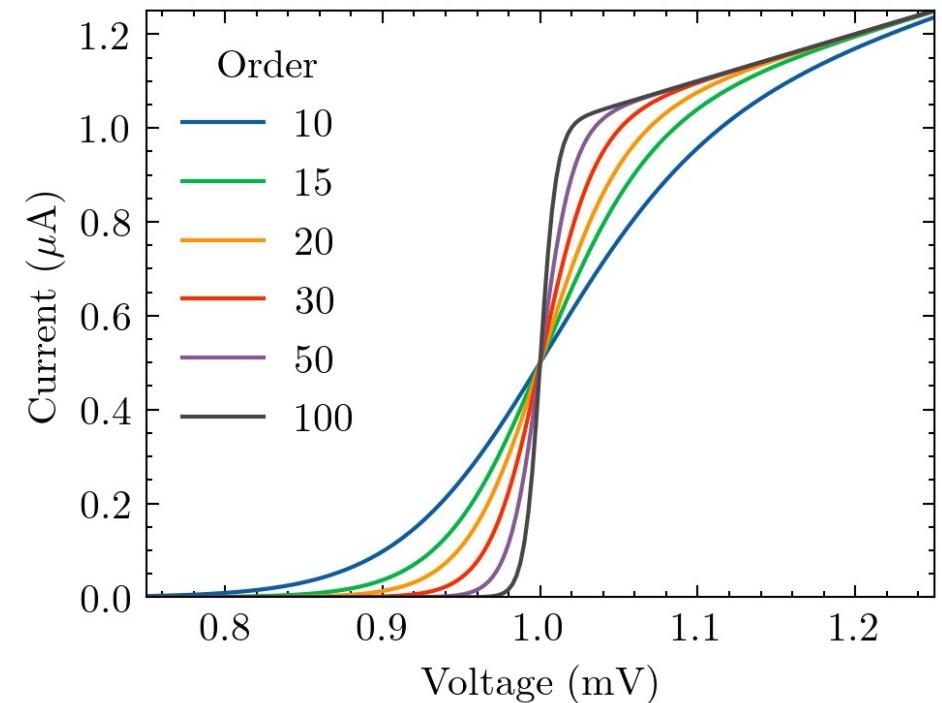


# Additionally

## Library – SciencePlots

### garrettj403/SciencePlots

- Render fonts with LaTeX.
- Professional colors and fonts.
- Can only be used with matplotlib!



# Thanks for listening

-  <https://github.com/spencerwoo>
-  <https://scholar.google.com/citations?user=Mf-JoyQAAAAJ>
-  <https://spencerwoo.com>