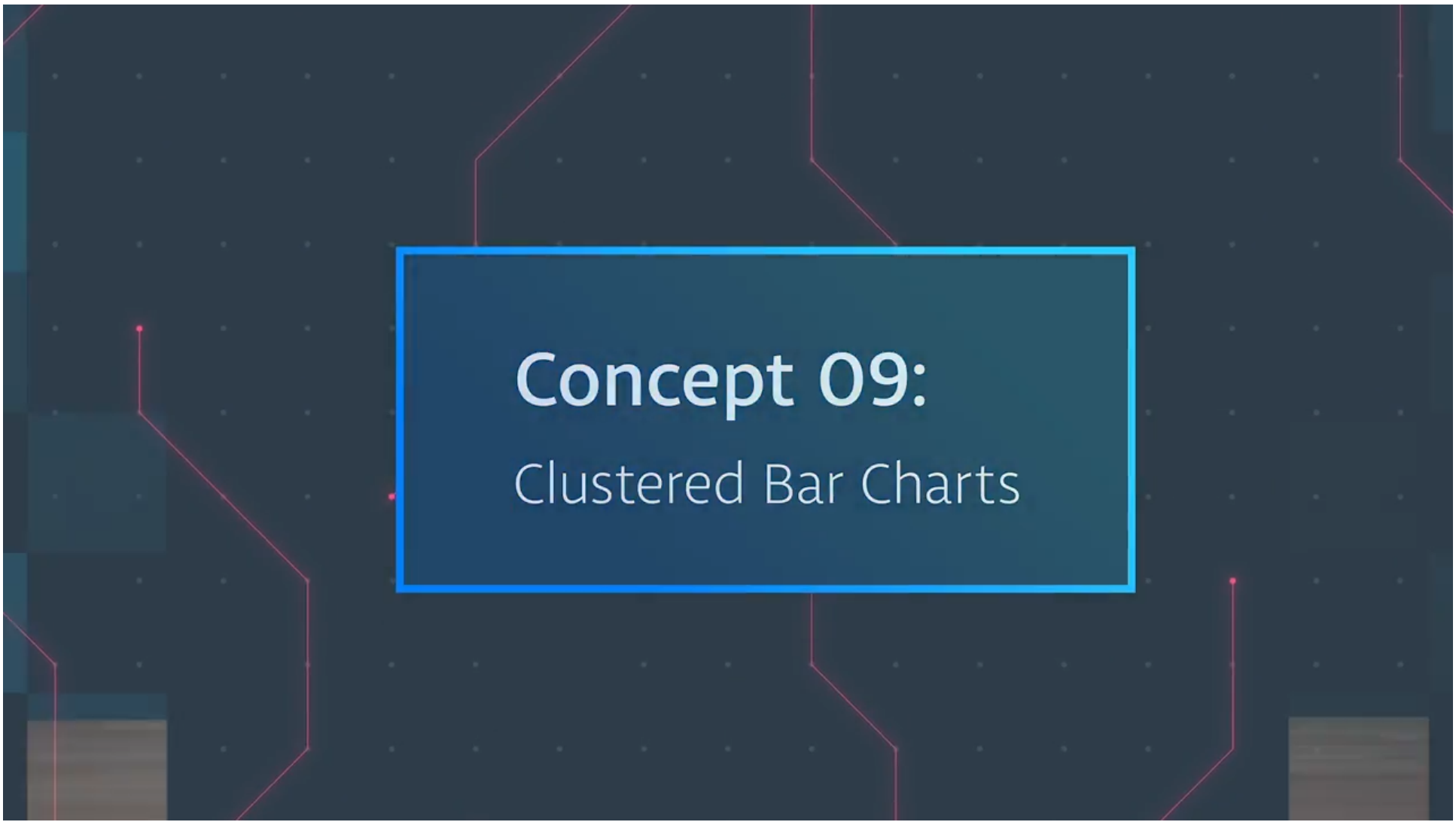
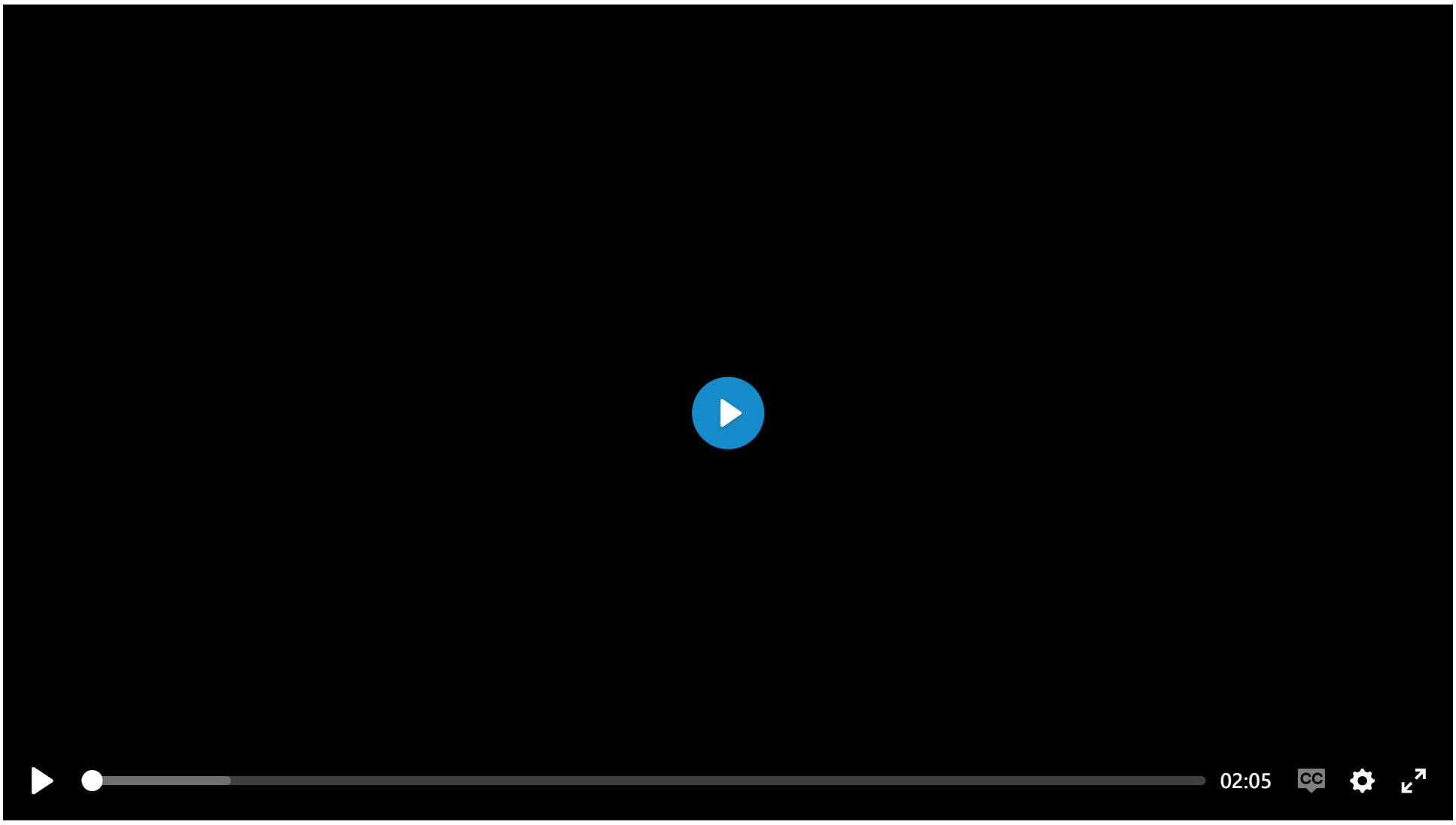
09. Clustered Bar Charts

L4 091 Clustered Bar Charts V4



Data Vis L4 C09 V1



Clustered Bar Charts

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. 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:

Example 1. Plot a Bar chart between two qualitative variables

Preparatory Step 1 - Convert the "VClass" column from a plain object type into an ordered 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
# Refer - https://pandas.pydata.org/pandas-
docs/version/0.23.4/generated/pandas.api.types.CategoricalDtype.html
vclasses = pd.api.types.CategoricalDtype(ordered=True, categories=sedan_classes)

# Use pandas.astype() to convert the "VClass" column from a plain object type into an ordered categorical
type
fuel_econ['VClass'] = fuel_econ['VClass'].astype(vclasses);
```

Preparatory Step 2 - Add a new column for transmission type - Automatic or Manual

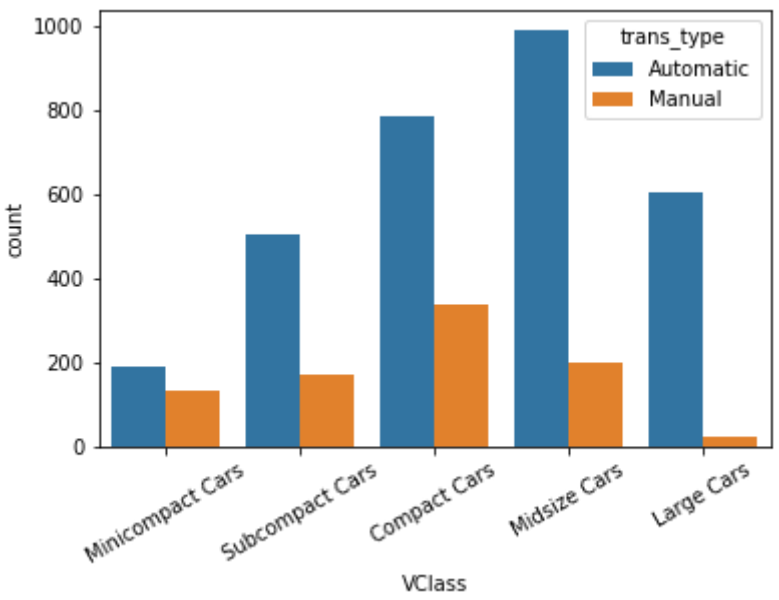
```
# The existing `trans` column has multiple sub-types of Automatic and Manual.
# But, we need plain two types, either Automatic or Manual. Therefore, add a new column.

# The Series.apply() method invokes the `lambda` function on each value of `trans` column.
# In python, a `lambda` function is an anonymous function that can have only one expression.
fuel_econ['trans_type'] = fuel_econ['trans'].apply(lambda x:x.split()[0])
fuel_econ.head()
```

year	VClass	drive	trans	fuelType	cylinders	displ	...	pv4	city	UCity	highway	UHighway	comb	co2	feScore	ghgScore	trans_type
2013	Subcompact Cars	All-Wheel Drive	Automatic (AM6)	Premium Gasoline	6	3.8	...	0	16.4596	20.2988	22.5568	30.1798	18.7389	471	4	4	Automatic
2013	Compact Cars	Front-Wheel Drive	Automatic (AM-S6)	Premium Gasoline	4	2.0	...	0	21.8706	26.9770	31.0367	42.4936	25.2227	349	6	6	Automatic
2013	Compact Cars	Front-Wheel Drive	Automatic (S6)	Premium Gasoline	6	3.6	...	0	17.4935	21.2000	26.5716	35.1000	20.6716	429	5	5	Automatic
2013	Compact Cars	All-Wheel Drive	Automatic (S6)	Premium Gasoline	6	3.6	...	0	16.9415	20.5000	25.2190	33.5000	19.8774	446	5	5	Automatic
2013	Midsize Cars	Front-Wheel Drive	Automatic (S6)	Regular Gasoline	4	2.4	...	95	24.7726	31.9796	35.5340	51.8816	28.6813	310	8	8	Automatic

DataFrame after adding a new column `trans_type`

Step 3. Plot the bar chart

```
sb.countplot(data = fuel_econ, x = 'VClass', hue = 'trans_type')
```



Bar chart between two qualitative variables, and one of them is ordered.

Alternative Approach

Example 2. Plot a Heat Map between two qualitative variables

One alternative way of depicting the relationship between two categorical variables is through a heat map. Heat maps were introduced earlier as the 2-D version of a histogram; here, we're using them as the 2-D version of a bar chart. The seaborn function heatmap() is at home with this type of heat map implementation, but the input arguments are unlike most of the visualization functions that have been introduced in this course. Instead of providing the original dataframe, we need to summarize the counts into a matrix that will then be plotted.

Step 1 - Get the data into desirable format - a DataFrame

```
# Use group_by() and size() to get the number of cars and each combination of the two variable levels as a
pandas Series
ct_counts = fuel_econ.groupby(['VClass', 'trans_type']).size()
```

```
VClass              trans_type
Minicompact Cars    Automatic      188
                    Manual         133
Subcompact Cars     Automatic      502
                    Manual         171
Compact Cars        Automatic      784
                    Manual         338
Midsize Cars        Automatic      989
                    Manual         199
Large Cars          Automatic      605
                    Manual          20
dtype: int64
```

Number of cars in each vehicle type and transmission combination

```
# Use Series.reset_index() to convert a series into a dataframe object
ct_counts = ct_counts.reset_index(name='count')
```

	VClass	trans_type	count
0	Minicompact Cars	Automatic	188
1	Minicompact Cars	Manual	133
2	Subcompact Cars	Automatic	502
3	Subcompact Cars	Manual	171
4	Compact Cars	Automatic	784
5	Compact Cars	Manual	338
6	Midsize Cars	Automatic	989
7	Midsize Cars	Manual	199
8	Large Cars	Automatic	605
9	Large Cars	Manual	20

A DataFrame object created from the Series generated in the step above

trans_type VClass	Automatic	Manual
Minicompact Cars	188	133
Subcompact Cars	502	171
Compact Cars	784	338
Midsize Cars	989	199
Large Cars	605	20

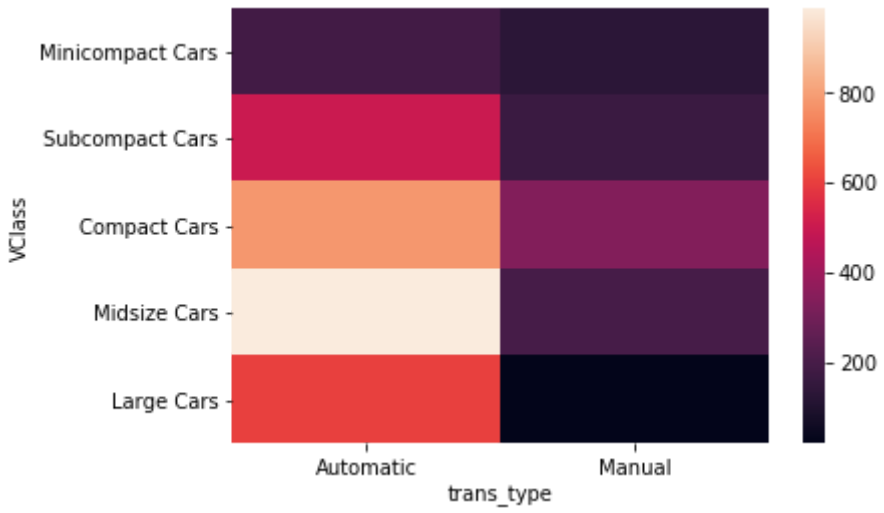
The DataFrame to plot on heatmap

```
# Use DataFrame.pivot() to rearrange the data, to have vehicle class on rows
ct_counts = ct_counts.pivot(index = 'VClass', columns = 'trans_type', values = 'count')
```

Documentation: Series `reset_index`, DataFrame `pivot`

Step 2 - Plot the heatmap

```
sb.heatmap(ct_counts)
```

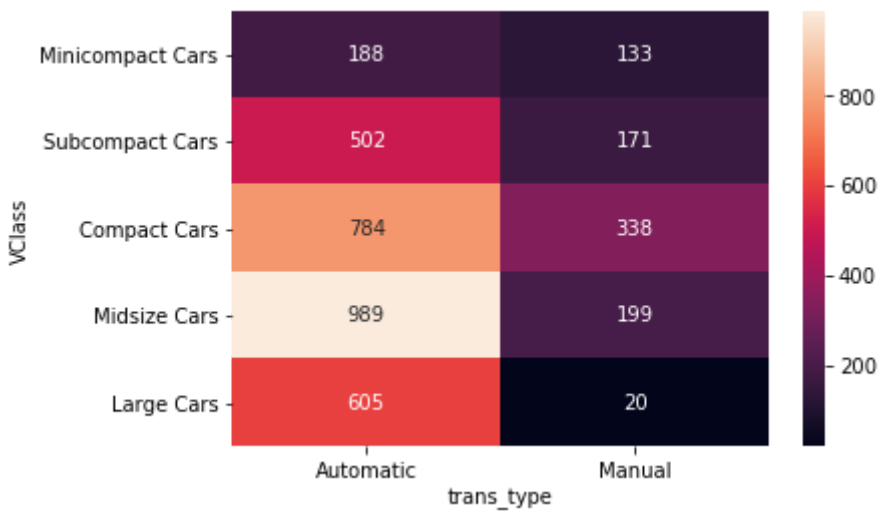


The heat map tells the same story as the clustered bar chart.

Example 3. Additional Variation

```
sb.heatmap(ct_counts, annot = True, fmt = 'd')
```

`annot = True` makes it so annotations show up in each cell, but the default string formatting only goes to two digits of precision. Adding `fmt = 'd'` means that annotations will all be formatted as integers instead. You can use `fmt = '.0f'` if you have any cells with no counts, in order to account for NaNs.



Next Concept