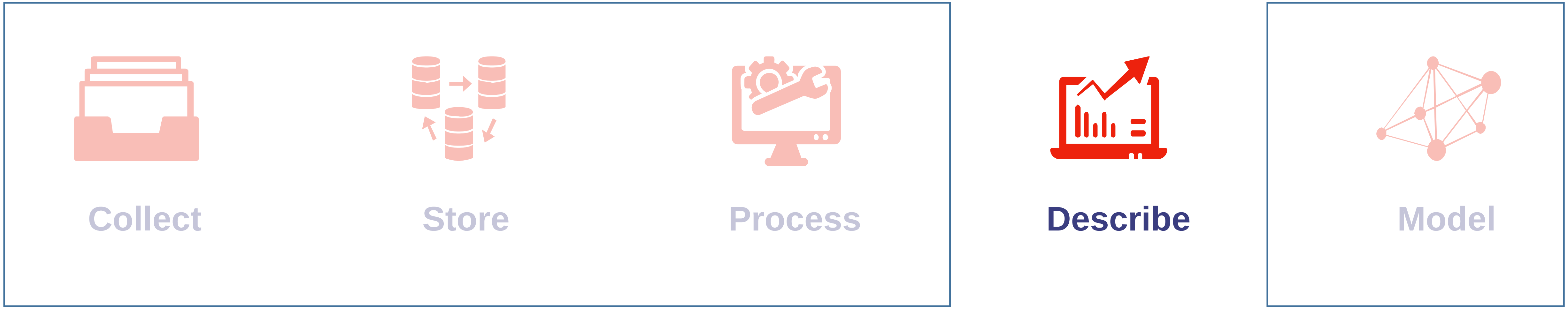
Data Visualization

Data Visualization

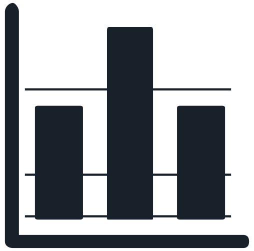


- **Describing data is an act of compressing information to focus on**
- **One way is to compute statistics**
- **The other way is to visualize the data**

Data Visualization

Describe

- Visual encoding of data



- 2021 March 01

What we are interested in are representations that can be decoded by us

This encoding can be identified by us, but can be decoded only by machines

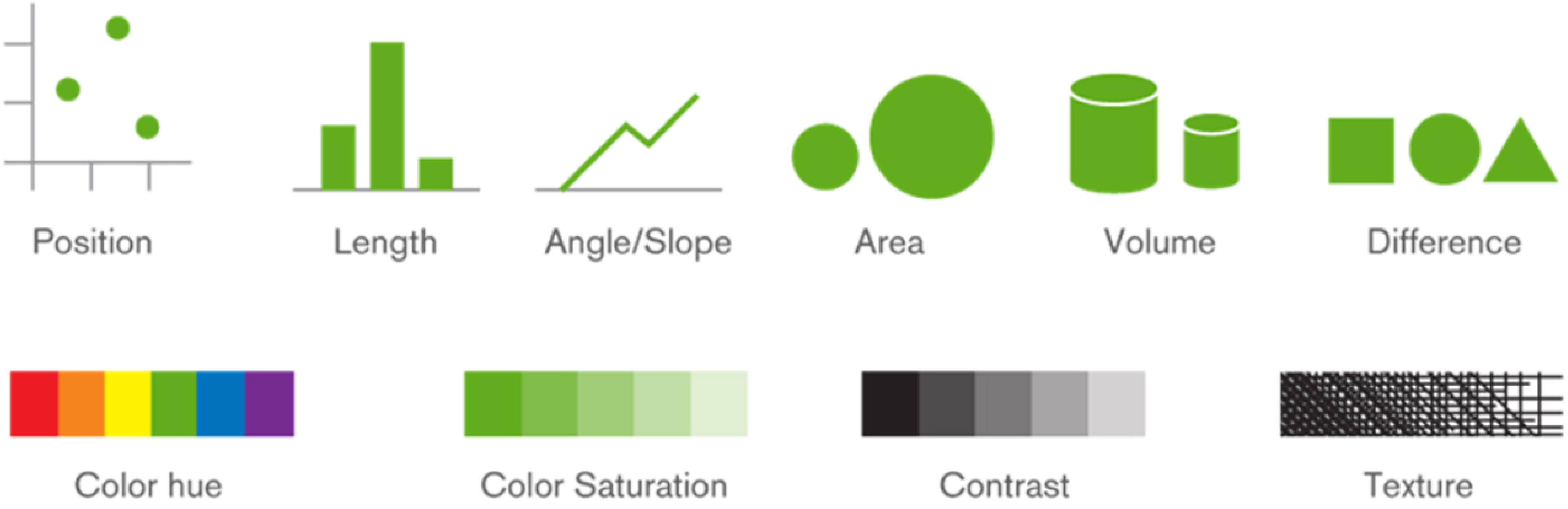
PRIME INTUIT
Finishing School

Data Visualization: What

Visual elements for encoding data

Here's a non-exhaustive list of ways you can encode data:

- Size
- Shape
- Colour
- Grouping
- Area
- Position
- Saturation
- Line pattern
- Line weight
- Angle
- Connections



PRIME INTUIT
Finishing School

Data Visualization: What

Visual elements for encoding data

Example	Encoding	Ordered	Useful values	Quantitative	Ordinal	Categorical	Relational
	position, placement	yes	infinite	Good	Good	Good	Good
1, 2, 3; A, B, C	text labels	optional alpha or num	infinite	Good	Good	Good	Good
	length	yes	many	Good	Good		
	size, area	yes	many	Good	Good		
	angle	yes	medium	Good	Good		
	pattern density	yes	few	Good	Good		
	weight, boldness	yes	few		Good		
	saturation, brightness	yes	few		Good		
	color	no	few (<20)			Good	
	shape, icon	no	medium			Good	
	pattern texture	no	medium			Good	
	enclosure, connection	no	infinite			Good	Good
	line pattern	no	few				Good
	line endings	no	few				Good
	line weight	yes	few		Good		

PRIME INTUIT
Finishing School

Visualization: What

Tabulation

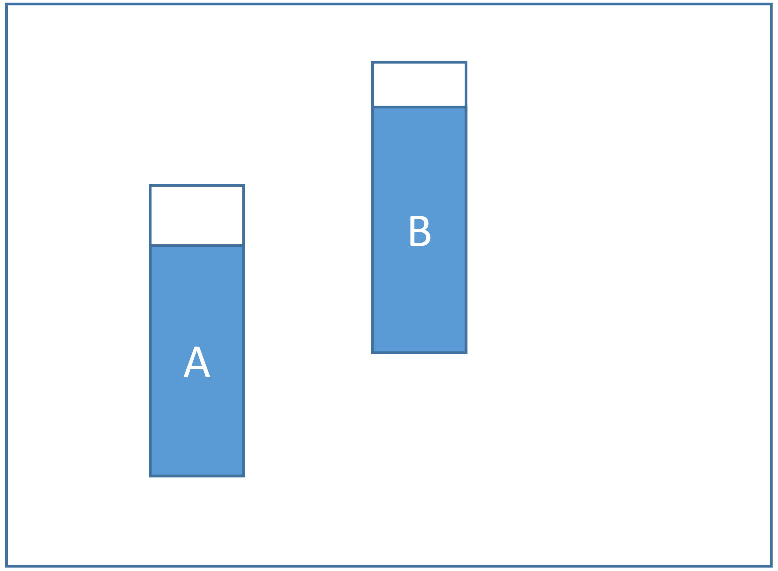
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3Dz9Sen1HTu5o&psig=AOvVaw2D8fbyLxSgGfuLNMCqmTNn&ust=1639585218518000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCJiQ5IzZ4_QCFQAAAAAdAAAAABAD

Import urllib.request

We make perceptual errors in decoding these elements

Our goal is to reduce perceptual errors

Which bar is longer A or B

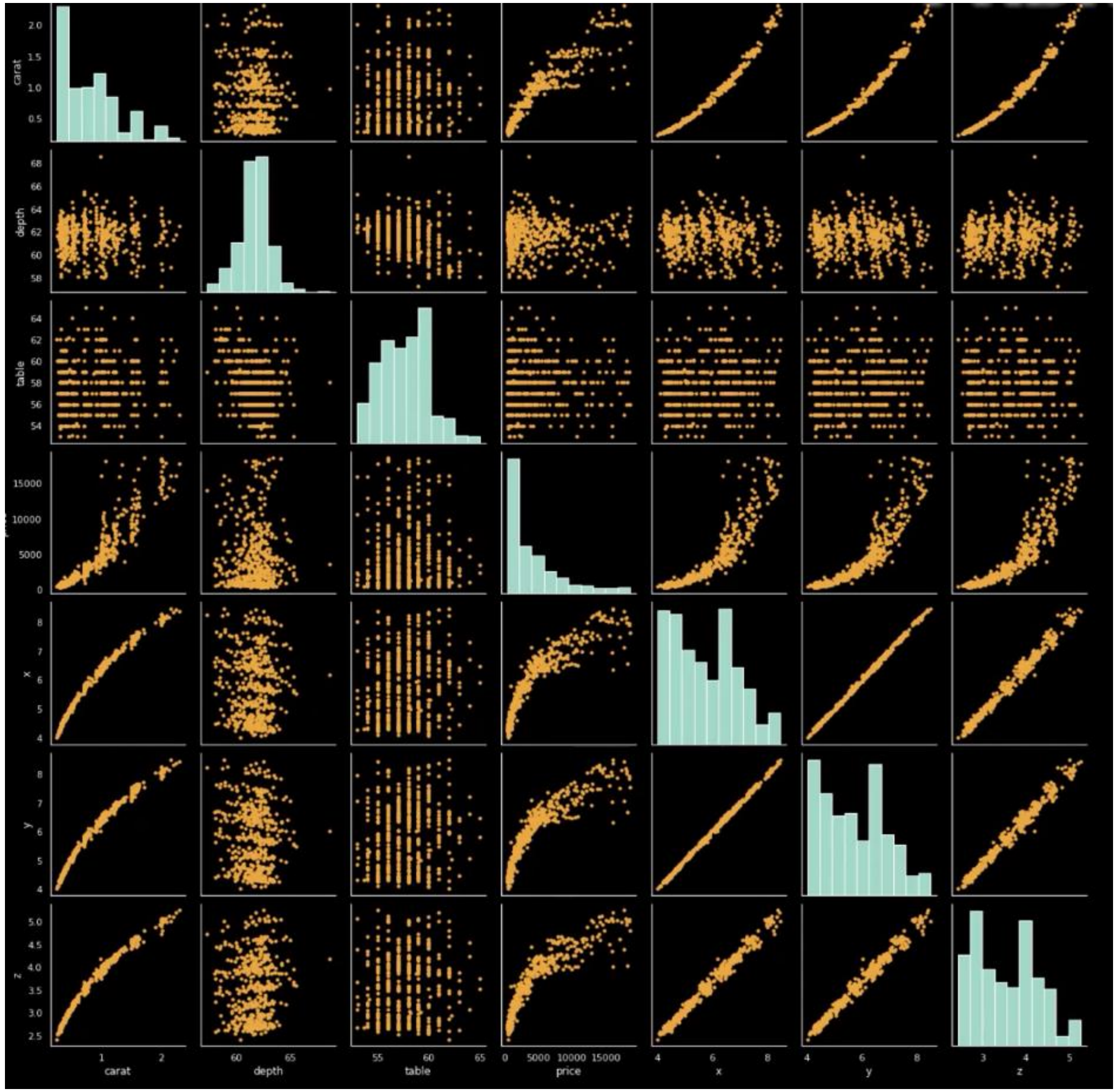


Visualization: Why

Discover insights from the data

Diamonds dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   carat     53940 non-null  float64
 1   cut       53940 non-null  object
 2   color     53940 non-null  object
 3   clarity   53940 non-null  object
 4   depth     53940 non-null  float64
 5   table     53940 non-null  float64
 6   price     53940 non-null  int64
 7   x         53940 non-null  float64
 8   y         53940 non-null  float64
 9   z         53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```



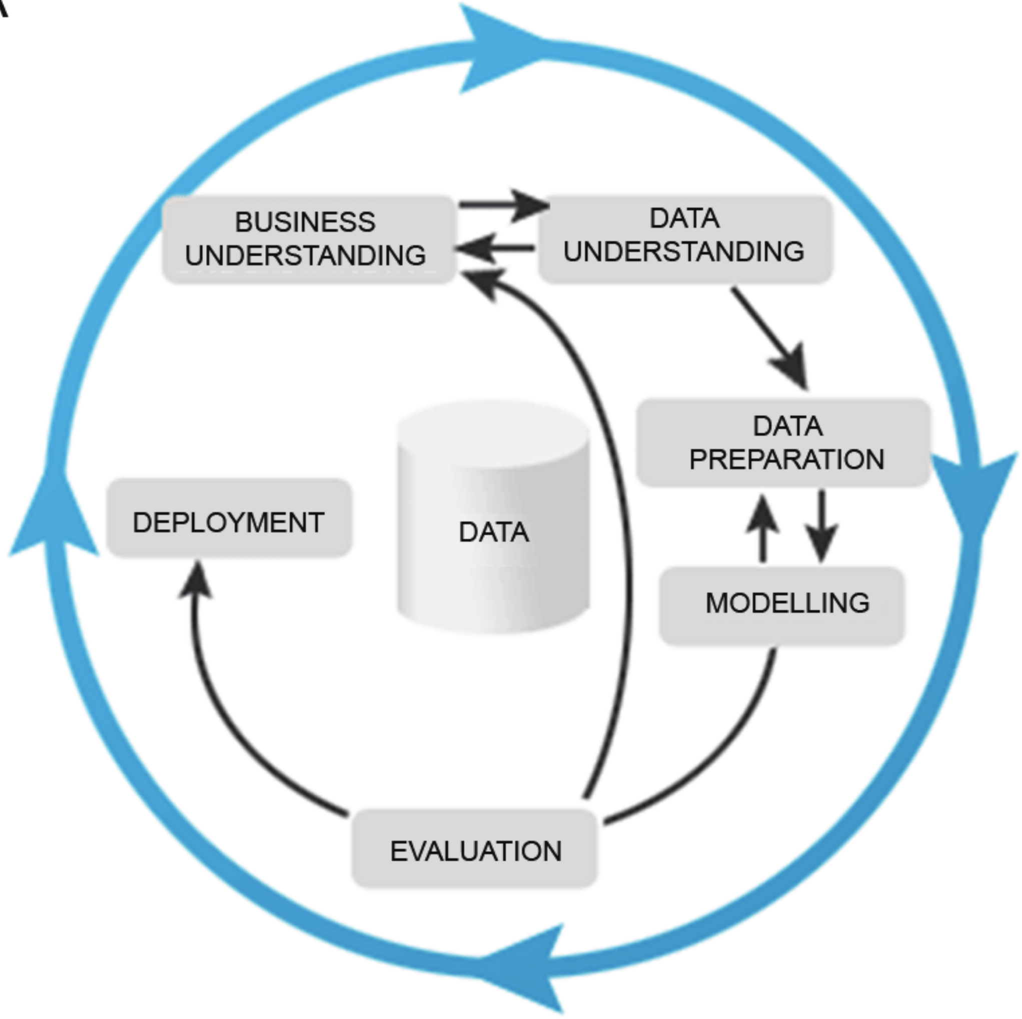
Data visualization as an aid to explore the data

Visualization: Why

Discover insights from the data

A

Communicate insights effectively

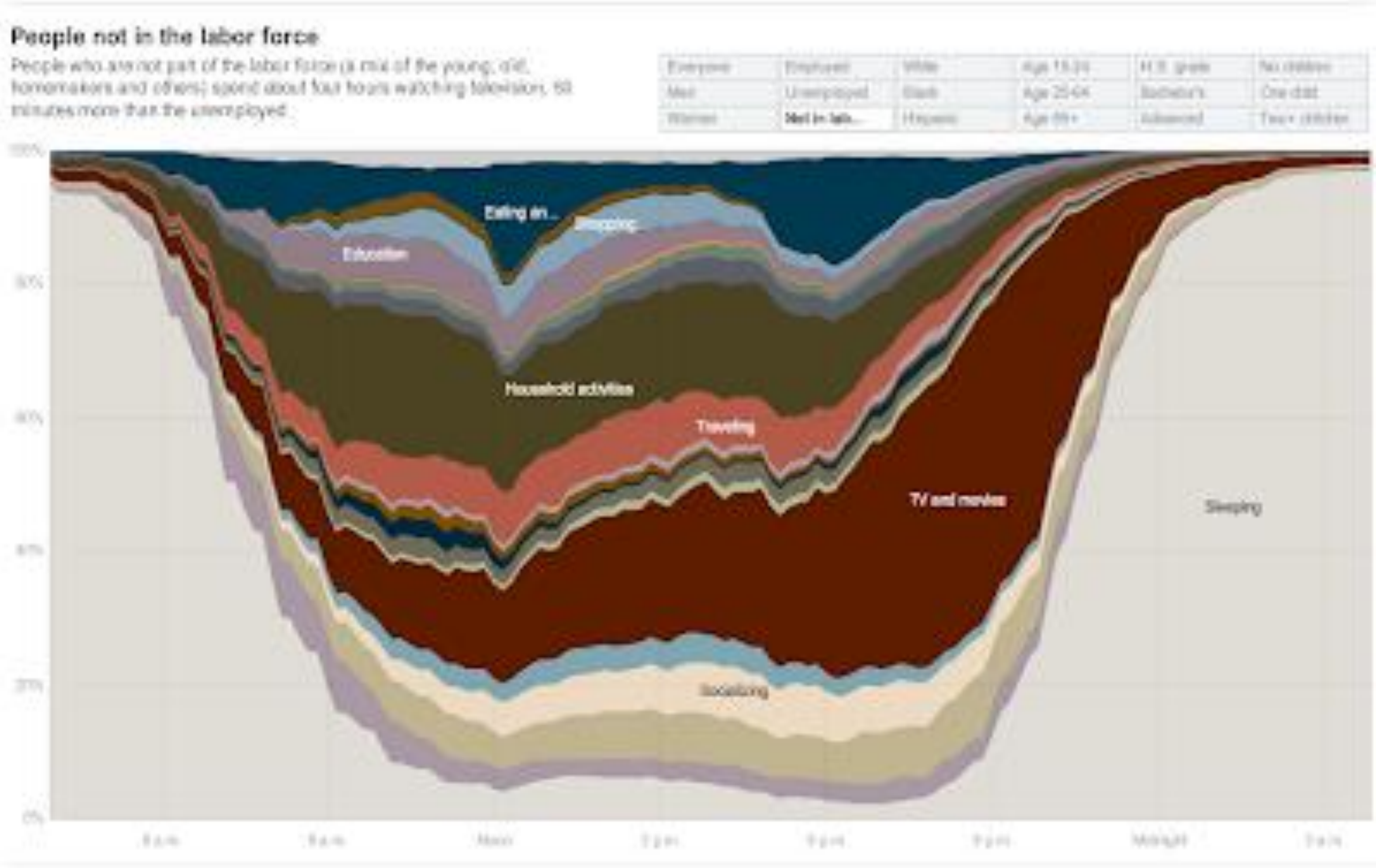


PRIME INTUIT
Finishing School

Visualization: Why

Discover insights from the data

Communicate insights effectively



PRIME INTUIT
Finishing School

Visualization: Why

Discover insights from the data

Animated gapminder plot by Hans Rosling

https://youtu.be/jbkSRLYSojo

Pitfalls in data visualization, - It’s not that important

https://www.maartenlambrechts.com/2017/01/25/an-ode-to-charts-from-1939.html

MAGIC IN GRAPHS

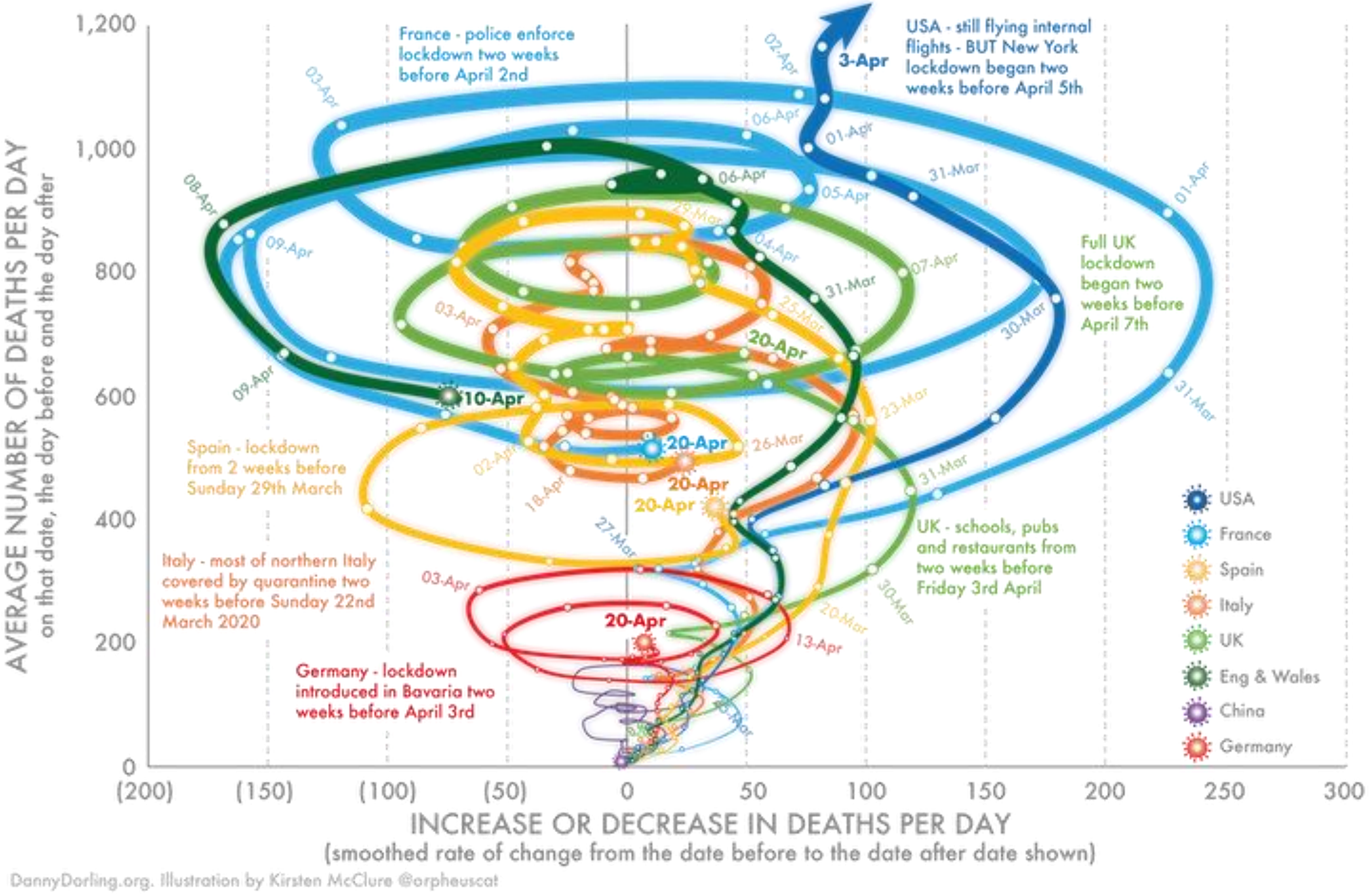
THERE is a magic in graphs. The profile of a curve reveals in a flash a whole situation —the life history of an epidemic, a panic, or an era of prosperity. The curve informs the mind, awakens the imagination, convinces.

PRIME INTUIT
Finishing School

Visualization: Why

Pitfalls in data visualization

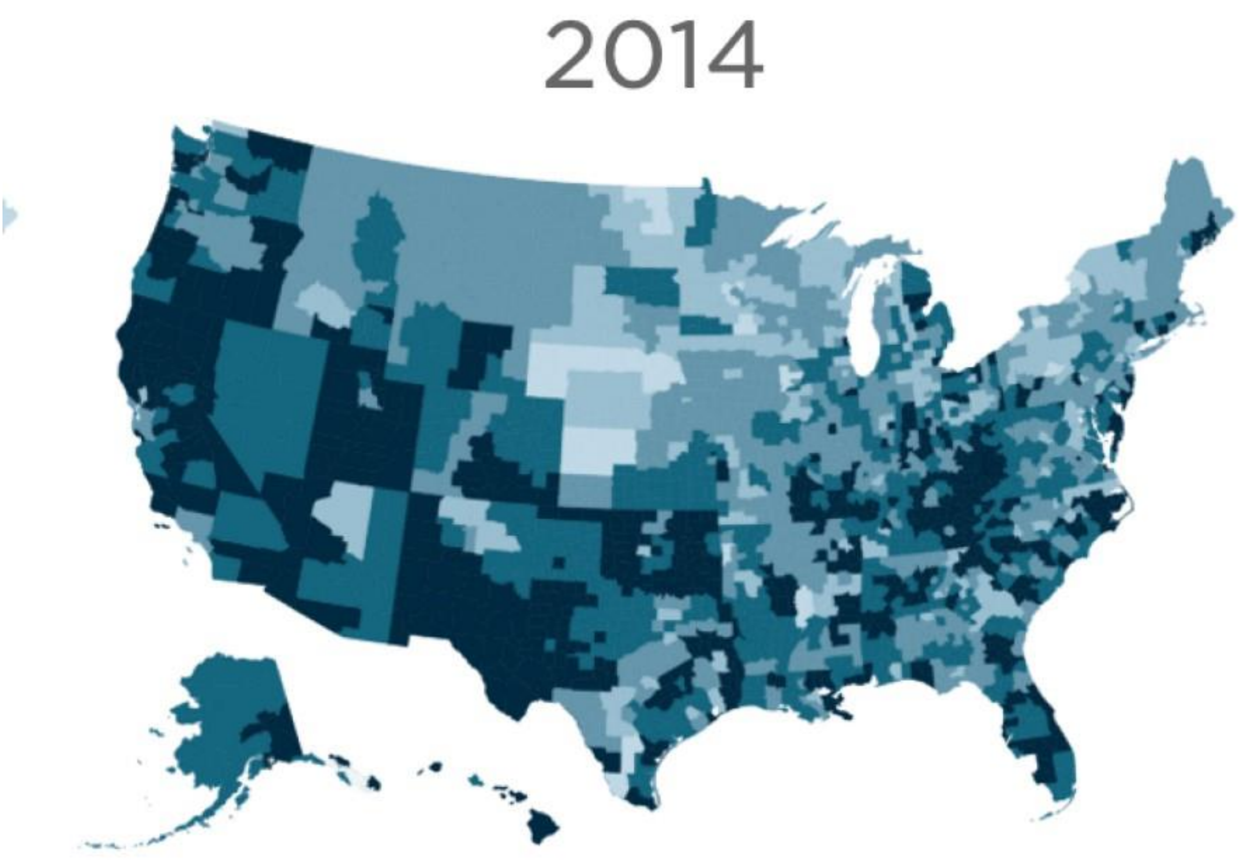
Its about drawing cool images



Visualization: Why

Pitfalls in data visualization

Aesthetic features like colors do not mater



Limited number of courses
on data visualization

Important to figure out what and
How to learn

Visualization: Why

Data visualization

What to Learn

Syntax of plotting libraries	Syntax is not hard to learn, or goggle many small details can be easily found
Understanding plot types and usages	Most crucial to develop a structured approach to Thinking about plots, when to use which plots and why
Aesthetics of plots	Clever people have been thinking about this, good libraries have aesthetic default choices
Communicating with plots	Very important to use plots as an element in communication (research paper, presentation, or blog)
Our focus	Understanding, plot types from first principles and then seeing examples of their usage
Pay attention to	How to tradeoff readability Vs information Different plot elements and how do they compose Changing mindset from data centric to viewer centric

Visualization: Why

Data visualization

Packages:

Built on top of MatPlotLib Abstracts many underlaying details

Inspired by Matlab plotting and OOP syntax

SEABORN

MATPLOTLIB

PLOTLY

Relation to pandas:

**There is a deep integration of these packages
With Pandas.
Hence, we will be learning new ideas in Padas also**

SEABORN

MATPLOTLIB

PANDAS

PRIME INTUIT
Finishing School

Visualization: working with complex JSON files

Tabulation

Import numpy as np
Import pandas as pd

Import matplotlib.pyplot as plt
Import seaborn as sns

https://api.covid19india.org/states_daily.json

!wget https://api.covid19india.org/states_daily.json

PRIME INTUIT
Finishing School

Visualization: working with complex JSON files

Tabulation

```
Import json
With open('data.json') as f:
Data = json.load(f)
Data

Data = data['state_daily']                    # dictionary
Cd = pd.json_normalization(data)

Cd
```

PRIME INTUIT
Finishing School

Visualization: working with complex JSON files

Tabulation

```
cd.date = pd to_datetime(cd.date)
Cd = cd[cd.status == ‘confirmed’]

Cd.drop(‘status’, axis = 1, inplace = True)

cd.set_index(‘date’, inplace = True)

Cd

cd.Info()
cd.tn
Pd.to_numeric(df.tn)
Cd = cd.apply(pd.to_numeric)
cd.info()
```

Visualization: working with complex JSON files

Tabulation

```
cd.info()

cd.tail(7)
```

Visualization: working with complex JSON files

Styling Tabulation

```
Cd = cd.tail(7)

cd.Style

Def colour_red_negative(x):
      color = ‘red’ if x < 0 else ‘white’
      return ‘colour: ‘  +  color

cd.style.applymap(colour_red_negative)

cd.drop(un, axis = 1, inplace = True)

cd.style
```

PRIME INTUIT
Finishing School

Visualization: working with complex JSON files

Styling Tabulation

```
cd.style.highlight_max(color = ‘red’)

cd.drop([‘dd’, ‘id’], axis = 1, inplace = True)

cd.style.highlight_max(color = ‘red’).highlight_min(color = ‘green’)

cd.drop(‘tt’, axis = 1, inplace = True)

Def bold_max_value(x):
    is max = (x == x.max())
    return [‘font-weight: bold’ if y else ‘ ‘ for y in is_max]

cd.style.apply(bold_max_value)
```

Visualization: working with complex JSON files

Styling Tabulation

```
cd.style.apply(bold_max_value). highlight_min(color = ‘green’)

cd.style.apply(bold_max_value). highlight_min(color = ‘red’, axis = 1)

cd.style.background_gradient(cmap = ‘Reds’)

cd.style.background_gradient(cmap = ‘Reds’, axis =1)

cd.style.background_gradient(cmap = ‘Reds’, subset = [‘mh’, ‘tn’, ‘dl’])
```

Visualization: working with complex JSON files

Styling Tabulation

cd.style.bar()

cd.style. bar(subset = [‘mh’, ‘tn’, ‘dl’])

Cd[[‘mh’, ‘tn’, ‘dl’]].style.bar(subset = [‘mh’], color = ‘red’). bar(subset = [‘tn’], color = ‘orange’). bar(subset = [‘dl’], color = ‘yellow’)

PRIME INTUIT
Finishing School

Visualization: Histograms

Visualization: Histograms

Distribution of data

```
X = np.random.normal(size = 1000)

Sns.distplot(x)

Sns.distplot(x);                        # suppress the warning

Sns.distplot(x, kde = False);

Sns.distplot(x, kde = False, rug = True);

Sns.set(color_codes = True)
Sns.distplot(x, kde = False, rug = True);
```

PRIME INTUIT
Finishing School

Visualization: Histograms

Distribution of data

```
Sns.distplot(x, kde = False, rug = True, bins = 20);

Sns.distplot(x, kde = False, rug = True, bins = 50);

Sns.kdeplot(x);                    # only the outline

Sns.kdeplot(x, shade = True);


# super impose  2 plots together
Y = np.random.uniform(size = 1000)
Sns.kdeplot(x, shade = True)
Sns.kdeplot(y, shade = True);
```

Visualization: Histograms

Distribution of data

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

d.info()
Sns.distplot(d.carot);


Sns.distplot(d.price);


Sns.distplot(d.x);
Sns.distplot(d.x, rug = True);
```

Visualization: Histograms

Distribution of data

```
Sns.distplot(d.sample(1000).x, rug = True);

Sns.distplot(d.sample(1000).x, rug = True, bins = 30);


Sns.kdeplot(d.x)
Sns.kdeplot(d.y)
Sns.kdeplot(d.z);

Sns.kdeplot(d.x, shade = True)
Sns.kdeplot(d.y, shade = True)
Sns.kdeplot(d.z, shade = True);
```

PRIME INTUIT
Finishing School

Visualization: Box plots

Visualization: Box Plots

 # Distribution of data

```
X = np.random.normal(size = 1000)

Sns.boxplot(x)


y = np.random.uniform(size = 1000)

Sns.boxplot(y);


Sns.boxplot(y, whis = 0.5);

Sns.boxplot(x, whis = 0.5);
Sns.boxplot(x, whis = 0.5, fliersize = 1);
```

PRIME INTUIT
Finishing School

Visualization: Box Plots

Distribution of data

```
Sns.boxplot(x, whis = 0.5, fliersize = 1, orient = ‘v’);

Sns.boxplot(d.price);

Sns.kdeplot(d.price);

Sns.boxplot(d.x);

Sns.kdeplot(d.x);

Sns.distplot(d.x);
```

Visualization: Box Plots

Distribution of data

Sns.boxplot(d.carat);

PRIME INTUIT
Finishing School

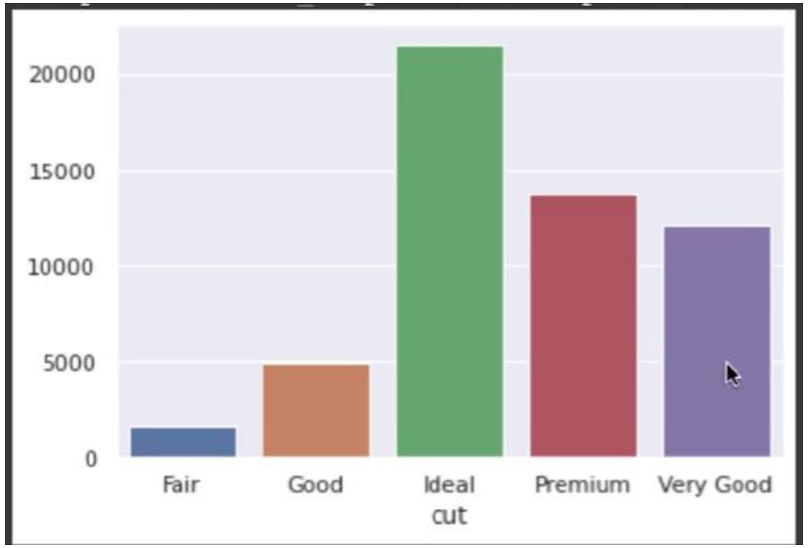
Visualization: Distribution of categorical variables

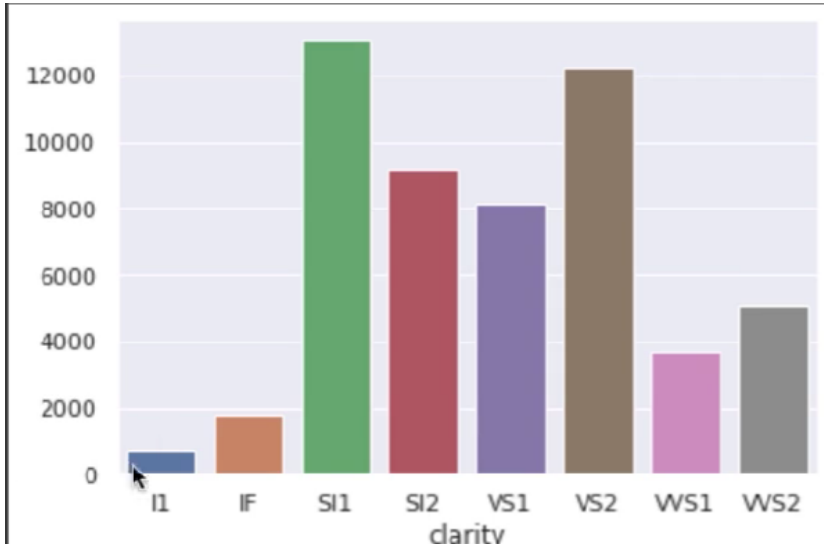
Visualization: Distribution of categorical variables

Distribution of categorical variables

D

```
d.groupby('cut').count()

d.goupby('cut')['cut'].count()

C = d.goupby('cut')['cut'].count()
Sns.barplot( x= c.index,  y = c.values)

C = d.goupby('clarity')['clarity'].count()
Sns.barplot( x= c.index,  y = c.values);

C = d.goupby('color')['color'].count()
Sns.barplot( x= c.index,  y = c.values);
```





Visualization: Joint distribution of 2 variables

PRIME INTUIT
Finishing School

Visualization: Joint Distribution of 2 variables

joint Distribution of 2 variables

X = np.random.normal(size, 1000)
y = np.random.normal(size, 1000)

Df = pd.DataFrame({'x': x, 'y': y})

~~Sns.jointplot(df.x, df.y)~~
Or
Sns.jointplot('x', 'y', data = df);

Sns.jointplot('x', 'y', data = df, kind = 'kde');

PRIME INTUIT
Finishing School

Visualization: Joint Distribution of 2 variables

joint Distribution of 2 variables

```
X = np.random.normal(size, 1000)
y = 3 * x +np.random.normal(size, 1000)/ 5

Df = pd.DataFrame({'x': x, 'y': y})

Sns.jointplot('x', 'y', data = df, kind = 'kde');


# As you increase x the distribution of y changes significantly
```

Finishing School

Visualization: Joint Distribution of 2 variables

joint Distribution of 2 variables

plot the diamonds data set

Sns.jointplot(‘carat’, ‘price’, data = d, kind = ‘kde’);

As you increase carat the price of the diamond changes significantly

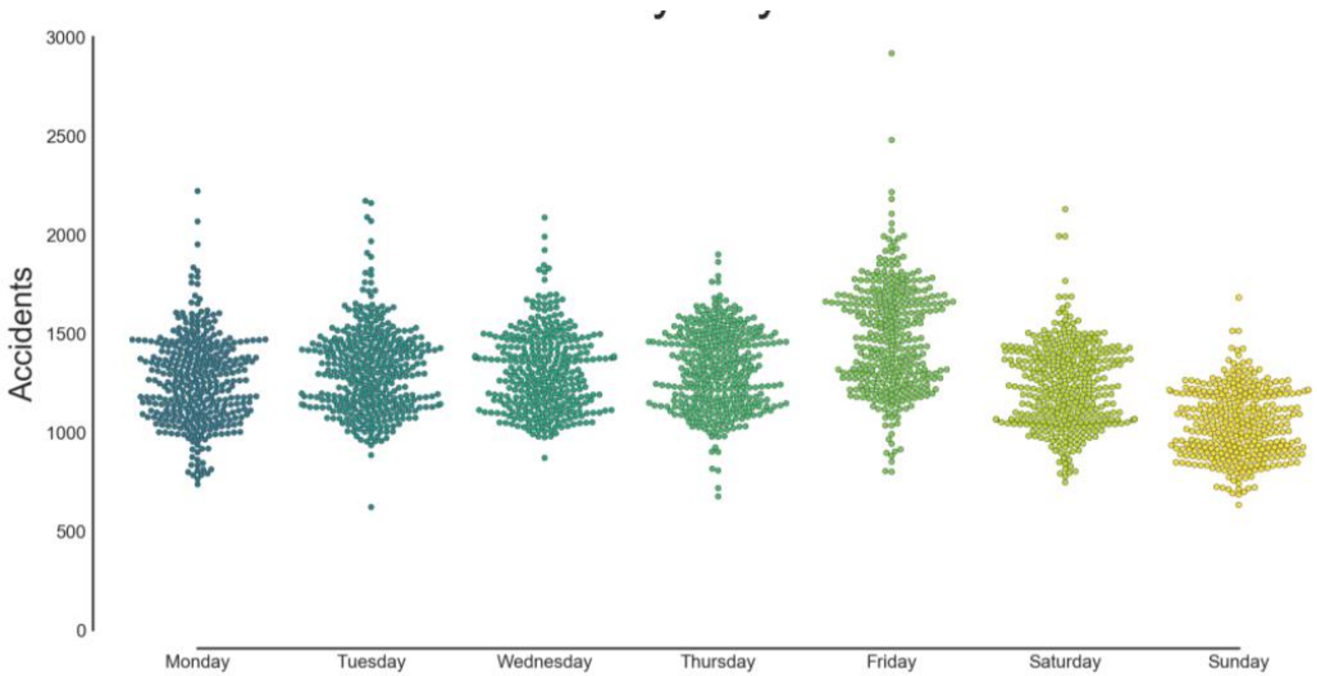
Sns.jointplot(‘carat’, ‘price’, data = d.sample(500));

Sns.jointplot(‘depth’, ‘price’, data = d.sample(500));

Sns.jointplot(‘x’, ‘price’, data = d.sample(500));

Sns.jointplot(‘x’, ‘price’, data = d.sample(500), kind = ‘kde’);

Visualization: Swarm Plot



Visualization: Swarm Plot

Swarm Plot

```
# plot the diamonds data set

Sns.swarmplot(d.carat)

Sns.swarmplot(d.head(1000).carat)

Sns.swarmplot(d.head(1000).price)

Sns.swarmplot(d.sample(1000).price)

Sns.swarmplot(d.sample(1000).carat)
```

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

Multiple Swarm Plots

```
# plot the diamonds data set

d.info()
Sns.swarmplot(x = 'cut', y = 'price' data = d.sample(1000));


Sns.swarmplot(x = 'color', y = 'price' data = d.sample(1000));


Sns.swarmplot(x = 'clarity', y = 'price' data = d.sample(1000));
```

Visualization: Swarm Plot

 # Multiple Swarm Plots

 # penguin data set

P = sns.load_dataset(‘penguins’)

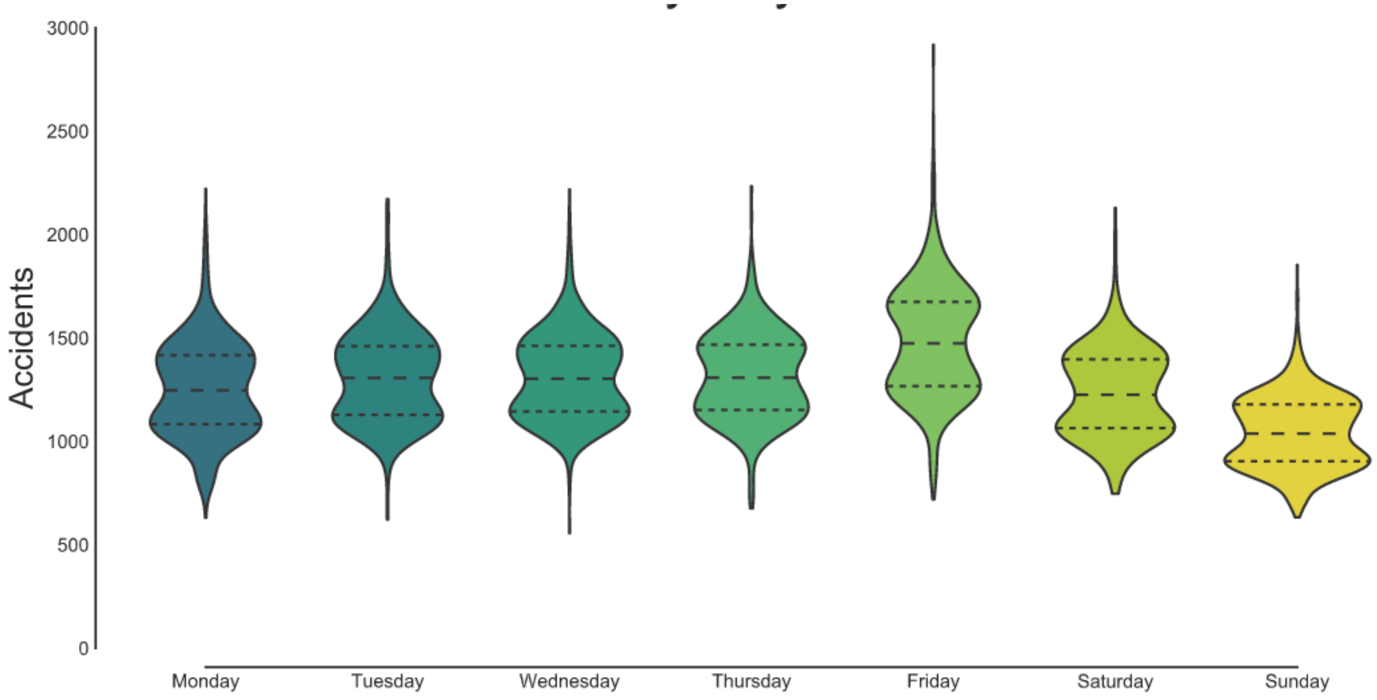
P

Sns.swarmplot(x = ‘species’, y = ‘body_mass_g’, data = p);

Sns.swarmplot(x = ‘island’, y = ‘body_mass_g’, data = p);

Sns.swarmplot(x = ‘body_mass_g’, data = p);

Visualization: Violin Plot



Visualization: Swarm Plot

Single violin plot

penguin data set

P = sns.load_dataset(‘penguins’)

P

Sns.violineplot(x = ‘body_mass_g’, data = p);

median, iqr, whiskers, kde

Sns.swarmplot(x = ‘body_mass_g’, data = p);
Sns.violinplot(x = ‘body_mass_g’, data = p);
Sns.boxplot(x = ‘body_mass_g’, data = p);
Sns.kdeplot(p.body_mass_g, shade = True);

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

 # Multiple violin plot

```
# plot together

Fig, axs = plt.subplots(nrows =4)
Sns.swarmplot(x = ‘body_mass_g’, data = p, ax = axs[0]);
Sns.violinplot(x = ‘body_mass_g’, data = p, ax = axs[1]);
Sns.boxplot(x = ‘body_mass_g’, data = p, ax = axs[2]);
Sns.kdeplot(p.body_mass_g, shade = True, ax = axs[3]);

Fig.set_size_inches(5, 10)
```

Visualization: Swarm Plot

 # multiple violin plot

```
# plot together

Fig, axs = plt.subplots(nrows =4)
Fig.set_size_inches(5, 10)

P1 = Sns.swarmplot(x = ‘body_mass_g’, data = p, ax = axs[0]);
P1.set(xlim = (2000, 7500));
P2 = Sns.violinplot(x = ‘body_mass_g’, data = p, ax = axs[1]);
P2.set(xlim = (2000, 7500));
P3 = Sns.boxplot(x = ‘body_mass_g’, data = p, ax = axs[2]);
P3.set(xlim = (2000, 7500));
P4 = Sns.kdeplot(p.body_mass_g, shade = True, ax = axs[3]);
P4.set(xlim = (2000, 7500));
```

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

Multiple violin plot

plot together

Sns.violinplot(x='body_mass_g', data = p);
Sns.violinplot(y='body_mass_g', data = p);

why did we plot it vertically..

Sns.violinplot(x = 'species', y='body_mass_g', data = p, orient = 'v');
Sns.violinplot(x = 'species', y='body_mass_g', data = p); # no need of orient

Sns.violinplot(x = 'species', y='flipper_length_mm', data = p);
Sns.violinplot(x = 'island', y='flipper_length_mm', data = p);
Sns.violinplot(x = 'sex', y='flipper_length_mm', data = p);

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

Multiple violin plot

 # plot together

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p, hue = ‘island’);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p, hue = ‘sex’);

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p, hue = ‘species’);
 # diamond data
Sns.swarmplot(x = ‘cut’, y=‘price’, data = d.sample(10000));
Sns.swarmplot(x = ‘cut’, y=‘price’, data = d,sample(1000), hue = ‘color’);

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

 # paired violin plot

 # plot together

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p, hue = ‘sex’);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p[p.sex]);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p[p.sex == ‘Male’]);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, data = p[p.sex==‘Female’]);

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, hue = ‘sex’, split = True, data = p);
Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, hue = ‘sex’, split = True, inner = ‘quartile’, data = p);

PRIME INTUIT
Finishing School

Visualization: Swarm Plot

paired violin plot

plot together

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’, hue = ‘species’, split = True, inner = ‘quartile’, data = p);

Sns.swarmplot(x = ‘island’, y=‘flipper_length_mm’,hue = ‘species’ inner = ‘quartile’, data = p);

P[‘binary_species’] = p.species.apply(lambda x: 0 if x == ‘Gentoo’ else = 1)

P

Sns.violinplot(x = ‘island’, y=‘flipper_length_mm’, hue = ‘binary_species’, split = True, data = p);

Visualization: Swarm Plot

 # paired violin plot

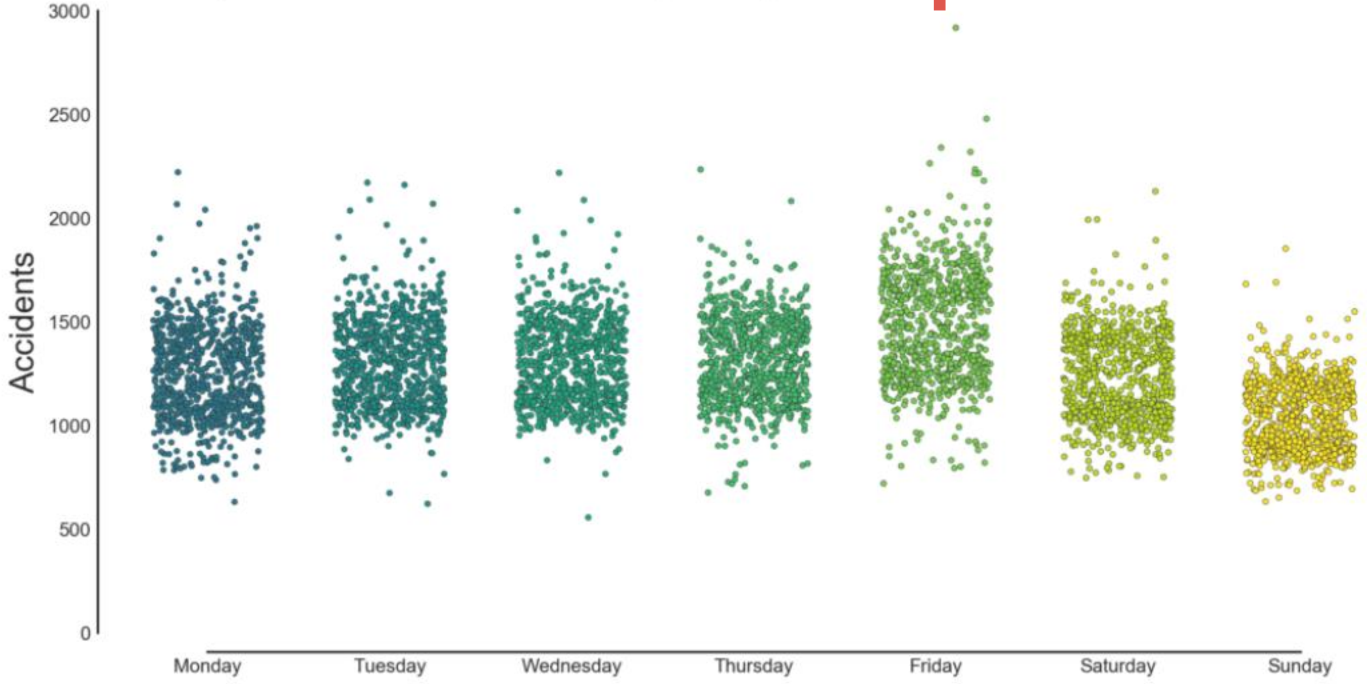
 # plot together

P[‘binary_species’] = p.species.apply(lambda x: ‘Gentoo’ if x == ‘Gentoo’ else = ‘Adelie | Chinstrap’)

P

Sns.violinplot(x = ‘island’, y=‘flipper_length_mm’, hue = ‘binary_species’, split = True, data = p);

Visualization: Strip Plot



PRIME INTUIT
Finishing School

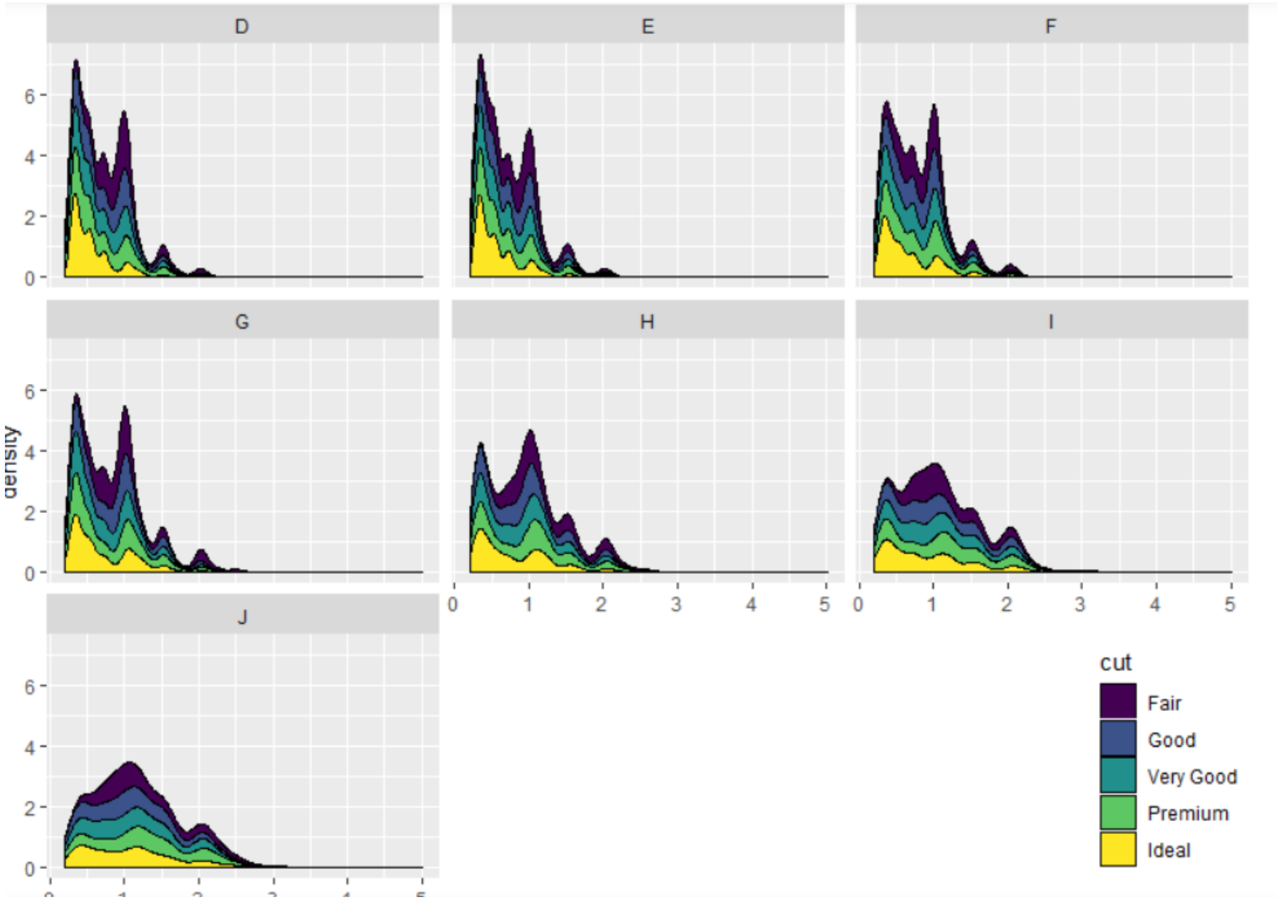
Visualization: Strip Plot

```
# tips data set

tips = sns.load_dataset(‘tips’)

tips
Sns.stripplot(x = tips[‘total_bill’]);

Sns.stripplot(x = ‘day’, y =‘total_bill’ data = tips, jitter = True);

sns.stripplot(x="total_bill", y="day", data=tips)

sns.stripplot(x="total_bill", y="day", data=tips, linewidth=1)
```

Visualization: Strip Plot

```
sns.stripplot(x="sex", y="total_bill", hue="day", data=tips)

sns.stripplot(x="day", y="total_bill", hue="smoker",data=tips,
palette="Set2", dodge=True)

sns.stripplot(x="time", y="tip", data=tips,order=["Dinner", "Lunch"]

sns.violinplot(x="day", y="total_bill", data=tips,inner=None, color=".8")
sns.stripplot(x="day", y="total_bill", data=tips)
```

Visualization: Faceted Plot



Visualization: Faceted Plot

```
Sns.kdeplot(p.flipper_length_mm, shade = True);

Sns.kdeplot(p[p.species ==‘Gentoo’].flipper_length_mm, shade = True);
Sns.kdeplot(p[p.species ==‘Adelie’].flipper_length_mm, shade = True);
Sns.kdeplot(p[p.species ==‘Chinstrap’].flipper_length_mm, shade = True);


Sns.kdeplot(p[p.species ==‘Gentoo’].flipper_length_mm, shade = True);
Sns.kdeplot(p[p.species ==‘Adelie’].flipper_length_mm, shade = True);
Sns.kdeplot(p[p.species ==‘Chinstrap’].flipper_length_mm, shade = True);
Plt.legend(title = ‘Species’, labels = [‘Gentoo’, ‘Adelie’, ‘Chinstrap’]);
```

PRIME INTUIT
Finishing School

Visualization: Faceted Plot

```
Sns.boxplot(p[p.species ==‘Gentoo’].flipper_length_mm);
Sns.boxplot(p[p.species ==‘Adelie’].flipper_length_mm);
Sns.boxplot(p[p.species ==‘Chinstrap’].flipper_length_mm);
Plt.legend(title = ‘Species’, labels = [‘Gentoo’, ‘Adelie’, ‘Chinstrap’]);
```

```
Fig, axs = plt.subplots(nrows = 3);
Sns.boxplot(p[p.species ==‘Gentoo’].flipper_length_mm, ax = axs[0]);
Sns.boxplot(p[p.species ==‘Adelie’].flipper_length_mm , ax = axs[1]);
Sns.boxplot(p[p.species ==‘Chinstrap’].flipper_length_mm, ax = axs[2);
#Plt.legend(title = ‘Species’, labels = [‘Gentoo’, ‘Adelie’, ‘Chinstrap’]);
```

Visualization: Faceted Plot

```
Fig, axs = plt.subplots(nrows = 3);
Sns.boxplot(p[p.species ==‘Gentoo’].flipper_length_mm, ax = axs[0]);
Sns.boxplot(p[p.species ==‘Adelie’].flipper_length_mm , ax = axs[1]);
Sns.boxplot(p[p.species ==‘Chinstrap’].flipper_length_mm, ax = axs[2]);
Plt.tight_layout()
#Plt.legend(title = ‘Species’, labels = [‘Gentoo’, ‘Adelie’, ‘Chinstrap’]);
```

```
Column_name = ‘species’
Nrows = len(p[column_name].unique())
Fig, axs = plt.subplots(nrows = nrows);
I = 0
For c_v in p[column_name].unique():
    sns.kdeplot(p[p[column_name] == c_v].flipper_length_mm, shade =
    True, ax = axs[i]);
    I += I
```

Visualization: Faceted Plot

```
# more efficient way

G = sns.FacetGrid( p, row = 'species');
g.map(sns.kdeplot, 'flipper_length_mm', shade = True);


G = sns.FacetGrid( p, col = 'species');
g.map(sns.kdeplot, 'flipper_length_mm', shade = True);

G = sns.FacetGrid( p, col = 'island');
g.map(sns.kdeplot, 'flipper_length_mm', shade = True);

G = sns.FacetGrid( p, col = 'island');
g.map(sns.distplot, 'flipper_length_mm');
```

Visualization: Faceted Plot

```
# more efficient way ( Rows and Col)

G = sns.FacetGrid( p, col = 'island', row = 'sex');
g.map(sns.distplot, 'flipper_length_mm');


G = sns.FacetGrid( p, col = 'island', row = 'sex');
g.map(sns.kdeplot, 'flipper_length_mm', shade = True);

G = sns.FacetGrid( p, col = 'island', row = 'sex');
g.map(sns.violinplot, 'flipper_length_mm');
```

Visualization: Pair Plot

PRIME INTUIT
Finishing School

Visualization: Pair Plot

```
# Pair plots
Sns.joinplot(p.body_mass_g, p.flipper_length_mm);

Sns.joinplot(p.body_mass_g, p.culmen_depth_mm);

Sns.pairplot(p);

Sns.pairplot(p, hue = ‘sex’);

Sns.pairplot(p, hue = ‘species’);

Sns.pairplot(d.sample(1000));

Sns.pairplot(d.sample(1000), hue = ‘cut’);

Sns.pairplot(d.sample(1000), hue = ‘cut’, corner = True);
```

Visualization: Boxen Plot

Visualization: Boxen Plot

```
# Boxen plots

X = np.random.normal(size = 1000)

Sns.boxplot(x);

Sns.kdeplot(x);

Sns.boxplot(d.sample(5000). Carat);

Sns.boxenplot(d.sample(5000).carat);

Sns.boxenplot( x = ‘island’, y = ‘body_mass_g’, data = p)
```

Recap

PRIME INTUIT
Finishing School

Visualization: Recap

Tabulation

Tabulation

Styling tabulation

Distribution of single continuous variable

histogram

box plot

boxen plot

Distribution of categorical variable

bar plots

Join distribution of two variables

join plots

swarm plots

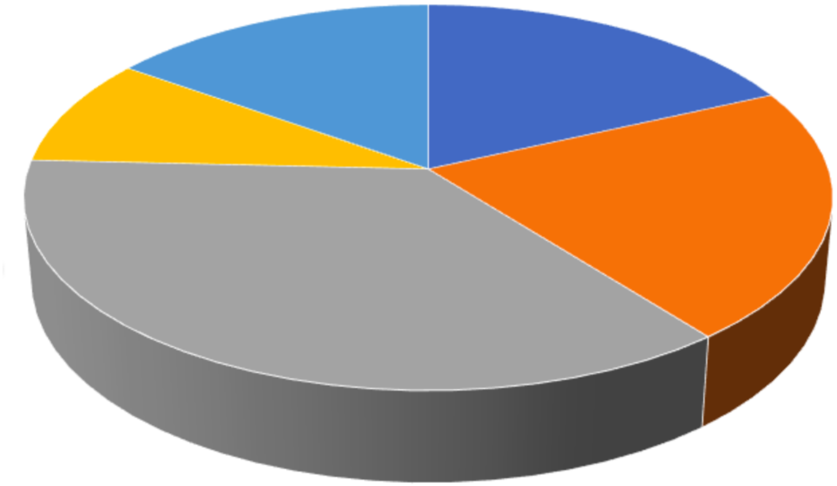
violin plots

strip plots

faceted plots

pair plots

Visualization: Pie Chart



Visualization: Pie Chart

Plotting the composition of data
Static Composition
Dynamic composition

P = sns.load_dataset('penguins')

p.head(20)

p.groupby('species').count()

p.groupby('species')['species'].count()

Z = p.groupby('species')['species'].count()

Plt.pie(z);
Plt.show()

Visualization: Pie Chart

```
# Plotting the composition of data

Plt.pie(z, labels = z.index);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f%%);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f%%, explode = [0, 1, 0]);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f%%, explode = [0, 1, 0], startangle = 180);
Plt.show()
```

Visualization: Pie Chart

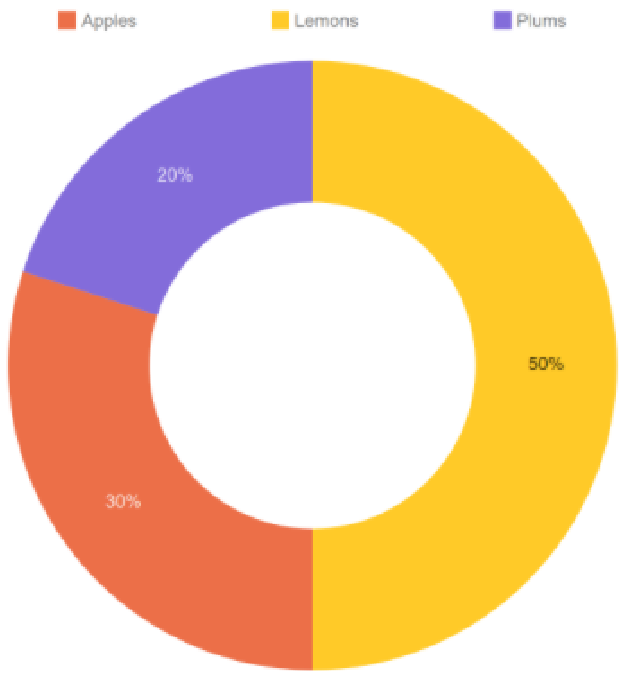
Plotting the composition of data

Plt.pie(z, labels = z.index, autopct = %.2f%%, explode = [0, 1, 0], startangle = 180, shadow = True);
Plt.show()

- Association of sectors to % or numbers
- Consumes lot of space
- Notion of representing the entire data into circle and cutting parts

Plt.pie(np.random.randint(0, 10, 10);
Plt.show()

Visualization: Donut Chart



Visualization: Pie Chart

```
# Plotting the composition of data

Plt.pie(np.random.randint(0, 10, 10), wedgeprops = dict(width =0.3));
Plt.show()

Cmap = plt.get_cmap(‘pastel1’)
My_colours = cmap(np.arange(10))
Plt.pie(np.random.randint(0, 10, 10, wedgeprops = dict(width =0.3),
colors = my_colours);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f%%, wedgeprops = dict(width
=0.3));
Plt.show()
```

Visualization: Pie Chart

```
# Plotting the composition of data

Plt.pie(np.random.randint(0, 10, 10, wedgeprops = dict(width =0.3));
Plt.show()

https://matplotlib.org/stable/tutorials/colors/colormaps.html
Cmap = plt.get_cmap(‘pastel1’)
My_colours = cmap(np.arange(10))
Plt.pie(np.random.randint(0, 10, 10, wedgeprops = dict(width =0.3),
colors = my_colours);
Plt.show()

Plt.pie(z, labels = z.index, autopct = %.2f%%, wedgeprops = dict(width
=0.3));
Plt.show()
```

Visualization: Pie Chart

Plotting the composition of data

y = p.groupby(‘island’)[‘island’].count()

Plt.pie(y, labels = y.index, autopct = %.2f%%, wedgeprops = dict(width =0.3));
Plt.show()

bringing in the 3rd dimension

Pd.crosstab(p.species, p.island)
X = Pd.crosstab(p.species, p.island)
X = x.T

Visualization: Pie Chart

 # Plotting the composition of data (Nested plots)

```
Plt.pie(x.sum(axis = 1), labels = x.index, radius =1, wedgeprops =
dict(width =0.3));
Plt.show()
```

```
Plt.pie(x.sum(axis = 1), labels = x.index, radius =1, wedgeprops =
dict(width =0.3));
Plt.pie(x.values.flatten(), radius = .7, wedgeprops = dict(width = 0.2));
Plt.show()
```

Visualization: Pie Chart

```
# Plotting the composition of data

Cmap = plt.get_cmap('tab20c')
Outer_colors = cmap(np.array([0, 4, 8]))
Inner_colors = cmap(np.array([1, 2, 3, 5, 6,7, 9, 10, 11]))


Plt.pie(x.sum(axis = 1), labels = x.index, radius =1, wedgeprops =
dict(width =0.3), colors = outer_colors);
Plt.pie(x.values.flatten(), radius = .7, wedgeprops = dict(width = 0.2),
colors = inner_colors);
Plt.show()
```

Visualization: Pie Chart

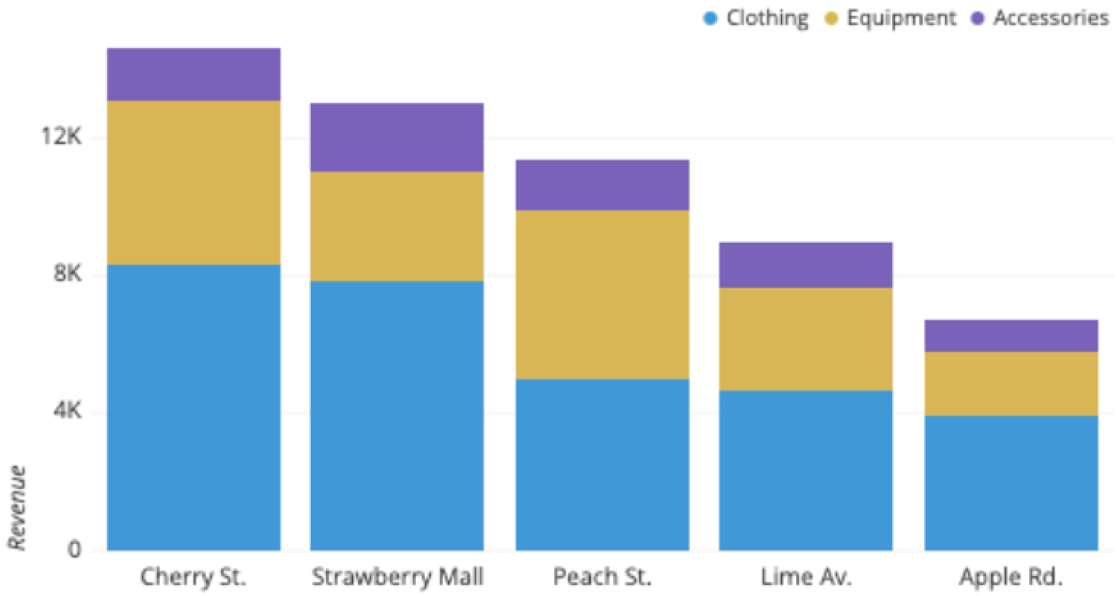
```
# Plotting the composition of data

Cmap = plt.get_cmap('tab20c')
Outer_colors = cmap(np.array([0, 4, 8]))
Inner_colors = cmap(np.array([1, 2, 3, 5, 6,7, 9, 10, 11]))


Plt.pie(x.sum(axis = 1), labels = x.index, radius =1, wedgeprops =
dict(width =0.3), colors = outer_colors);
Plt.pie(x.values.flatten(), radius = .7, wedgeprops = dict(width = 0.2),
labels =['A', 'C', 'G', 'A', 'C', 'G', 'A', 'C', 'G' ], colors = inner_colors);
Plt.show()
```

Visualization: Pie Chart

 # Plotting the composition of data

Plt.pie(x.sum(axis = 1), labels = x.index, radius =1, wedgeprops = dict(width =0.3), colors = outer_colors);
Plt.pie(x.values.flatten(), radius = .7, wedgeprops = dict(width = 0.2), labels =['A', ' ', 'G', 'A', 'C', ' ', 'A', ' ', ' '], colors = inner_colors, labeldistance = 0.75, textprops = dict(color = 'w'));
Plt.show()

Visualization: Stacked bar Plot



Visualization: Stacked bar plots

Stacked bar plots

url = ‘https://api.covid19india.org/states_daily.json’

Import urllib.request

Urllib.request.urlretrieve(url, ‘data.json’

check the file in the default folder

Covid_data = pd.read_json(‘data.json’)

Covid_data

Visualization: Stacked bar plots

Stacked Bar plots

```
Import json
With open('data.json') as f:
Data = json.load(f)
Data

Data = data['state_daily']                    # dictionary
Cd = pd.json_normalization(data)

Cd
```

Visualization: Stacked bar plots

```
# Stacked bar plts

Cd_ = cd.tail(3)                              # consider last 3 days

Cd_.drop('date', axis = 1, inplace = True)    # Drop dates

Cd_.set_index('status', inplace = True)       # set status as index

Cd_ = cd.T                                    # Transpose to have 3 cols

Cd_ = cd_.apply(pd.to_numeric)                # Objects 2 Numbers

Cd_.drop('tt', inplace = True)                # Drop total tally col

Cd_.head(5)
```

PRIME INTUIT
Finishing School

Visualization: Stacked bar plots

```
# Stacked bar plts

Plt.bar(cd_.index, cd_.Confirmed);                    # math plot bar

Plt.xticks(rotation 90);                              # Rotate x - labels

# To add the next stack (Recovered)
Plt.bar(cd_.index, cd_.Confirmed);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed);
Plt.xticks(rotation 90);

# To add the next stack (Deceased)
Plt.bar(cd_.index, cd_.Confirmed);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed);
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered);
Plt.xticks(rotation 90);
```

Visualization: Stacked bar plots

Stacked bar plts

# Fixing issues, make the plot more visible
Fig = plt.gcf() # unlike ‘subplots’, ‘gcf’ only returns a figure
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.Confirmed);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed);
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered);
Plt.xticks(rotation 90);

# Changing the color
Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.Confirmed, color = ‘Orange’);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed, color = ‘Green’);
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered, color = ‘Red’);
Plt.xticks(rotation 90);

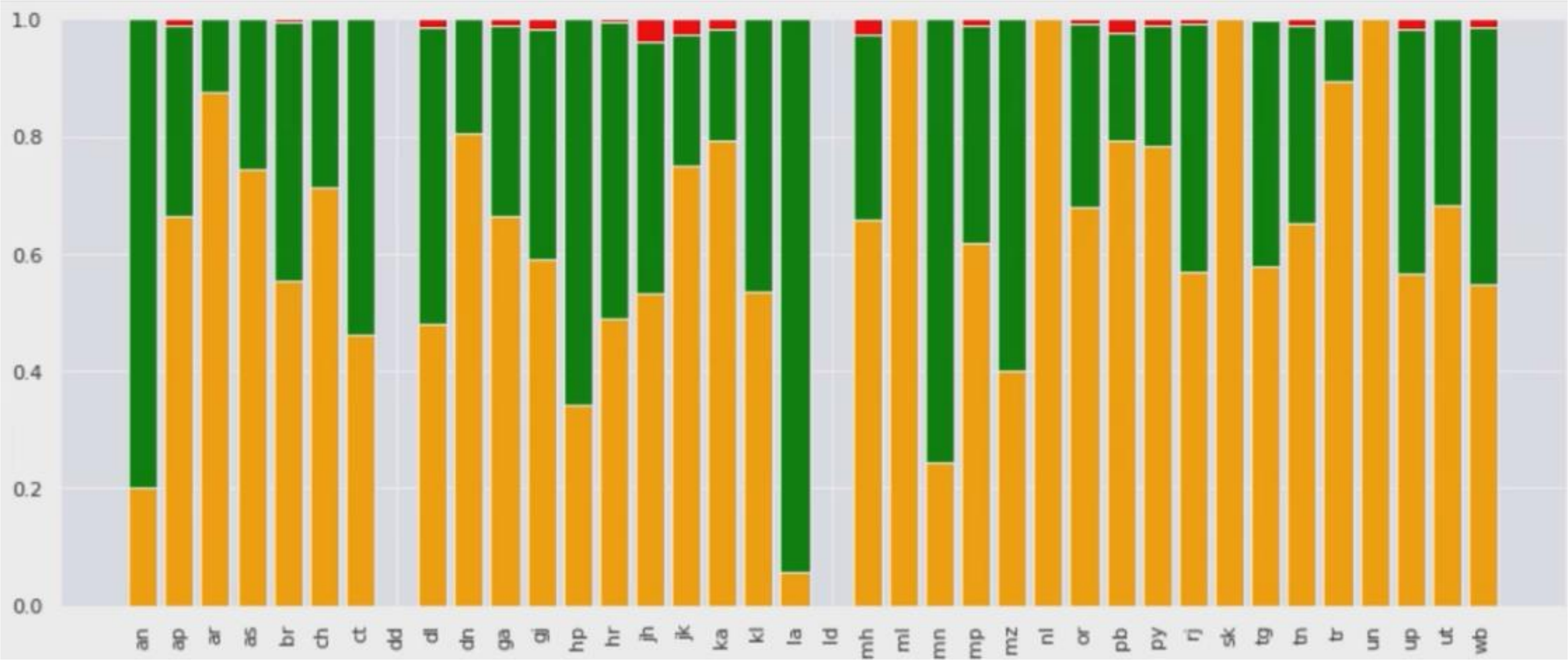
Visualization: Stacked bar plots

```
# Stacked bar plts

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.Confirmed, color = ‘Orange’);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed, color = ‘Green’);
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered, color = ‘Red’)
Plt.xticks(rotation 90);

# Calculate the total value and print it on top of the bar
For i, val in enumerate(cd_.index):      # enumerate func, returns the index and the value
    print(i, val)

    y = cd_.loc[val].sum()
    x = i
    plt.text(x,y, str(y));
```

Visualization: Stacked bar plots

```
# Stacked bar plts

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.Confirmed, color = ‘Orange’);
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed, color = ‘Green’);
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered, color = ‘Red’);
Plt.xticks(rotation 90);

# improve the annotation
For i, val in enumerate(cd_.index):      # enumerate func, returns the index and the value
    print(i, val)
    y = cd_.loc[val].sum() + 100
    if y > 1000:
        x = i
        plt.text(x,y, str(y), ha = “center”);
```

Visualization: Relative Stacked bar Plot



Visualization: Relative Stacked bar plots

 # Relative Stacked bar plts

Cd_.head(5)

 # add the additional required columns to previous data set on Covid.

```
Cd_['Total'] = cd_.sum(axis = 1)
cd_.head(5)
Cd_['ConfirmedFraction'] = cd_['Confirmed'] / cd_['Total']
Cd_.head(5)
Cd_['RecoveredFraction'] = cd_['Recovered'] / cd_['Total']
Cd_['DeceasedFraction'] = cd_['Deceased'] / cd_['Total']
Cd_.head(5)
```

Visualization: Relative Stacked bar plots

```
# Relative Stacked bar plts

# Ploting the graph.

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.ConfirmedFraction, color = 'Orange');
Plt.bar(cd_.index, cd_RecoveredFraction, bottom = cd_.ConfirmedFraction, color = 'Green');
Plt.bar(cd_.index, cd_DeceasedFraction, bottom = cd_.ConfirmedFraction + cd_.RecoveredFraction, color = 'Red');
Plt.xticks(rotation 90);
```

PRIME INTUIT
Finishing School

Visualization: Relative Stacked bar plots

```
# Relative Stacked bar plts

# improving the plot, .
Cd_ = cd_.sort_values('CofirmedFraction', ascending = False)
Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.ConfirmedFraction, color = 'Orange');
Plt.bar(cd_.index, cd_RecoveredFraction, bottom = cd_.ConfirmedFraction, color =
'Green');
Plt.bar(cd_.index, cd_DeceasedFraction, bottom = cd_.ConfirmedFraction +
cd_.RecoveredFraction, color = 'Red');
Plt.xticks(rotation 90);
```

PRIME INTUIT
Finishing School

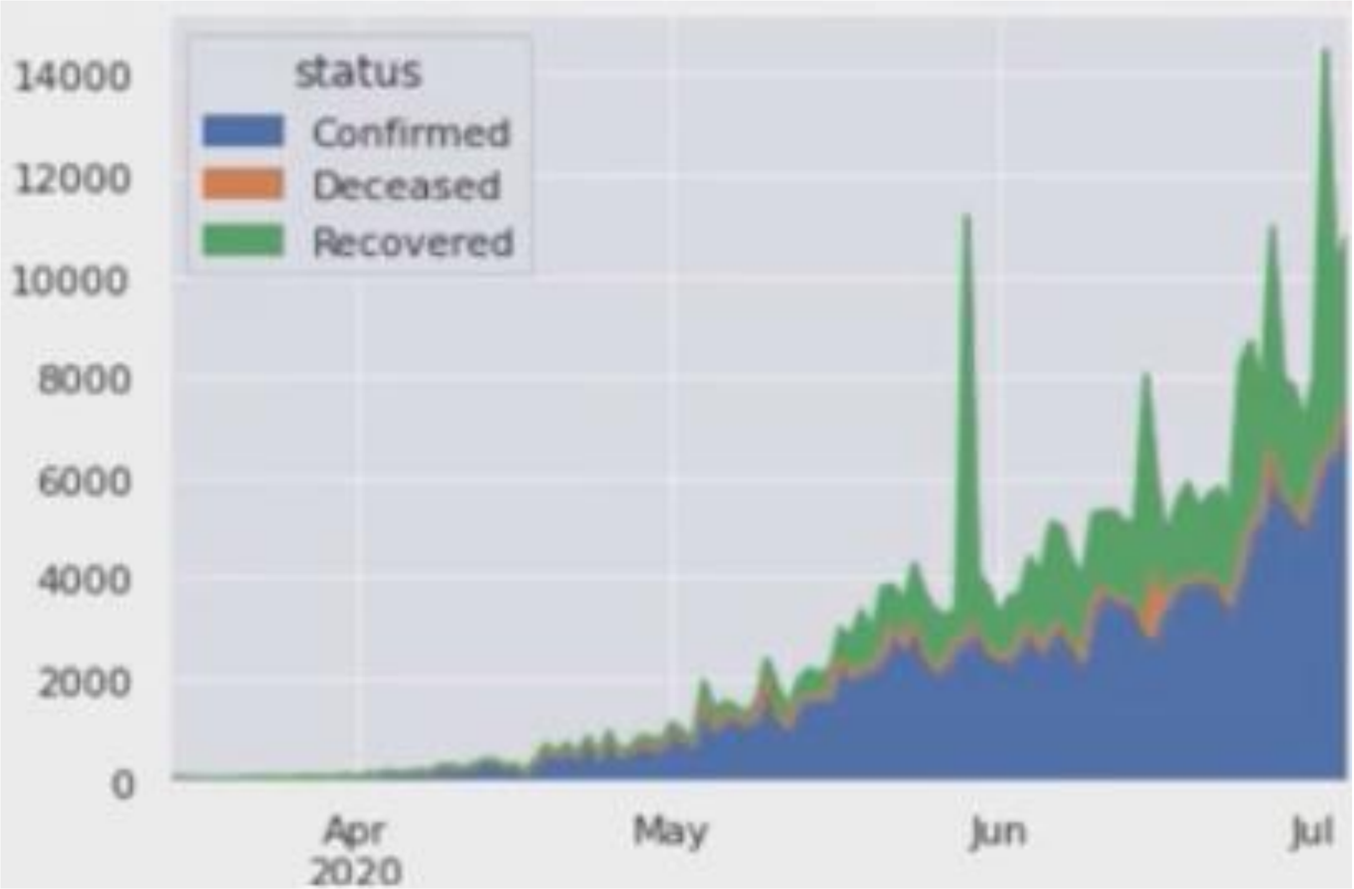
Visualization: Relative Stacked bar plots

```
# Relative Stacked bar plts

# improving the plot, sorting stackedbar chart.

Cd_ = cd_.sort_values('Total', ascending = False)

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.bar(cd_.index, cd_.Confirmed, color = 'Orange');
Plt.bar(cd_.index, cd_Recovered, bottom = cd_.Confirmed, color = 'Green');
Plt.bar(cd_.index, cd_Deceased, bottom = cd_.Confirmed + cd_.Recovered, color = 'Red');
Plt.xticks(rotation 90);
```

Visualization: Relative Stacked bar plots

```
# Relative Stacked bar plts

# improving the plot, Horizontal stacked bar chart.

Cd_ = cd_.sort_values('Total', ascending = False)

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Plt.barh(cd_.index, cd_.Confirmed, color = 'Orange');
Plt.barh(cd_.index, cd_Recovered, left = cd_.Confirmed, color = 'Green');
Plt.barh(cd_.index, cd_Deceased, left = cd_.Confirmed + cd_.Recovered, color = 'Red');
Plt.xticks(rotation 90);
```

Visualization: Stacked Area Plot



Visualization: Stacked Area Plots

```
# Time Varying compostion of data

cd.head()

Cd_mh = cd[['mh', 'data', 'status']]
Cd_mh.head()

Cd_mh['mh'] = pd.to_numeric(cd_mh['mh'])
Cd_mh['date'] = pd.to_datetime(cd_mh['date'])

Cd_mh.head()

## what if we wanted to represent data in a different looking table
Date |   Confirmed  |  Recovered  |  Deceased  |
2021 – 03 -14 | 14 | 0 | 0|
2021 – 03 -15 | 15 | 0 | 0|
```

Visualization: Stacked Area Plots

```
# Time Varying compostion of data

cd.head()

Cd_mh = cd_mh.pivot_table(values = ‘mh’, columns = ‘status’, index = ‘date’)

Cd_mh.head()
```

Visualization: Stacked Area Plots

```
# Stacked area Plot

Cd_mh.plot.area();                                        # Panda function

# using math plot lib
Plt.stackplot(cd_mh.index, cd_mh.confirmed, cd_mh.Recovered, cd_mh.Deceased);

# cleanup
Fig = plt.gcf();
Fig.set_size_inches(15, 6);
Plt.stackplot(cd_mh.index, cd_mh.confirmed, cd_mh.Recovered, cd_mh.Deceased);

# change colors
Plt.stackplot(cd_mh.index, cd_mh.confirmed, cd_mh.Recovered, cd_mh.Deceased, colors
= [‘orange’, ‘green’, ‘red’]);
```

PRIME INTUIT
Finishing School

Visualization: Stacked Area Plots

```
# Stacked area Plot

# Build a legend
Fig = plt.gcf();
Fig.set_size_inches(15, 6);
Plt.stackplot(cd_mh.index, cd_mh.confirmed, cd_mh.Recovered, cd_mh.Deceased, labels = ['Confirmed', 'Recovered', 'Deceased'], colors = ['orange', 'green', 'red']);
Plt.legend();
```

PRIME INTUIT
Finishing School

Visualization: Stacked Area Plots

 # Relative Stacked area Plot

 # Build a legend
Fig = plt.gcf();
Fig.set_size_inches(15, 6);
Plt.stackplot(cd_mh.index, cd_mh.confirmed / cd_mh. Sum(axis = 1), cd_mh.Recovered / cd_mh. Sum(axis = 1), cd_mh.Deceased / cd_mh. Sum(axis = 1), labels = [‘Confirmed’, ‘Recovered’, ‘Deceased’], colors = [‘orange’, ‘green’, ‘red’]);
Plt.legend();

Visualization: Stacked Area Plots

```
# Function to plot Stacked area Plot

Def plot_stk_area_state(state):
Cd_xx = cd[[state, 'data', 'status']]
Cd_xx[state] = pd.to_numeric(cd_xx[state])
Cd_xx['date'] = pd.to_datetime(cd_xx['date'])
Cd_xx = cd_xx.pivot_table(values = state, columns = 'status', index = 'date')

Fig = plt.gcf();
Fig.set_size_inches(15, 6);
Plt.stackplot(cd_mh.index, cd_xx.confirmed / cd_xx. Sum(axis = 1), cd_xx.Recovered /
cd_xx. Sum(axis = 1), cd_xx.Deceased / cd_xx. Sum(axis = 1),  labels = ['Confirmed',
'Recovered', 'Deceased'], colors = ['orange', 'green', 'red']);
Plt.legend();

plot_stk_area_state('ka')
```

Visualization: Scatter Plots

PRIME INTUIT
Finishing School

Visualization: Scatter Plots (Recap….)

Recap

- Plotting Tabulated Data
  - ❖ Tabulation
  - ❖ Styling Tabulation
- Plotting distribution of data
  - ❖ Histogram
  - ❖ box plot
  - ❖ boxen plot
  - ❖ bar plots
  - ❖ join plots
  - ❖ swarm plots
  - ❖ faceted plots
  - ❖ pair plots
- Plotting Composition of data
  - ❖ Pie plots
  - ❖ Donut plots
  - ❖ Stacked bar plots
  - ❖ Relative stacked bar plots
  - ❖ Stacked area plots
- Plotting Relationships between data
  - ❖ Scatter Plots

PRIME INTUIT
Finishing School

Visualization: Scatter Plot

 # Function to plot Stacked area Plot

Import seaborn as sns

T = sns.load_dataset(‘tips’)
t.head()

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t);

T[‘tip_fraction’] = t[‘tip’] / t[‘total_bill’]

Sns.scatterplot(x = ‘total_bill’, y = ‘tip_fraction’, data = t);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘time’);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘sex’);

Visualization: Scatter Plot

Function to plot Stacked area Plot

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘smoker’);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘size’);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘size’, style = ‘sex’);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘time’, style = ‘sex’, size = ‘size’);

Sns.scatterplot(x = ‘total_bill’, y = ‘tip’, data = t, hue = ‘time’, style = ‘sex’, size = ‘size’);
Plt.legend(bbox_to_anchor = (1.05, 1));

transcition to reggration plots

PRIME INTUIT
Finishing School

Visualization: Scatter Plot

 # Function to plot Stacked area Plot

Sns.regplot(x = ‘total_bill’, y = ‘tip’, data = t);

Sns.regplot(x = ‘total_bill’, y = ‘tip_fraction’, data = t);

Sns.regplot(x = ‘total_bill’, y = ‘tip_fraction’, data = t, marker = ‘+’);

Diamonds data

D = Sns.load_dataset(‘diamonds’)

D.head()

Sns.scatterplot(‘x’, ‘price’, data = d);
Sns.scatterplot(‘x’, ‘price’, data = d.sample(1000));

PRIME INTUIT
Finishing School

Visualization: Scatter Plot

 # Function to plot Stacked area Plot

Sns.regplot(‘x’, ‘price’, data = d.sample(1000));

Sns.regplot(‘x’, ‘price’, data = d.sample(1000), order = 2);

Sns.regplot(‘x’, ‘price’, data = d.sample(1000), order = 2, marker = ‘+’);

PRIME INTUIT
Finishing School

Visualization: Bar Plots

PRIME INTUIT
Finishing School

Visualization: Bar Plot

Categorical variable

Sns.barplots(x = ‘day’, y = ‘tip’, data = t);

Sns.barplots(x = ‘day’, y = ‘tip_fraction’, data = t);

Sns.barplots(x = ‘day’, y = ‘tip’, data = t, estimator = np.median);

passing your own function.

```
Def my_estimate (V):
      return np.quantile(v, 0.25)
```

Sns.barplots(x = ‘day’, y = ‘tip’, data = t, estimator = my_estimate);

change percentile to 0.75 and rerun

PRIME INTUIT
Finishing School

Visualization: Bar Plot

 # Categorical variable

 # adding another category

Sns.barplots(x = ‘day’, y = ‘tip’, hue = ‘sex’, data = t, estimator = np.median);

Sns.barplots(x = ‘day’, y = ‘tip’, hue = ‘smoker’, data = t, estimator = np.median);

Sns.barplots(x = ‘day’, y = ‘tip’, hue = ‘time’, data = t, estimator = np.median);

Sns.barplots(x = ‘day’, y = ‘tip_fraction’, hue = ‘time’, data = t, estimator = np.median);

Visualization: Continuous Vs Continuous

Visualization: Bar Plot

```
# Continues  variable

Sns.scatterplots( ‘x’, ‘price’, data = d.sample(1000));

Sns.barplot( ‘x’, ‘price’, data = d.sample(500));

# Divide the continue value in x to a few bins or quantize x into a few bins

D[‘x_q’] = pd.cut( d[‘x’], bins = 7 );

d.head()

D[‘x_q’].unique()

Sns.barplot(‘x_q’, ‘price’, data = d.sample(1000));    # change bins size to 15
```

PRIME INTUIT
Finishing School

Visualization: Bar Plot

 # Continues variable

D[‘x_q’] = pd.cut(d[‘x’], bins = 7, labels = False);

d.head()

Sns.barplot(‘x_q’, ‘price’, data = d.sample(1000));

Visualization: Line plot

PRIME INTUIT
Finishing School

Visualization: Line Plot

```
# Time series based data

F = sns.load_dataset('fmri')

f.head()

Sns.lineplot('timepoint', 'signal', data = f);

 Sns.lineplot('timepoint', 'signal', data = f, hue = 'region');

 Sns.lineplot('timepoint', 'signal', data = f, hue = 'event');

 Sns.lineplot('timepoint', 'signal', data = f, hue = 'event', style = 'region');

Sns.lineplot('timepoint', 'signal', data = f, marker = True, estimator = np.median);

Sns.lineplot('timepoint', 'signal', data = f, units = 'subject', estimator = None);
```

Visualization: Line Plot

Time series based data

Plot for each subject

Sns.lineplot(‘timepoint’, ‘signal’, data = f, units = ‘subject’, estimator = None);

F_ = f[(f.region == ‘parietal’) & (f.event ==‘cue’)]

F_.head()

Sns.lineplot(‘timepoint’, ‘signal’, data = f_, units = ‘subject’, estimator = None);

Sns.lineplot(‘timepoint’, ‘signal’, data = f_, hue = ‘subject’, estimator = None);

Visualization: Line Plot

```
# Time series based data

# ploting with numpy array

X = np.array([ -3, -2, -1, 0, 1, 2, 3])

Y = x * x

Sns.lineplot(x, y);
```

Visualisation: Line Plots

Line Plot with Covid data

```
Import json
With open('data.json') as f:
Data = json.load(f)
Data = data['state_daily']                    # dictionary
Cd = pd.json_normalization(data)
Cd['date'] = pd.to_datatime(cd['date'])
cd.drop('tt', axis = 1, inplace = True)
cd.set_index('date', inplace = True)
Cd = cd[cd['status'] == 'confirmed']
Cd.drop('status', axis =1, inplace = True)
Cd = cd.apply(pd.to_numeric)
cd.reset_index(inplace = True)
cd.head()
```

Visualisation: Line Plots

Line Plot with Covid data

```
##  Date        | State   |          Value
2020-03-14   |an       |             0
2020-03-14   |ap       |             1
Cd_ - pd.melt(cd, id_vars = ‘date’, value_vars = list(cd.columns).remove(‘date’),
var_name = ‘state’, value_name= ‘confirmed’)

Cd_.head()

Sns.lineplot(‘date’, ‘confirmed’, data = cd_);

Sns.lineplot(‘date’, ‘confirmed’, hue = ‘state’, data = cd_);

States = [‘mh’, ‘tn’, ‘dl’, ‘wb’, ‘ka’, ‘gj’]
cd_ = cd_[cd_.state.isin(states)]
cd_.head()

Sns.lineplot(‘date’, ‘confirmed’, hue = ‘state’, data = cd_);
```

Visualization: Line Plots

Improving a Line Plot with Covid data

Sns.lineplot('date', 'confirmed', hue = 'state', data = cd_, palette = 'reds');

Fig = plt.gcf()
Fig.set_size_inches(15, 6);
Sns.lineplot('date', 'confirmed', hue = 'state', data = cd_, palette = 'reds', hue_order = ['ka', 'wb', 'gj', 'dl', 'tn', 'mh']);

Rolling Mean:

Cd = cd.rolling(7).mean()

Visualization: Heat Map

Visualization: Heat map

Line plots: continuous variable on x and y or time series

What if we have descreate data on both x and y

Matrix: x [10 x 10]

X[I, j] = value

X = random.rand(10, 10)

Sns.heatmap (x);

Mon	Tue	Wed	Thurs	Fri	Sat	Sun
Wheat Bread						
Pan cake						
omelate						
Curry buns						
baccon						
sausage						
Scrambel ed eggs						

Visualization: Heat map

Flt = sns.load_dataset('flights')

Flt.head()

Flt.sample(10)

Year / month	jan	feb	march	april
1949				
1950				
1951				

Flt_ = flt.pivot(index = 'year', columns = 'month', values = 'passengers')
Flt_.head()

Visualization: Heat map

```
Sns.heatmap(flt_)

Sns.heatmap(flt_.T)

Sns.heatmap(flt_.T, annot = True, fmt = ‘d’);

Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.heatmap(flt_.T, annot = True, fmt = ‘d’);


Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.heatmap(flt_.T, annot = True, fmt = ‘d’, cmap = ‘YlGnBu’);
```

PRIME INTUIT
Finishing School

Visualization: Heat map

```
Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.heatmap(flt_.T, annot = True, fmt = ‘d’, cmap = sns.diverging_palette(250, 10, n= 10));

## Change color numbers….  (50, 200, n = 45)

Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.heatmap(flt_.T, annot = True, fmt = ‘d’, cmap = sns.diverging_palette(250, 10, n= 10),
Center = flt_.loc[1955, ‘July’]);

Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.heatmap(flt_.T, annot = True, fmt = ‘d’, cmap = sns.diverging_palette(250, 10, n= 10),
Center = flt_.loc[1954, ‘January’]);
```

Visualisation: Tasks on open ended visualisation

PRIME INTUIT
Finishing School

Visualization: Heat map

http://ml-india.org/datasets

Ameo_2015

Df = pd.read_excel(‘Ameo_2015.xlsx’)

Df.head()

Approaching a open ended Data science problem

Recap on NumPy and Pandas

Preface

How to handle missing data

Missing data with Pandas

Open ended descriptive statistics

Example Part 1

Example Part 2

Handling missing data with NumPy

```
Import numpy as np
Import pandas as pd
Import seaborn as sns

X = np.array([1, 2, 3, 4, 5])

x.sum()

Print(x.dtype)

X = np.array([1, 2, 3, '—', 5])

Print(x.dtype)

x.sum()
```

Handling missing data with NumPy

X = np.array([1, 2, 3, None, 5])

Print(x.dtype)

x.sum()

X = np.array([1, 2, 3, np.nan, 5])

x.sum()

1 + np.nan
1* np.nan

How would you approach this problem ?

PRIME INTUIT
Finishing School

Handling missing data with NumPy

```
X_b = np.array([True, True, True, False, True])

#Indexing

X[x_b]

X[x_b].sum()

X[x_b].mean()
```

Standard np function masked arrays

```
M_x = np.ma.masked_array(x, mask = [0, 0, 0, 1, 0])

M_x.sum()
M_x.mean()
```

PRIME INTUIT
Finishing School

Handling missing data with Pandas

```
# Pandas does not support masked arrays

Df = pd.read_csv("rooms.csv")

Df.head()

Df.dtypes

%timeit np.arange(100000, dtype = "int").sum()

%timeit np.arange(100000, dtype = "object").sum()

Df.Room_Number.isnull()

Df.Room_Number.isnull().sum()
```

Handling missing data with Pandas

```
# Pandas does not support masked arrays

Df.isnull()

# note na is not identified as null

Df.isnull().sum()

Missing_values = [“NA”,  “n/a”, “na”]

Df = pd.read_csv(“rooms.csv”, na_values = Missing_values)

Df.isnull()

Df.Num_Students.sum()
Df.Num_Students.mean()
```

Handling missing data with Pandas

 # Pandas does not support masked arrays

Missing_values = [“NA”, “n/a”, “na”, “Empty”, “- -”]

Df = pd.read_csv(“rooms.csv”, na_values = Missing_values)

Df.isnull()

Df.Department.unique()

Handling missing data with Pandas

```
# How do you fill some useful values for the missing data

Df.Occupied.fillna("N")

Df.Occupied.fillna("N", inplace = True)

Def convert_to_binary(v):
    if v == 'Y':
            return True
    else:
            return False

Df.Occupied = Df.Occupied.apply(convert_to_binary)

Df
Df["Dept2"] = df.Department
Df.Department.fillna(method = "ffill", inplace = True)
Df
```

Handling missing data with Pandas

 # How do you fill some useful values for the missing data

```
Df.Dept2.fillna(method = “bfill”, inplace = True)
Df

Df.Num_Students.fillna(df.Num_Students.median(), inplace = True)

Df

Df.Room_number.interpolate(inplace = True)
df
```

PRIME INTUIT
Finishing School

Mini Project 1, AMEO 2015

AMEO 2015

```
# Understanding the data

Df = pd.read_excel("Ameo_2015.xlsx")

Df.head()

Df.shape

Df.isnull.sum()

Df.isnull.sum().sum()

Df.dtypes
```

AMEO 2015

Questions:

1) **Does the data indicate any gender bias in terms of education (10^{th}, 12^{th}, Degree) ?**
2) **Is there a Gender bias in the Job opportunities?**

AMEO 2015

Df.Gender.unique()

Sns.violinplot(x = ‘Gender’, y = ‘Salary’, data = df);

Df[[’10percentage’, ’12percentage’, ‘collegeGPA’, ‘Gender’]].groupby(‘Gender’).mean()

Df[[’10percentage’, ’12percentage’, ‘collegeGPA’, ‘Gender’]].groupby(‘Gender’).median()

Df[[‘conscientiousness’, ‘agreeableness’, ‘extraversion’, ‘nueroticism’, ‘openess_to_experience’, ‘Gender’]]. groupby(‘Gender’).mean()

Df[[‘conscientiousness’, ‘agreeableness’, ‘extraversion’, ‘nueroticism’, ‘openess_to_experience’, ‘Gender’]]. groupby(‘Gender’).median()

PRIME INTUIT
Finishing School

AMEO 2015

Df[[‘Salary’, ‘Gender’]].groupby(‘Gender’).mean()

Df.Salary.mean()

Th = Df.Salary.mean() + df.Salary.std()

Df[‘HighIncome’] = (df.Salary > th)

Df.sample(10)

Df[[‘Salary’, ‘HighIncome’, ‘Gender’]].groupby([‘Gender’, ‘HighIncome’]).mean()

Df[[‘Salary’, ‘HighIncome’, ‘Gender’]].groupby([‘Gender’, ‘HighIncome’]).count()

PRIME INTUIT
Finishing School

AMEO 2015

Print('Low Income female percentage' , f/(m+fnumbers) *100)

Print('High Income female percentage' , f/(m+fnumbers) *100)

Df.head()

Df.CollegeTier.unique()

Df[['CollegeTier', 'HighIncome', 'Salary']].groupby(['HighIncome', 'CollegeTier']).count()

Print('Low Income college tier 2 percentage is' , high income false 2 /(high income false 1 + 2) *100)

Print('high Income college tier 2 percentage is' , high income True 2 /(high income True 1 + 2) *100)

AMEO 2015

Df[['Gender', 'CollegeTier', 'Salary']].groupby(['CollegeTier', 'Gender']).count()

Print('in college tier 1 female percentage is' , --*100)

Print('in college tier 2 female percentage is' , --*100)

Mini Project 2, India Agriculture data

Agri Data

```
Df = pd.read_csv('apy.csv')

Df.head()

Df.State_Name.unique()

Df.Crop_Year.unique()

Df.Season.unique()

Df.Crop.unique()

Df.dtypes

Pd.to_numeric(Df. Production)

Df = pd.read_csv('apy.csv', na_values = '=')
Df.dtypes
```

Agri Data

Df.Production.isnull().sum()

df.shape

Df.dropna(inplace = True) # , thresh

Df.shape

What are questions we can answer ?

State Production

Crops

Variations within states

Agri Data

Data Distribution:

Sns.kdeplot(df.Production);

Sns.boxplot(df.Production);

Sns.boxplot(df.Area);
Sns.boxplot(df.Area);

 # state wise production

Df[df.State_Name == ‘Karnataka’]

Df[df.State_Name == ‘Karnataka’][‘District_Name’].unique()

PRIME INTUIT
Finishing School

Agri Data

```
Df.groupby([‘State_Name’, ‘Crop’, ‘Crop_Year’]).sum()

Df[df.State_Name == ‘West Bengal”][‘Crop’].unique()

Df.groupby([‘State_Name’, ‘Crop_Year’]).sum()

Df_ = Df.groupby([‘State_Name’, ‘Crop_Year’]).sum()

Df_.reset_index(inplace = True)

Df_.head()

Df_[[‘State_Name’, ‘Crop_Year’]].groupby(‘State_Name’).count()

Sns.lineplot(x=‘Crop_Year’, y = ‘Area’, data = df[df.State_Name == “Karanataka”]);

Sns.lineplot(x=‘Crop_Year’, y = ‘Area’, data = df[df.State_Name == “Odisha”]);
```

Agri Data

```
Sns.lineplot(x=‘Crop_Year’, y = ‘Production’, data = df[df.State_Name == “Karanataka”]);

Sns.lineplot(x=‘Crop_Year’, y = ‘Production’, data = df[df.State_Name == “Odisha”]);


Sns.lineplot(x=‘Crop_Year’, y = ‘Production’, data =df, hue =‘State_Name’);

Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.lineplot(x=‘Crop_Year’, y = ‘Production’, data =df, hue =‘State_Name’);
```

PRIME INTUIT
Finishing School

Agri Data

```
Sns.lineplot(x='Crop_Year', y = 'Production', data = df[df.State_Name == "Karanataka"]);

Sns.lineplot(x='Crop_Year', y = 'Production', data = df[df.State_Name == "Odisha"]);


Sns.lineplot(x='Crop_Year', y = 'Production', data =df, hue ='State_Name');

Fig = plt.gcf();
Fig.set_size_inches(15, 10)
Sns.lineplot(x='Crop_Year', y = 'Production', data =df, hue ='State_Name');
```

Plotly library

Import plotly_express as px

!pip3 install plotly_express

Px.scatter(df_, x = ‘Area’, y = ‘Production’, animation_frame = ‘Crop_Year’, animation_grop = ‘State_Name’, color = ‘State_Name’)

- - - - - - - - - - - - - - - - -

Df_.sort(‘Crop_Name’, inplace = True)

Df[(df.State_Name == ‘Kerala’) & (df.Crop_Year = = 2000)].sort_values(‘Production’)

Df[df.Crop.isin([‘Rice’, ‘Wheat’, ‘Maize’, ‘Ragi’])]

Df_ = Df[df.Crop.isin([‘Rice’, ‘Wheat’, ‘Maize’, ‘Ragi’])]. Groupby([‘State_Name’, ‘Crop_Year’]).sum()

Df_.reset_index(inplace = True)

Plotly library

Df_.sort('Crop_Name', inplace = True)

Px.scatter(df_, x = 'Area', y = 'Production', animation_frame = 'Crop_Year', animation_grop = 'State_Name', color = 'State_Name')

Df_['Efficiency'] = df_['Production'] / df_['Area']

Px.scatter(df_, x = 'Area', y = 'Efficiency', size = 'Production', animation_frame = 'Crop_Year', animation_grop = 'State_Name', color = 'State_Name')

Px.scatter(df_, x = 'Area', y = 'Efficiency', size = 'Production', animation_frame = 'Crop_Year', animation_grop = 'State_Name', color = 'State_Name', range_y = [0.75, 5], range_x = [1E6, 20E6])

PRIME INTUIT
Finishing School

Thank You !