Data Analysis Course

Data Visualization

Ahmed Muhumed

Sahan Institute of Technology

Hargeisa, Somaliland

December 22, 2022

Data Visualization in Data Analysis

Data Visualization

- Data visualization: Data visualization is in many ways the culmination of the data analysis process.
- Data visualization is the graphical representation of data.
- It can help you share meaningful insights about your data with stakeholders and tell your data story more effectively.
- In short, data visualization is the graphic representation and presentation of data.
- data visualization is a way to show a complex data in a form that is graphical and easy to understand
- Also since a picture is worth a thousand words then plots and graphs can be very effective in conveying a clear description of the data especially when disclosing findings to an audience.

Data Visualization

Why Build Visuals?

- For exploratory data analysis
- Communicate data clearly
- Share unbiased representation of data
- Use them to support recommendations to different stakeholders

When creating a visual, always remember:

- Less is more effective
- Less is more attractive
- Less is more impactive

Data Visualization

- Visualizations can be used two main purposes when analyzing;
  1. Exploratory Data Analysis
  2. Explanatory Data Analysis
- In exploratory data analysis, you are:
  - Looking for relationships in the data
  - Connecting questions about the data
  - And the visualizations don't need to be perfect
- Explanatory data analysis:
  - Is done after you find an insight
  - Visualizations need to be surrounded by a story
  - Do not need to include unneeded information

Data Visualization

- In this course, we will make use of two Python visualizations libraries; **Matplotlib** and **Seaborn**, and also plotting functions built into the pandas library
- Often, when choosing a tool to work with, there is a trade-off between flexibility and productivity.
- On one end, you have a tool that has a high amount of flexibility, and on the other end, you have a tool with good at productivity.
- **Matplotlib** has higher flexibility where **seaborn** and **pandas plotting** are more productive.

Design of Visualizations

Design of Visualizations

- Before making visualizations, it is important to understand how visualizations are designed, and what makes a visualization "good" or "bad."
- Explanatory visualizations utilize specific design principles in order to clearly and accurately convey findings.
- Design principles can also be useful in the exploratory phase of the data analysis process.
- Visuals can be bad if they:
  - Don't convey the desired message.
  - Are misleading.

The Four Levels of Measurement

- In order to choose an appropriate plot type or method of analysis for your data, you need to understand the types of data you have.
- One common method divides the data into four levels of measurement:
  1. **Nominal Data**: ure labels without inherent order (no label is intrinsically greater or less than any other)
  2. **Ordinal Data**: abels with an intrinsic order or ranking (comparison operations can be made between values, but the magnitude of differences are not be well-defined)
  3. **Interval Data**: numeric values where absolute differences are meaningful (addition and subtraction operations can be made)
  4. **Ratio Data**: numeric values where relative differences are meaningful (multiplication and division operations can be made)

Continue...

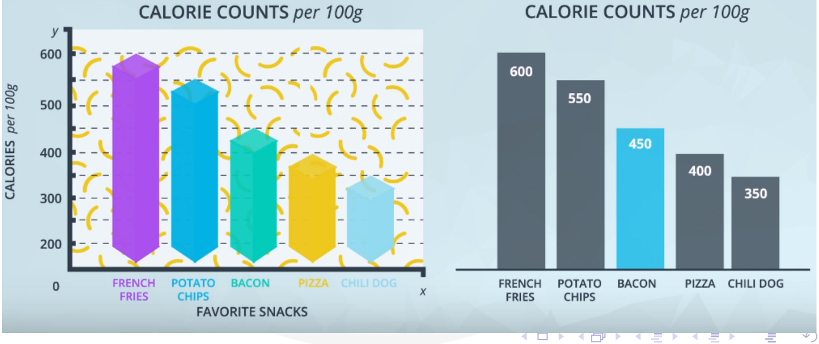
- The first two are known as **Qualitative** or categorical types (non-numeric types)
- The interval data and ratio data are known as **Quantitative** or numeric types
- All quantitative-type variables also come in one of two varieties: discrete and continuous.
    - **Discrete** quantitative variables can only take on a specific set values at some maximum level of precision.
    - **Continuous** quantitative variables can (hypothetically) take on values to any level of precision.
- Distinguishing between continuous and discrete can be a little tricky –a rule of thumb is if there are few levels, and values can't be subdivided into further units, then it's discrete. Otherwise, it's continuous.

Visual Encoding and Chart Junk

- Encoding in data viz basically means translating the data into a visual element on a chart/map/whatever you are making.
- You need to do it right, because doing it right will mean that other people looking at your visualizations can understand what you are trying to say or show.
- Chart junk refers to all visual elements in charts and graphs that are not necessary to comprehend the information represented on the graph or that distract the viewer from this information.
    - Heavy grid lines
    - Unnecessary text
    - Pictures surrounding the visual
    - Shading or 3d components
    - Ornamented chart axes

Data-ink Ratio

- The data-ink ratio, credited to Edward Tufte, is directly related to the idea of chart junk.
- The more of the ink in your visual that is related to conveying the message in the data, the better.
- Limiting chart junk increases the data-ink ratio.



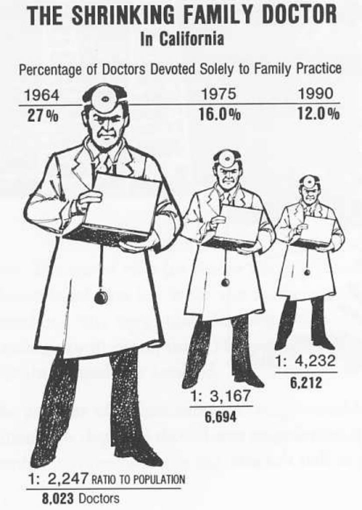
Design Integrity

- It is key that when you build plots you maintain integrity for the underlying data.
- One of the main ways discussed here for looking at data integrity was with the **lie factor**.
- Lie factor depicts the degree to which a visualization distorts or misrepresents the data values being plotted. It is calculated in the following way:

$$lie factor = \frac{\triangle Visual/Visual_{start}}{\triangle Data/Data_{start}} \tag{1}$$

- The delta symbol (Δ) stands for difference or change.
- In words, the lie factor is the relative change shown in the graphic divided by the actual relative change in the data.

Ideally, the lie factor should be 1: any other value means that there is some mismatch in the ratio of depicted change to actual change.



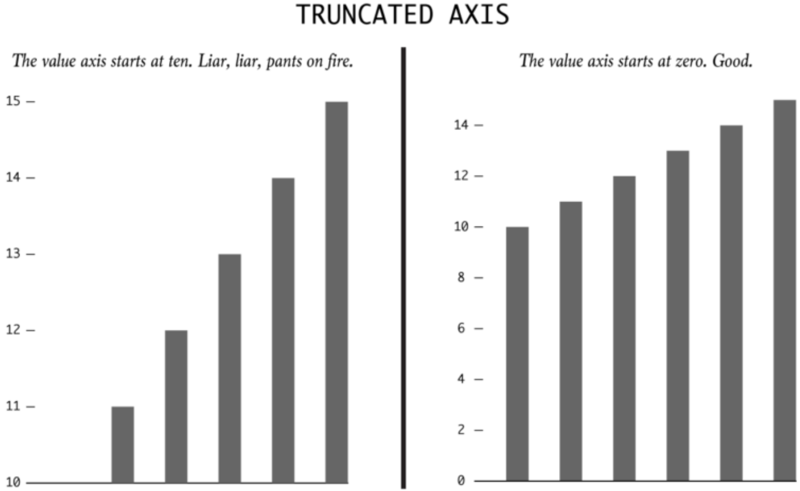
The number of pixels related to the largest image is 79,000 and 16,500 for the smallest.

The percentage change is 27% to 12%. So, the lie factor is calculated as:

$$liefactor = \frac{(79000 - 16500)/16500}{(27 - 12)/12}$$

$$= 3.03$$

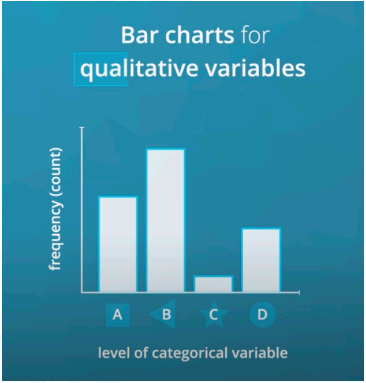
- Bar charts use length as their visual cue, so when someone makes the length shorter using the same data by truncating the value axis, the chart dramatizes differences.
- Someone wants to show a bigger change than is actually there.



Univariate Exploration of Data

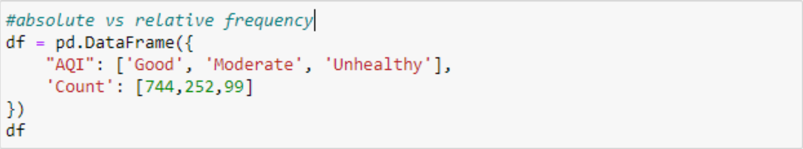
Bar Charts

- A bar chart depicts the distribution of a categorical variable.
- In a bar chart, each level of the categorical variable is depicted with a bar, whose height indicates the frequency of data points that take on that level.
- For nominal data, the bars can be ordered by frequency to easily see which category is the most common.
- Ordinal data should not be re-ordered because the inherent ordering of the levels is typically more important to display.

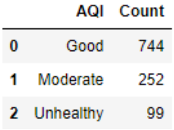


Absolute Vs. Relative Frequency

- In relative frequency, the charts are encoded the proportion of the data that falls in each category
- In absolute frequency, the charts are encoded the total number of data points for each category.
- For example; let say we have this dataframe on daily air quality from three different cities



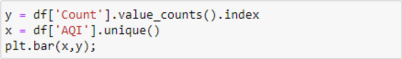
```
#absolute vs relative frequency
df = pd.DataFrame({
    "AQI": ['Good', 'Moderate', 'Unhealthy'],
    'Count': [744,252,99]
})
df
```



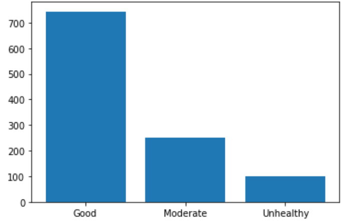
	AQI	Count
0	Good	744
1	Moderate	252
2	Unhealthy	99

Continue...

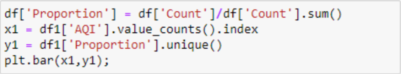
- Absolute Frequency



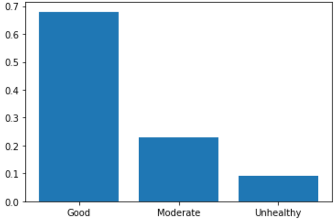
```
y = df['Count'].value_counts().index
x = df['AQI'].unique()
plt.bar(x,y);
```



- Relative Frequency

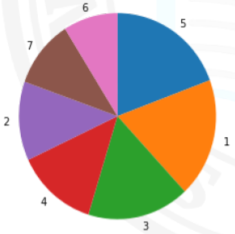


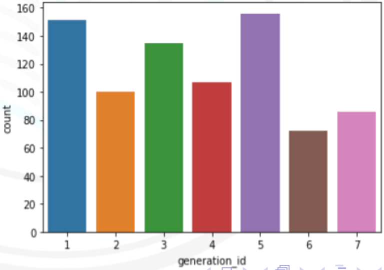
```
df['Proportion'] = df['Count']/df['Count'].sum()
x1 = df1['AQI'].value_counts().index
y1 = df1['Proportion'].unique()
plt.bar(x1,y1);
```



Pie Chart

- A pie chart is a common univariate plot type that is used to depict relative frequencies for levels of a categorical variable.
- Frequencies in a pie chart are depicted as wedges drawn on a circle: the larger the angle or area, the more common the categorical value taken.
- Use a Pie chart only when the number of categories is less, and you'd like to see the proportion of each category on a chart.





Guidelines to Use a Pie Chart

If you want to use a pie chart, try to follow certain guidelines:

- Make sure that your interest is in relative frequencies. Areas should represent parts of a whole, rather than measurements on a second variable (unless that second variable can logically be summed up into some whole).
- Limit the number of slices plotted. A pie chart works best with two or three slices, though it's also possible to plot with four or five slices as long as the wedge sizes can be distinguished. If you have a lot of categories, or categories that have small proportional representation, consider grouping them together so that fewer wedges are plotted, or use an 'Other' category to handle them.
- Plot the data systematically. One typical method of plotting a pie chart is to start from the top of the circle, then plot each categorical level clockwise from most frequent to least frequent.

Histogram

- A histogram is used to plot the distribution of a numeric variable.
- It's the quantitative version of the bar chart.
- However, rather than plot one bar for each unique numeric value, values are grouped into continuous bins, and one bar for each bin is plotted to depict the number.
- You can use either Matplotlib or Seaborn to plot the histograms.
- You can use the default settings for matplotlib's hist() function to plot a histogram with 10 bins:

```
plt.hist(data = pokemon, x = 'speed')
```

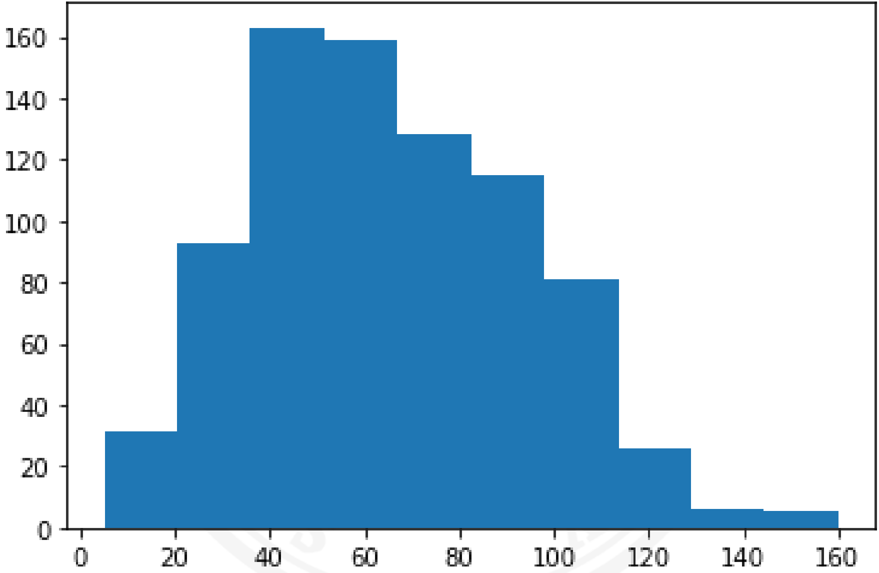
Figures, Axes and Subplots

- The base of visualization in matplotlib is a Figure object. Contained within each Figure will be one or more Axes objects, each Axes object containing a number of other elements that represent each plot.

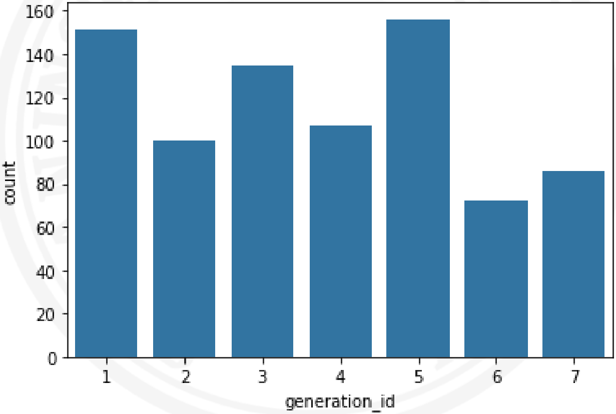
```
#Create a new figure
fig = plt.figure()
```

- The argument of add_axes represents the dimensions [left, bottom, width, height] of the new axes.

```
ax = fig.add_axes([.125, .125, .775, .755])
ax.hist(data=pokemon, x='speed');
```


160
140
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160

```
fig = plt.figure()
ax = fig.add_axes([.125, .125, .775, .755])
base_color = sb.color_palette()[0]
sb.countplot(data = pokemon, x = 'generation_id',
                color = base_color, ax = ax)
```



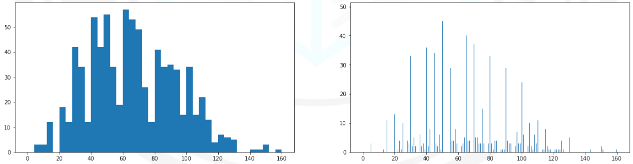
- Let's review in detail how subplot() was used on the Histograms

```
# Resize the chart, and have two plots side-by-side
# set a larger figure size for subplots
plt.figure(figsize = [20, 5])

# histogram on left, example of too-large bin size
# 1 row, 2 cols, subplot 1
plt.subplot(1, 2, 1)
bins = np.arange(0, pokemon['speed'].max()+4, 4)
plt.hist(data = pokemon, x = 'speed', bins = bins);

# histogram on right, example of too-small bin size
plt.subplot(1, 2, 2) # 1 row, 2 cols, subplot 2
bins = np.arange(0, pokemon['speed'].max()+1/4, 1/4)
plt.hist(data = pokemon, x = 'speed', bins = bins);
```

- First of all, plt.figure(figsize = [20, 5])creates a new Figure, with the ”figsize” argument setting the width and height of the overall figure to 20 inches by 5 inches, respectively.
- Then, plt.subplot(1, 2, 1) creates a new Axes in our Figure, its size determined by the subplot() function arguments
- The first two arguments says to divide the figure into one row and two columns, and the third argument says to create a new Axes in the first slot.
- Finally, plt.subplot(1, 2, 2) creates a new Axes in the second subplot slot



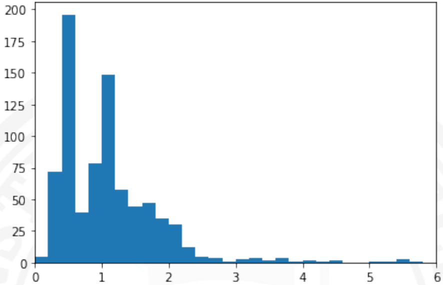
Choosing a Plot for Discrete Data

- If you want to plot a discrete quantitative variable, it is possible to select either a histogram or a bar chart to depict the data.
- Here, the discrete means non-continuous values. In general, a discrete variable can be assigned to any of the limited (countable) set of values from a given set/range.
- The quantitative term shows that it is the outcome of the measurement of a quantity.
- The histogram is the most immediate choice since the data is numeric, but there's one particular consideration to make regarding the bin edges.
- Since data points fall on set values (bar-width), it can help to reduce ambiguity by putting bin edges between the actual values taken by the data.

Descriptive Statistics, Outliers and Axis Limits

- Visualizations will give you insights into the data that you can't get from descriptive statistics. A plot can show:
  - if the data is symmetric or skewed
  - interesting areas for further investigation or clarification
  - potential errors in the data
- In a histogram, you can observe whether or not there are outliers in your data.
- In order to change a histogram's axis limits, you can add a Matplotlib xlim() call to your code. The function takes a tuple of two numbers specifying the upper and lower bounds of the x-axis range. See the example below.

```
bins = np.arange(0, pokemon['height'].max()+0.2, 0.2)
plt.hist(data=pokemon, x='height', bins=bins);
plt.xlim((0,6));
```



- In the generic example above, we might be interested in comparing patterns in other variables between data points that take values less than 6 to those that take values greater than 6.
- For anything that is concentrated on the bulk of the data in the former group (< 6), the use of axis limits can allow focusing on data points in that range without needing to go through the creation of new DataFrame filtering out the data points in the latter group (> 6).

Bivariate Exploration of Data

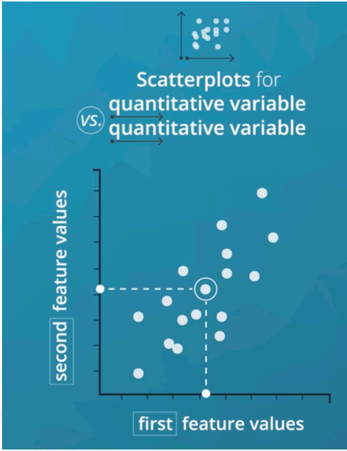
Bivariate Exploration of Data

- Bivariate Visualizations are those that involve two variables, the variation in one variable will affect the value of the other variable.
- Bi means two and variate means variable, so here there are two variables.
- The analysis is related to cause and the relationship between the two variables.
- Bivariate plots are these three major plots:
  - **Scatterplots** for quantitative variable versus quantitative variable
  - **Violin** plots for quantitative variable versus qualitative variable
  - **Clustered bar** charts for qualitative variable versus qualitative variable

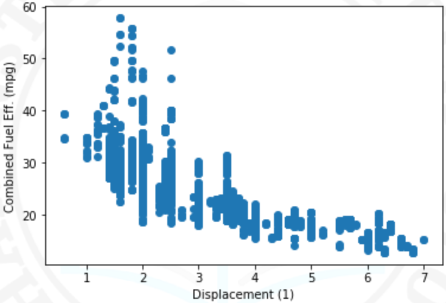
Scatterplots and Correlation

- A scatterplot is used to show the relationship between two quantitative variables.
- The two variables are indicated on X and Y-axis, respectively.
- Through the scatterplots, we can see clearly how these two variables correlate with each other.
- To quantify how strong the correlation is between the variables, we use a **correlation coefficient**.
- **Pearson correlation coefficient (r)** captures linear relationships. It is a value ranging from -1 to +1.
- A positive value of r indicates the increase in one variable tends to increase another variable.
- On the other hand, a negative r means the increase in one variable tends to cause a decrease in another variable.
- A value close to 0 indicates a weak correlation, and a value close to -1 and +1 indicates a strong correlation.

- If we want to inspect the relationship between two numeric variables, the standard choice of plot is the scatterplot.
- In a scatterplot, each data point is plotted individually as a point, its x-position corresponding to one feature value and its y-position corresponding to the second.
- Let's look at the Matplotlib's Scatter function in the next slide.



```
fuel_econ = pd.read_csv('fuel_econ.csv')
plt.scatter(data = fuel_econ, x = 'displ', y = 'comb');
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)')
```

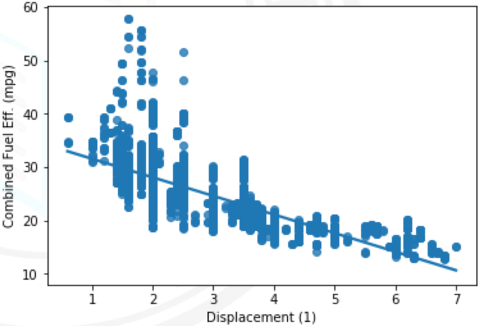


In the example above, the relationship between the two variables is negative because as higher values of the x-axis variable are increasing, the values of the variable plotted on the y-axis are decreasing.

- An alternative way of creating a scatter plot is through Seaborn's regplot function. Seaborn's regplot() function combines scatterplot creation with regression function

```
sb.regplot(data = fuel_econ, x = 'displ', y = 'comb');
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)')
```

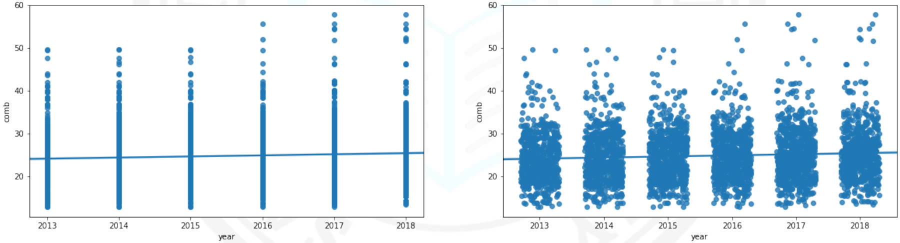
The regression line in a scatter plot showing a negative correlation between the two variables.



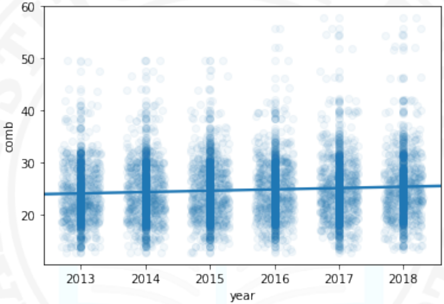
Overplotting, Transparency, and Jitter

- If we have a very large number of points to plot or our numeric variables are discrete-valued, then it is possible that using a scatterplot straightforwardly will not be informative.
- The visualization will suffer from **overplotting**, where the high amount of overlap in points makes it difficult to see the actual relationship between the plotted variables.
- In overplotting, you have too many points in a small area
- Jitter is changing the location of data points by randomly adding/subtracting a small value to each.
- Transparency is changing the appearance of data points to more easily see how many data points may be stacked on top of one another.

```
# Resize figure to accommodate two plots
plt.figure(figsize = [20, 5])
# PLOT ON LEFT - SIMPLE SCATTER
plt.subplot(1, 2, 1)
sb.regplot(data = fuel_econ, x = 'year', y = 'comb', truncate=False);
# PLOT ON RIGHT - SCATTER PLOT WITH JITTER
plt.subplot(1, 2, 2)
# In the sb.regplot() function below, the `truncate` argument accepts a boolean.
# If truncate=True, the regression line is bounded by the data limits.
# Else if truncate=False, it extends to the x axis limits.
# The x_jitter will make each x value will be adjusted randomly by +/-0.3
sb.regplot(data = fuel_econ, x = 'year', y = 'comb', truncate=False, x_jitter=0.3);
```



```
# The scatter_kws helps specifying the opaqueness of the data points.
# The alpha take a value between [0-1], where 0 represents transparent, and 1 is opa
sb.regplot(data = fuel_econ, x = 'year', y = 'comb', truncate=False, x_jitter=0.3,
# Alternative way to plot with the transparency.
# The scatter() function below does NOT have any argument to specify the Jitter
plt.scatter(data = fuel_econ, x = 'year', y = 'comb', alpha=1/20);
```



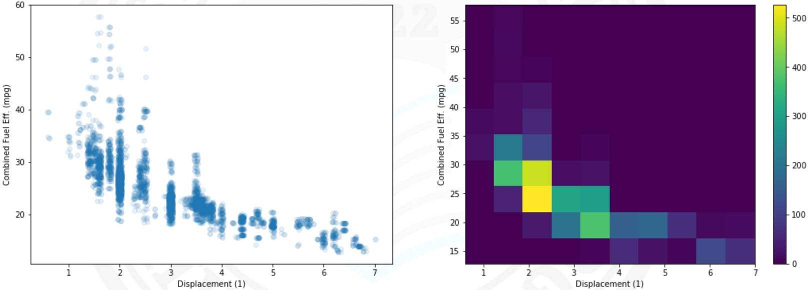
In the plot above, the jitter settings will cause each point to be plotted in a uniform ±0.3 range of their true values. Note that transparency has been changed to be a dictionary assigned to the "*scatter_kws*" parameter.

Heat Maps

- A heat map is a 2-d version of the histogram that can be used as an alternative to a scatterplot.
- Like a scatterplot, the values of the two numeric variables to be plotted are placed on the plot axes.
- Similar to a histogram, the plotting area is divided into a grid and the number of points in each grid rectangle is added up.
- Since there won't be room for bar heights, counts are indicated instead by grid cell color. A heat map can be implemented with Matplotlib's **hist2d()** function.
- In heat map, the field is divided into a grid of cells, and the number of data points in each cell is counted up.

Continue...

- Heat maps are useful in the following cases:
  1. To represent a plot for discrete vs. another discrete variable
  2. As an alternative to transparency when the data points are enormous

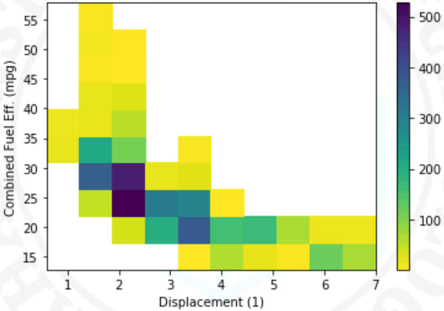


- In the example above, we added a colorbar() function call to add a colorbar to the side of the plot, showing the mapping from counts to colors.

```
# Read the CSV file
fuel_econ = pd.read_csv('fuel_econ.csv')
fuel_econ.head(10)
plt.figure(figsize = [18, 6])
# PLOT ON LEFT
plt.subplot(1, 2, 1)
sb.regplot(data = fuel_econ, x = 'displ', y = 'comb',
         x_jitter=0.04, scatter_kws={'alpha':1/10}, fit_reg=False)
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)');
# PLOT ON RIGHT
plt.subplot(1, 2, 2)
plt.hist2d(data = fuel_econ, x = 'displ', y = 'comb')
plt.colorbar()
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)');
```

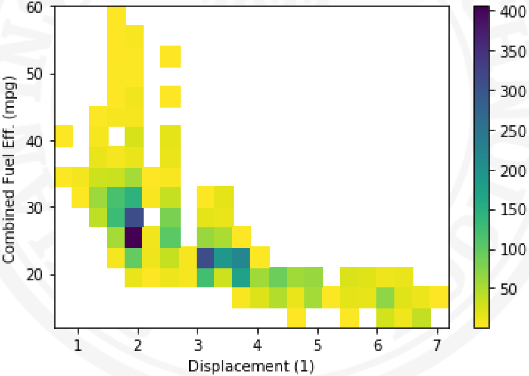
- Heat plot - Set a minimum bound on counts and a reverse color map
- To select a different color palette, you can set the ”cmap” parameter in hist2d.
- The most convenient way of doing this is to set the ”cmap” value as a string referencing a built-in Matplotlib palette.
- For now, I will just show an example of reversing the default ”viridis” color palette, by setting cmap = ’viridis_r’.
- Furthermore, I would like to distinguish cells with zero counts from those with non-zero counts.
- The ”cmin” parameter specifies the minimum value in a cell before it will be plotted.
- By adding a cmin = 0.5 parameter to the hist2d call, this means that a cell will only get colored if it contains at least one point.

```
# Use cmin to set a minimum bound of counts
# Use cmap to reverse the color map.
plt.hist2d(data = fuel_econ, x = 'displ', y = 'comb',
           cmin=0.5, cmap='viridis_r')
plt.colorbar()
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)');
```



As the count of points in the cell increases, the color in the heatmap gets brighter and moves from blue to yellow.

```
# Specify bin edges
bins_x = np.arange(0.6, 7+0.3, 0.3)
bins_y = np.arange(12, 58+3, 3)
plt.hist2d(data = fuel_econ, x = 'displ', y = 'comb',
      cmin=0.5, cmap='viridis_r', bins = [bins_x, bins_y])
plt.colorbar()
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)');
```

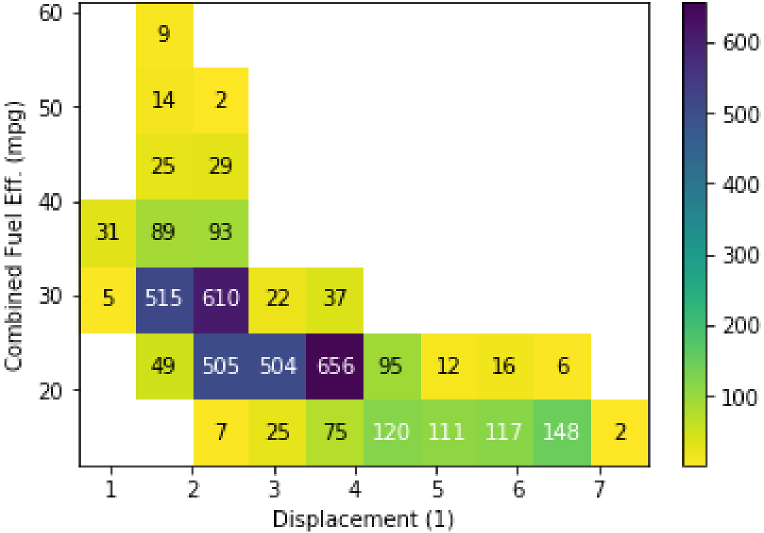


Annotations on each cell

- If you have a lot of data, you might want to add annotations to cells in the plot indicating the count of points in each cell.
- From hist2d, this requires the addition of text elements one by one, much like how text annotations were added one by one to the bar plots
- We can get the counts to annotate directly from what is returned by hist2d, which includes not just the plotting object, but an array of counts and two vectors of bin edges.
- If you have too many cells in your heat map, then the annotations will end up being too overwhelming, too much to attend to.
- In cases like that, it's best to leave off the annotations and let the data and colorbar speak for themselves.

```
# Specify bin edges
bins_x = np.arange(0.6, 7+0.7, 0.7)
bins_y = np.arange(12, 58+7, 7)
# Use cmin to set a minimum bound of counts
# Use cmap to reverse the color map.
h2d = plt.hist2d(data = fuel_econ, x = 'displ', y = 'comb',
       cmin=0.5, cmap='viridis_r', bins = [bins_x, bins_y])
plt.colorbar()
plt.xlabel('Displacement (1)')
plt.ylabel('Combined Fuel Eff. (mpg)');
# Select the bi-dimensional histogram, a 2D array of samples x and y.
# Values in x are histogrammed along the first dimension and
# values in y are histogrammed along the second dimension.
counts = h2d[0]
# Add text annotation on each cell
# Loop through the cell counts and add text annotations for each
for i in range(counts.shape[0]):
for j in range(counts.shape[1]):
c = counts[i,j]
if c >= 100: # increase visibility on darker cells
plt.text(bins_x[i]+0.35, bins_y[j]+3.5, int(c),
ha = 'center', va = 'center', color = 'white')
elif c > 0:
plt.text(bins_x[i]+0.35, bins_y[j]+3.5, int(c),
ha = 'center', va = 'center', color = 'black')
```

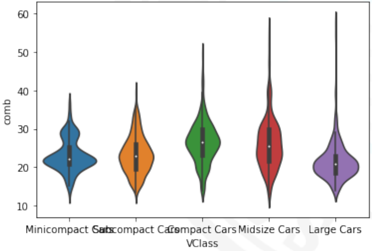
- Here is an annotated heat map



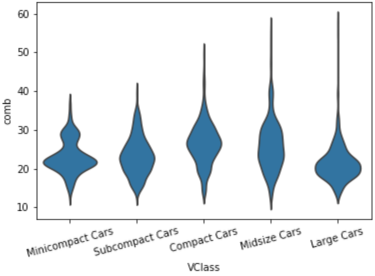
Violin Plots

- There are a few ways of plotting the relationship between one quantitative and one qualitative variable, that demonstrate the data at different levels of abstraction.
- The violin plot is on the lower level of abstraction
- For each level of the categorical variable, a distribution of the values on the numeric variable is plotted. The distribution is plotted as a kernel density estimate, something like a smoothed histogram.
- Seaborn's violinplot() function can be used to create violin plots.
- The code in the next slide is the Violin plot for plotting a Quantitative variable (fuel efficiency) versus Qualitative variable (vehicle class)

```
# Types of sedan cars
sedan_classes = ['Minicompact Cars', 'Subcompact Cars', 'Compact Ca
# Returns the types for sedan_classes with the categories and order
# Refer - https://pandas.pydata.org/pandas-docs/version/0.23.4/gene
vclasses = pd.api.types.CategoricalDtype(ordered=True, categories=s
# Use pandas.astype() to convert the "VClass" column from a plain o
fuel_econ['VClass'] = fuel_econ['VClass'].astype(vclasses);
sb.violinplot(data=fuel_econ, x='VClass', y='comb');
sb.violinplot(data=fuel_econ, x='VClass', y='comb', color=base_colo
plt.xticks(rotation=15);
```



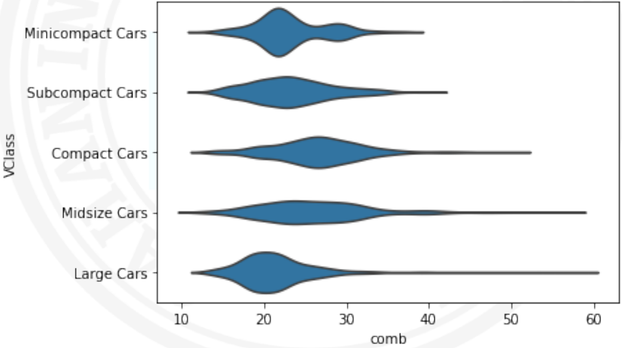
(a)



(b)

- Much like how the bar chart could be rendered with horizontal bars, the violin plot can also be rendered horizontally.
- Seaborn is smart enough to make an appropriate inference on which orientation is requested, depending on whether ”x” or ”y” receives the categorical variable

```
sb.violinplot(data=fuel_econ, y='VClass', x='comb',
                        color=base_color, inner=None);
```



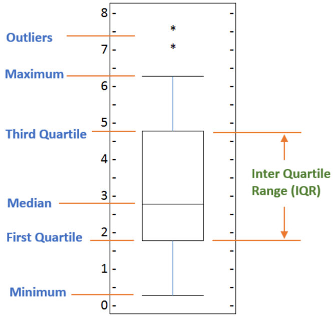
Box Plots

- A box plot can also be used to show the relationship between a numeric variable and a categorical variable.
- In a box, the central line indicates the median, the upper and lower edges show the 1st and 3rd quartiles.
- Whiskers outside of the box indicate the largest and smallest values. Box plots also have outliers plotted as points.
- Compared to the violin plot, the box plot leans more on the summarization of the data, primarily just reporting a set of descriptive statistics for the numeric values on each categorical level.
- A box plot can be created using seaborn's boxplot() function.

Continue...

A box plot is a way of statistically representing the distribution of the data through five main dimensions:

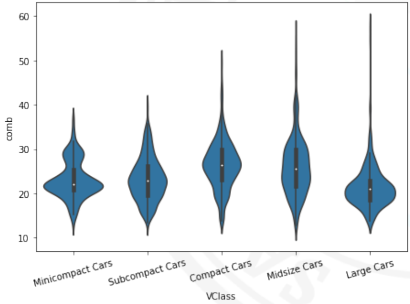
- **Minimum**: The smallest number in the dataset excluding the outliers.
- **First quartile**: Middle number between the minimum and the median.
- **Second quartile (Median)**: Middle number of the (sorted) dataset.
- **Third quartile**: Middle number between median and maximum.
- **Maximum**: The largest number in the dataset excluding the outliers

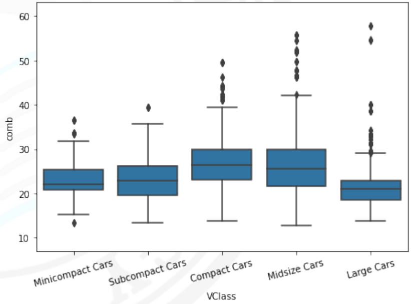


```
# Step 1. Import packages
# Step 2. Load data
# Step 3. Convert the "VClass" column from a plain object type into an orde
              #categorical type
# Types of sedan cars
sedan_classes = ['Minicompact Cars', 'Subcompact Cars', 'Compact Cars',
                                      'Midsize Cars', 'Large Cars']
# Returns the types for sedan_classes with the categories and orderedness
vclasses = pd.api.types.CategoricalDtype(ordered=True, categories=sedan_cla
# Use pandas.astype() to convert the "VClass" column from a plain object ty
                            #into an ordered categorical type
fuel_econ['VClass'] = fuel_econ['VClass'].astype(vclasses);
# Step 4.  TWO PLOTS IN ONE FIGURE
plt.figure(figsize = [16, 5])
base_color = sb.color_palette()[0]
# LEFT plot: violin plot
plt.subplot(1, 2, 1)
#Let's return the axes object
ax1 = sb.violinplot(data=fuel_econ, x='VClass', y='comb', color=base_color)
plt.xticks(rotation=15);
# RIGHT plot: box plot
plt.subplot(1, 2, 2)
sb.boxplot(data=fuel_econ, x='VClass', y='comb', color=base_color)
plt.xticks(rotation=15);
plt.ylim(ax1.get_ylim()) # set y-axis limits to be same as left plot
```

Continue...

- Here is the comparison of violin plot and box plot.
- Violin plot is good for exploration of data while box plot is good for explanatory analysis of data.





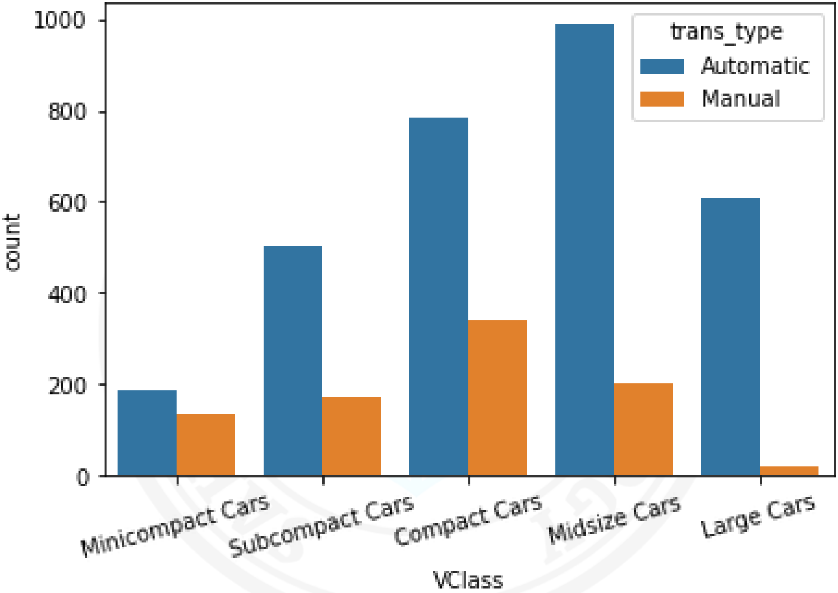
Clustered Bar Charts

- We can use clustered bar charts to show the relationship between two qualitative variables.
- However, to do this, we may need to process the second variable a little so it can be shown as a quantitative value along the y-axis.
- For example, you can count the frequency of the second variable. Also, you may also apply a color or texture encoding to distinguish the level of the second variable.
- To depict the relationship between two categorical variables, we can extend the univariate bar chart seen in the previous lesson into a clustered bar chart.
- Like a standard bar chart, we still want to depict the count of data points in each group, but each group is now a combination of labels on two variables.

Continue..

- So we want to organize the bars into an order that makes the plot easy to interpret.
- In a clustered bar chart, bars are organized into clusters based on levels of the first variable, and then bars are ordered consistently across the second variable within each cluster.
- This is easiest to see with an example, using seaborn's countplot function. To take the plot from univariate to bivariate, we add the second variable to be plotted under the "hue" argument:

```
sedan_classes = ['Minicompact Cars', 'Subcompact Cars', 'Compact Cars',
                                  'Midsize Cars', 'Large Cars']
vclasses = pd.api.types.CategoricalDtype(ordered=True, categories=sedan_classes)
fuel_econ['VClass'] = fuel_econ['VClass'].astype(vclasses);
fuel_econ['trans_type'] = fuel_econ['trans'].apply(lambda x:x.split()[0])
sb.countplot(data = fuel_econ, x = 'VClass', hue = 'trans_type')
plt.xticks(rotation = 15);
```


trans_type
Automatic
Manual
1000
800
600
400
200
0
count
Minicompact Cars
Subcompact Cars
Compact Cars
Midsize Cars
Large Cars
VClass

Faceting

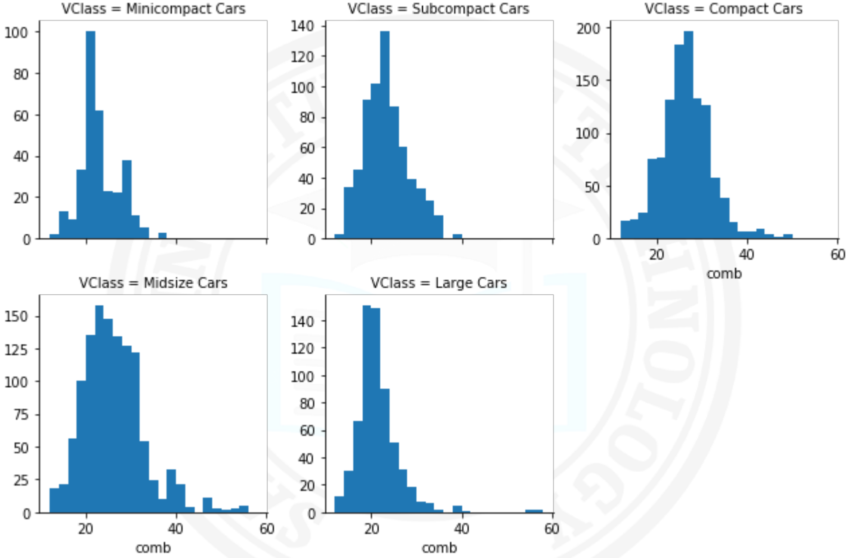
- One general visualization technique that will be useful for you to know about to handle plots of two or more variables is faceting.
- In faceting, the data is divided into disjoint subsets, most often by different levels of a categorical variable.
- For each of these subsets of the data, the same plot type is rendered on other variables.
- Faceting is a way of comparing distributions or relationships across levels of additional variables, especially when there are three or more variables of interest overall.
- While faceting is most useful in multivariate visualization, it is still valuable to introduce the technique here in our discussion of bivariate plots.

Continue...

- For example, rather than depicting the relationship between one numeric variable and one categorical variable using a violin plot or box plot, we could use faceting to look at a histogram of the numeric variable for subsets of the data divided by categorical variable levels.
- Seaborn's FacetGrid class facilitates the creation of faceted plots.
- There are two steps involved in creating a faceted plot. First, we need to create an instance of the FacetGrid object and specify the feature we want to facet by (vehicle class, "VClass" in our example).
- Then we use the map method on the FacetGrid object to specify the plot type and variable(s) that will be plotted in each subset (in this case, the histogram on combined fuel efficiency "comb").

- Faceting is most useful in multivariate visualizations
- Notice that each subset of the data is being plotted independently. Each uses the default of ten bins from hist to bin together the data, and each plot has a different bin size.

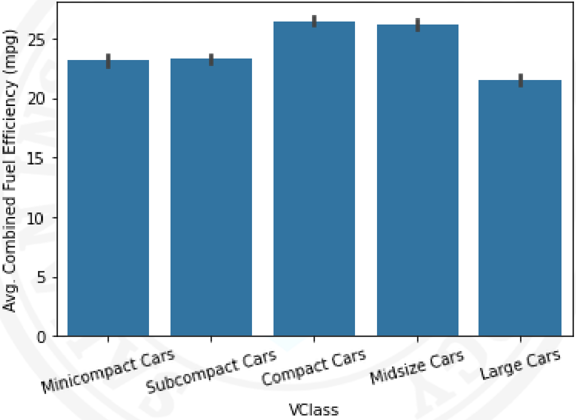
```
# Preparatory Step
fuel_econ = pd.read_csv('fuel_econ.csv')
# Convert the "VClass" column from a plain object type into an
                               #ordered categorical type
sedan_classes = ['Minicompact Cars', 'Subcompact Cars', 'Compact Cars',
                                   'Midsize Cars', 'Large Cars']
vclasses = pd.api.types.CategoricalDtype(ordered=True, categories=sedan_classes)
fuel_econ['VClass'] = fuel_econ['VClass'].astype(vclasses);
bin_edges = np.arange(12, 58+2, 2)
# Try experimenting with dynamic bin edges
# bin_edges = np.arange(-3, fuel_econ['comb'].max()+1/3, 1/3)
g = sb.FacetGrid(data = fuel_econ, col = 'VClass', col_wrap=3, sharey=False)
g.map(plt.hist, 'comb', bins = bin_edges);
```


VClass = Minicompact Cars
0
20
40
60
80
100
VClass = Subcompact Cars
0
20
40
60
80
100
120
140
VClass = Compact Cars
0
50
100
150
200
20
40
60
comb
VClass = Midsize Cars
0
25
50
75
100
125
150
20
40
60
comb
VClass = Large Cars
0
20
40
60
80
100
120
140
20
40
60
comb

Adapted Bar Charts

- Histograms and bar charts were introduced in the previous lesson as depicting the distribution of numeric and categorical variables, respectively, with the height (or length) of bars indicating the number of data points that fell within each bar's range of values.
- These plots can be adapted for use as bivariate plots by, instead of indicating count by height, indicating a mean or other statistic on a second variable.
- For example, we could plot a numeric variable against a categorical variable by adapting a bar chart so that its bar heights indicate the mean of the numeric variable. This is the purpose of seaborn's barplot function:

```
base_color = sb.color_palette()[0]
sb.barplot(data=fuel_econ, x='VClass', y='comb', color=base_color)
plt.xticks(rotation=15);
plt.ylabel('Avg. Combined Fuel Efficiency (mpg)')
```



Line Plots

- The line plot is a fairly common plot type that is used to plot the trend of one numeric variable against the values of a second variable.
- In contrast to a scatterplot, where all data points are plotted, in a line plot, only one point is plotted for every unique x-value or bin of x-values (like a histogram).
- If there are multiple observations in an x-bin, then the y-value of the point plotted in the line plot will be a summary statistic (like mean or median) of the data in the bin.
- The plotted points are connected with a line that emphasizes the sequential or connected nature of the x-values.
- If the x-variable represents time, then a line plot of the data is frequently known as a time series plot. For example, we have only one observation per time period, like in stock or currency charts.

Multivariate Exploration of Data

What is Multivariate Exploration of Data?

- A multivariate visualization is the visualiziation of three or more variables
- Multivariate analysis:- is performed to understand interactions between different fields in the dataset (or) finding interactions between variables more than 2. Ex:- Pair plot and 3D scatter plot.
- Multivariate analysis is required when more than two variables have to be analyzed simultaneously.
- It is a tremendously hard task for the human brain to visualize a relationship among 4 variables in a graph and thus multivariate analysis is used to study more complex sets of data.

Non-Positional Encodings for Third Variables

- There are four major cases to consider when we want to plot three variables together:
  1. Three numeric variables
  2. Two numeric variables and one categorical variable
  3. One numeric variable and two categorical variables
  4. Three categorical variables
- A numerical variable is a variable where the value has meaning (i.e., weight or age), but a value such as a phone number doesn't have meaning in the numbers alone.
- A categorical variable is a variable that holds a type (i.e., species or hair color).

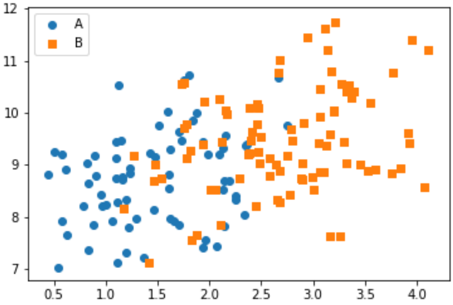
Continue...

- If we have at least two numeric variables, as in the first two cases, one common method for depicting the data is by using a scatterplot to encode two of the numeric variables, then using a non-positional encoding on the points to convey the value on the third variable, whether numeric or categorical.
- Three main non-positional encodings stand out:
  - Shape
  - Size
  - Color
- For Matplotlib and Seaborn, color is the easiest of these three encodings to apply for a third variable.
- Color can be used to encode both qualitative and quantitative data, with different types of color palettes used for different use cases.

Encoding via Shape

- Shape is a good encoding for categorical variables, using one shape for each level of the categorical variable.
- Unfortunately, there is no built-in way to automatically assign different shapes in a single call of the scatter or regplot function.
- Instead, we need to write a loop to call our plotting function multiple times, isolating data points by categorical level and setting a different ”marker” argument value for each one.

```
cat_markers = [['A', 'o'],
               ['B', 's']]
for cat, marker in cat_markers:
    df_cat = df[df['cat_var1'] == cat]
    plt.scatter(data = df_cat, x = 'num_var1', y = 'num_var2', marker = marker)
plt.legend(['A','B'])
```

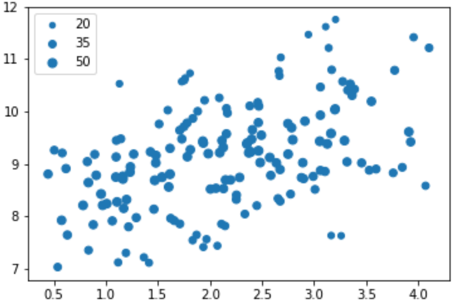


- From the positional encodings in the plot, you can see that there is a modest positive relationship between the two numeric variables.
- Adding the categorical variable via shape encoding, we can see that points of category 'A' tend to be smaller than those of category 'B' in terms of the numeric x-variable ("num_var1").

Encoding via Size

- Point size is a good encoding for numeric variables.
- Usually, we want the numeric values to be proportional to the area of the point markers; this is the default functionality of the ”s” parameter in scatter.

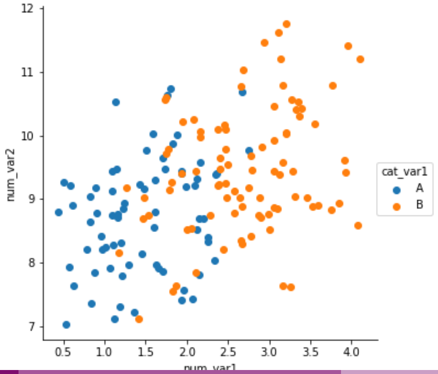
```
plt.scatter(data = df, x = 'num_var1', y = 'num_var2', s = 'num_var3')
# dummy series for adding legend
sizes = [20, 35, 50]
base_color = sb.color_palette()[0]
legend_obj = []
for s in sizes:
   legend_obj.append(plt.scatter([], [], s = s, color = base_color))
plt.legend(legend_obj, sizes)
```



- The size encoding for the third numeric variable (”num_var3”) shows that its values are largest in the ’middle’ of the distribution of values, and smaller on the upper and bottom edges.
- It is also clear that size is much less precise at encoding than position, so it is better used to make general qualitative judgments than precise judgments.

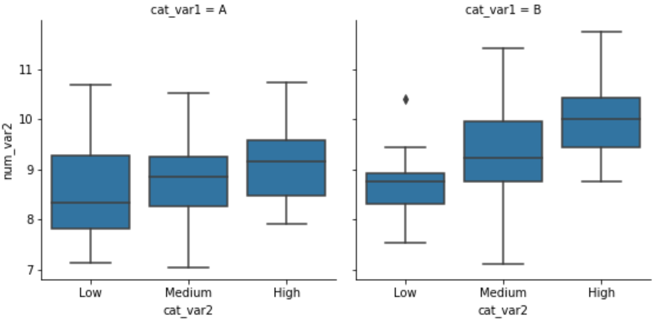
Color Palettes

- Color is a very common encoding for variables, for both qualitative and quantitative variables.
- If you have a qualitative variable, you can set different colors for different levels of a categorical variable through the "hue" parameter on seaborn's FacetGrid class.



Faceting for Multivariate Data

- In the previous slides, we saw how FacetGrid could be used to subset your dataset across levels of a categorical variable, and then create one plot for each subset.
- Where the faceted plots demonstrated were univariate before, you can actually use any plot type, allowing you to facet bivariate plots to create a multivariate visualization.



Feature Engineering

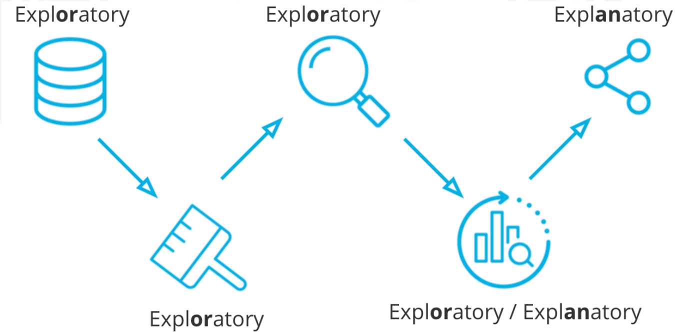
- Feature engineering is a tool that you can leverage as you explore and learn about your data.
- As you explore a dataset, you might find that two variables are related in some way.
- Feature engineering is all about creating a new variable with a sum, difference, product, or ratio between those original variables that may lend a better insight into the research questions you seek to answer.
- For example, if you have one variable that gives a count of crime incidents, and a second one that gives population totals, then you may want to engineer a new variable by dividing the former by the latter, obtaining an incident rate.
- This would account for a possible relationship between the original features where if there are more people, there might naturally be more chances for crimes to occur.

Explanatory Visualizations

Explanatory Analysis

- As we have discussed before, there are two main reasons for creating data visualizations; exploratory analysis and explanatory analysis.
- Exploratory analysis is done when you are searching for insights in your data. So, Exploratory analysis is primarily about searching for insights in the data.
- Explanatory analysis is done when you communicate your findings to others. So, explanatory analysis is primarily about conveying insights to others.
- As a reminder, let's briefly review the data analysis process and revisit the way that exploratory and explanatory visualizations fit into different parts of that process.

- The five steps of the data analysis process are:
  1. **Extract** - Obtain the data from a spreadsheet, SQL, the web, etc.
  2. **Clean** - Here we could use exploratory visuals.
  3. **Explore** - Here we use exploratory visuals.
  4. **Analyze** - Here we might use either exploratory or explanatory visuals.
  5. **Share** - Here is where explanatory visuals live.



Tell A Story

- Stories have the power of captivating in a way that facts just cannot compete.
- When you are presenting your findings, make sure to tell a story.
- Telling stories with data follows these steps:
  1. Start with a Question
  2. Repetition is a Good Thing
  3. Highlight the Answer
  4. Call Your Audience To Action

Polishing Plots

- In order to convey your findings to others quickly and efficiently, you'll need to put work into polishing your plots.
- There are many dimensions to consider when putting together a polished plot.
  - **Choose an appropriate plot**: Your choice of plot will depend on the number of variables that you have and their types: nominal, ordinal, discrete numeric, or continuous. Choice of plot also depends on the specific relationship that you want to convey.
  - **Choose appropriate encodings**: Your variables should impact not just the type of plot that is chosen, but also the variable encodings. For example, if you have three numeric variables, you shouldn't just assign x-position, y-position, and color encodings randomly.

Continue...

- **Pay attention to design integrity**: When setting up your plotting parameters, remember the design principles from the previous slides. Make sure that you minimize chart junk and maximize the data-ink ratio, as far as it maintains good interpretability of the data. When deciding whether or not to add non-positional encodings, make sure that they are meaningful.
    - For example, using color in a frequency bar chart may not be necessary on its own, but will be useful if those colors are used again later in the same presentation, matched with their original groups. By the same token, avoid using the same color scheme for different variables to minimize the chance of reader confusion.
- You should also ensure that your plot avoids lie factors as much as possible.

- **Label axes and choose appropriate tick marks**: For your positional axes, make sure you include axis labels. This is less important in exploration when you have the code available and have an extended flow to your work, but when you're conveying only the key pieces to others, it's crucial.
- When you add an axis label, make sure you also provide the units of measurement, if applicable (e.g., stating "Height (cm)" rather than just "Height").
- As for tick marks, you should include at least three tick marks on each axis. This is especially important for data that has been transformed: you want enough tick marks so that the scale of the data can be communicated there.
- If your values are very large or very small numbers, consider using abbreviations to relabel the ticks (e.g., use "250K" instead of "250000").

Continue...

- **Provide legends for non-positional variables**: Make sure that you add a legend for variables not depicted on the axes of your plot. For color encoding, you can add a color bar to the side of the plot. The most important new thing here is that you provide a descriptive label to your legend or color bar, just as you would the axes of your plot.
- **Title your plot and include descriptive comments**: Finally, make sure that you provide a descriptive title to your plot. If this is a key plot that presents important findings to others, aim to create a title that draws attention to those main points, rather than just state what variables are plotted.
- Also, realize that while a visualization might be the core mechanism by which you convey findings, it need not stand alone.

-