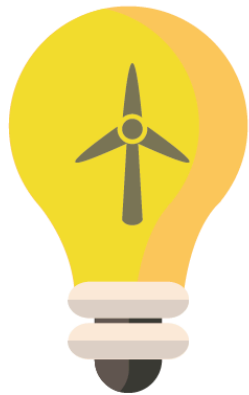


# BENV0091 Lecture 3: Introduction to Supervised Learning

Patrick de Mars



# Lecture Overview

1. Tidy Data
2. Using Facets
3. Supervised Learning

# Setup

- Create a new project as usual: lecture3
- Download the energy consumption data CSV file from Moodle
- Open a new Markdown file and save it as pivot.Rmd

# Tidy Data

- R and especially the tidyverse packages like **tidy data**
- In tidy data:
  - Every column is a variable
  - Every row is an observation
  - Every cell is a single value
- Tidy data is easier to plot!
- Most commonly, untidy data is in **wide format**, and should be converted to **long format**

*This data below gives the height of 2 people over 3 years. It is in wide format: a variable (year) is spread across columns – not tidy and hard to plot!*

```
# A tibble: 2 × 4
  name   `2016` `2017` `2018`
  <chr>   <dbl>   <dbl>   <dbl>
1 andy    160     165     167
2 bella   170     180     182
```

*The data has been tidied using `pivot_longer()` below*

```
# A tibble: 6 × 3
  name   year height
  <chr> <int>   <dbl>
1 andy   2016    160
2 andy   2017    165
3 andy   2018    167
4 bella  2016    170
5 bella  2017    180
6 bella  2018    182
```

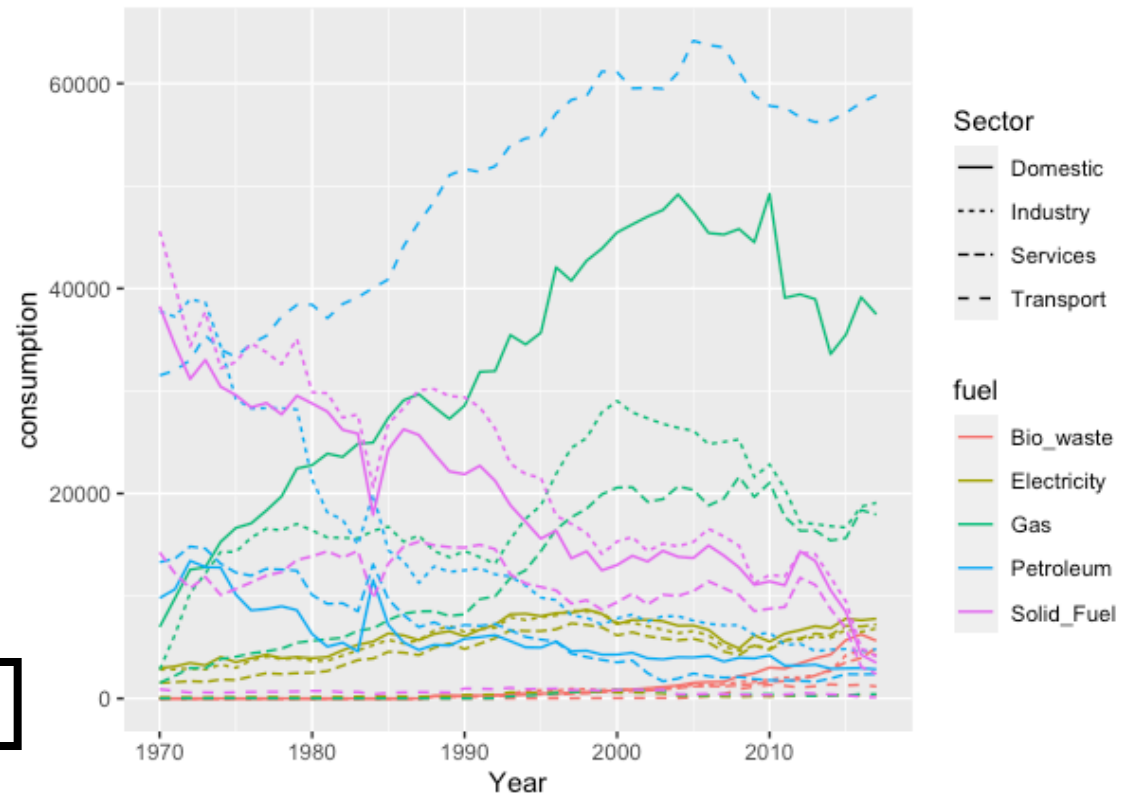
# Pivoting

- Read the energy consumption data, assigning to an object named `energy`
- How should we transform the data to create the plot on the right?
- Task: use `pivot_longer()` to make the energy data tidy
- Task: create the plot on the right with `ggplot2` and `geom_line()`

*Use the `linetype` aesthetic*

## Original data

	Year	Solid_Fuel	Petroleum	Gas	Bio_waste	Electricity	Sector
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	1970	45573.	37758.	2808.	0	2820.	Industry
2	1970	899.	31515.	1.23	0	107.	Transport
3	1970	38262.	9798.	6979.	0	2976.	Domestic
4	1970	14260.	13296.	1511.	0	1532.	Services
5	1971	40284.	37250.	6219.	0	2846.	Industry
6	1971	745.	31998.	8.57	0	107.	Transport



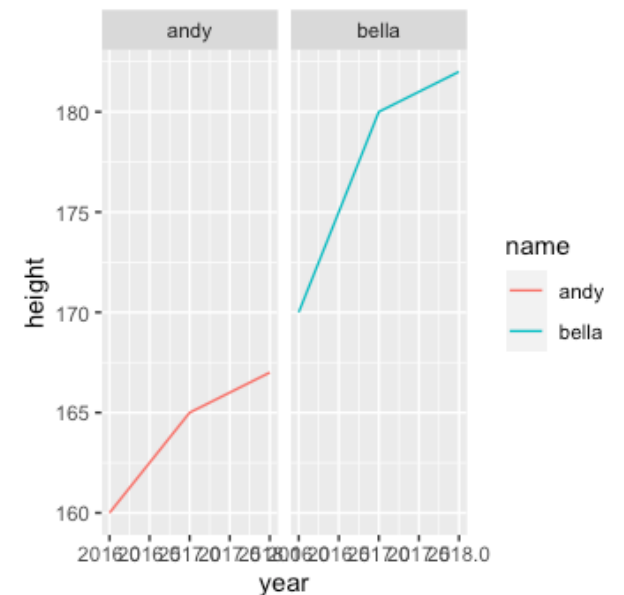
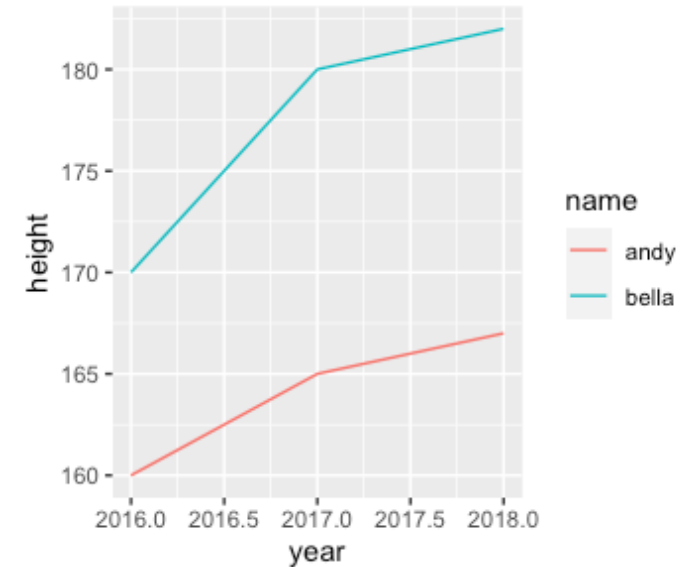
# Pivot Wider

- `pivot_wider()` does the opposite of `pivot_longer()`
- You won't need it so often but it's good to know
- Task: use `pivot_wider()` to return the pivoted energy data frame back to its original
- Task: use `pivot_wider()` to create a new data frame with the following columns: Sector, fuel, 1970, 1971,...2017 (see below)

	Sector	fuel	`1970`	`1971`	`1972`	`1973`	`1974`	`1975`	`1976`	`1977`	`1978`	`1979`	`1980`
	<i>&lt;chr&gt;</i>	<i>&lt;chr&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>	<i>&lt;dbl&gt;</i>
1	Indust...	Soli...	45573.	40284.	34345.	37748.	32206.	32822.	34644.	33783.	32615.	35081.	29877.
2	Indust...	Petr...	37758.	37250.	38944.	38626.	34362.	29229.	28290.	28333.	28332.	28197.	21386.
3	Indust...	Gas	2808.	6219.	10297.	12204.	14297.	14315.	15685.	16569.	16402.	17029.	16387.
4	Indust...	Bio_...	0	0	0	0	0	0	0	0	0	0	0
5	Indust...	Elec...	2820.	2846.	2925.	2840.	3311.	2994.	3731.	4084.	3847.	3944.	3648.

# Facets

- The line plot we made of energy consumption by fuel and sector was pretty busy
- There are sometimes advantages to plotting variables on different **panels**
- We can use `facet_grid()` and `facet_wrap()` to do this
- `facet_grid()` is used to plot the interaction **two variables** across rows and columns
- `facet_wrap()` splits the plot across **one variable**



*What happened here?*

# Facets: Wrap

- Task: use `facet_wrap()` to create a bar plot for each sector (see right)
- Task: use the ``scales`` argument to allow each panel to be freely scaled by consumption (each panel has its own y-axis limits)
- Task: create a bar plot of consumption vs. year, coloured by Sector, faceted by fuel



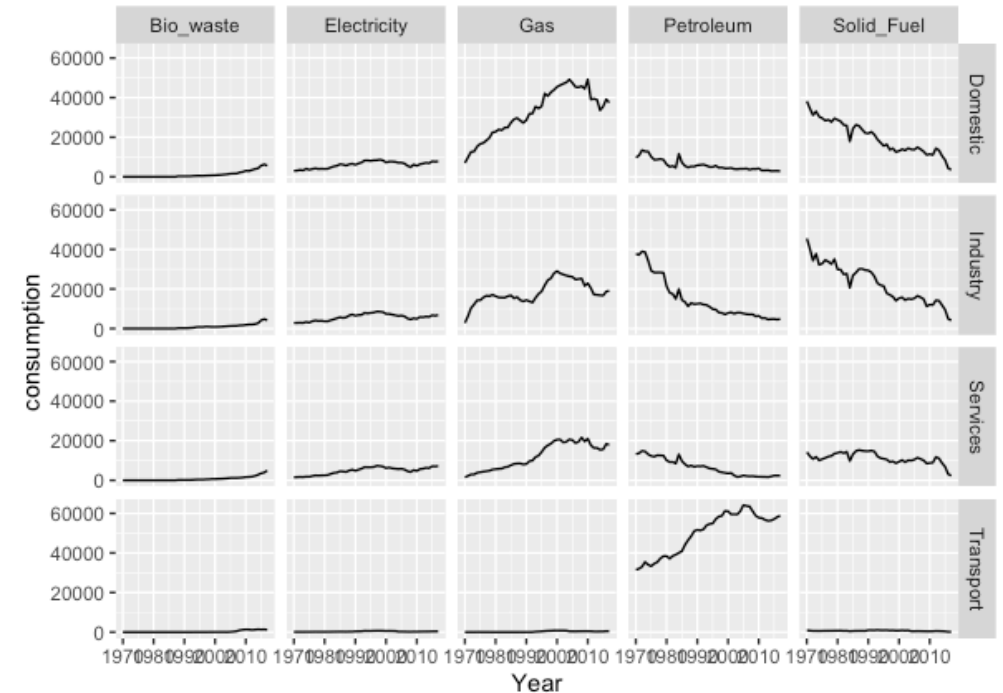
*Use `ggplot(data) + ... + facet_wrap(~var)` to create a panel for each value of var*

*You will need `geom_col()` and the `fill` aesthetic to colour bars*



# Facets: Grid

- Suppose we want to disaggregate the plot further, seeing each fuel and sector combination separately
- Task: use `facet_grid()` to create a line plot for each fuel and sector (see right)
- Task: look carefully at what happens when you set the `scales` argument to “free”
- Task: replace `facet_grid()` with `facet_wrap()` and set the `y` scale to be free



*Use `facet_wrap(var1 ~ var2)`  
to spread `var1` across rows  
and `var2` across columns*

# Supervised Learning

# The Task of Supervised Learning

- Supervised learning is the task of fitting a **model** (function) that maps inputs to outputs based on **labelled examples**
- The most common purposes of a supervised learning model are:
  - To predict things we don't have data for:
    - What will the temperature be tomorrow? (**Regression**)
    - What does this image show? (**Classification**)
  - To better understand or quantify (possibly causal) relationships between variables:
    - What are the drivers of fuel poverty?
- At a high level, both purposes use the same methods, but some models are less **interpretable** or have better **predictive power** than others

# Modelling MPG

- We will fit two models on the `mpg` dataset from ggplot2
- We will predict the highway MPG (`hwy`) with the following variables: **displ**, **cyl**, **drv**, **class**
- We will try a **linear regression model** and a **decision tree**
- Task: create a new data frame `df` which has only the following columns:
  - hwy
  - displ
  - cyl
  - drv

# Splitting Data

- The models will be trained on a subset of the data (**training set**), and tested on the remainder (**testing set**) to determine how well they generalise to new data and avoid **overfitting**
- Task: split the data into `train` (75%) and `test` (25%) sets:
  - There are several methods: this one uses `sample\_frac()` and `anti\_join()`:

```
set.seed(123)
```

```
train <- sample_frac(df, 0.75) # Take 75% of the rows from df  
test  <- anti_join(df, train) # All rows in df that are NOT in train
```

*Use `set.seed(x)` to set the **random seed**: this makes your results reproducible*

*Use `sample_frac(df, x/100)` to sample  $x\%$  of rows randomly from `df`*

# Dummy Variables

- When dealing with categorical variables (factors), we need to convert them to something which we can deal with numerically
- `drv` has 3 classes: `f` (front wheel drive); `r` (rear WD); `4` (4 WD)
- We could naively assign these to 3 values (e.g. 1, 2, 3) but this assumes an ordinal characteristic which is not there!
- Dummy variables spread  $N$  classes across  $N-1$  binary variables, with a 1 indicating the class
- R makes dummy variables **automatically** when fitting a model

*Wrong way!*

drv		drv
f	→	1
r		2
4		3



*Right way!*

drv		drvf	drvrr
f	→	1	0
r		0	1
4		0	0



*Note that `4` doesn't have a column: it is the case where all other values are 0*

# Formulae

- To fit a model, we need to specify a **formula**
- From the modelr package, you can use `model_matrix()` to see the model equation in matrix form
- This will also show you how dummy variables are going to be created

## *Explicit formula*

```
hwy ~ displ + cyl + drv + class
```

## *Predict hwy with everything else*

```
hwy ~ .
```

*modelr::model\_matrix(df, formula)  
returns the model equation explicitly  
in matrix form*

	<code>`(Intercept)`</code>	<code>displ</code>	<code>cyl</code>	<code>drvf</code>	<code>drv</code>	<code>classcompact</code>	<code>classmidsize</code>	<code>classminivan</code>	<code>classpickup</code>
	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>	<code>&lt;dbl&gt;</code>
1	1	1.8	4	1	0	1	0	0	0
2	1	1.8	4	1	0	1	0	0	0
3	1	2	4	1	0	1	0	0	0
4	1	2	4	1	0	1	0	0	0
5	1	2.8	6	1	0	1	0	0	0
6	1	2.8	6	1	0	1	0	0	0

# Linear Model

- Linear regression models have the form:

The diagram shows the linear regression equation  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + b_N x_N + \epsilon$ . Arrows point from labels to specific parts of the equation: 'Dependent variable' points to  $y$ ; 'Intercept' points to  $\beta_0$ ; 'Coefficient' points to  $\beta_2$ ; 'Independent variable' points to  $x_N$ ; and 'Noise/residual' points to  $\epsilon$ .

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + b_N x_N + \epsilon$$

Labels and arrows:

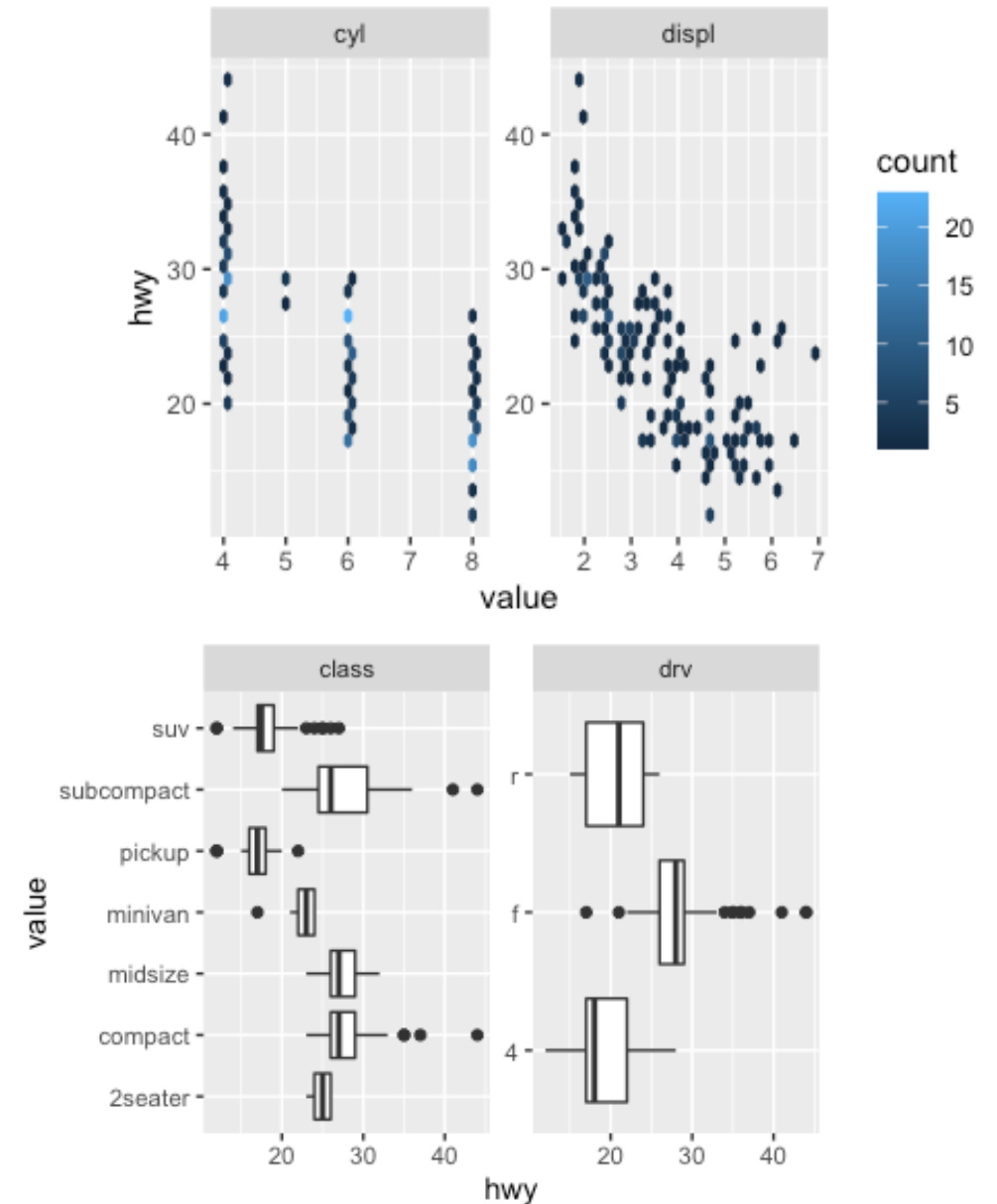
- Dependent variable points to  $y$ .
- Intercept points to  $\beta_0$ .
- Coefficient points to  $\beta_2$ .
- Independent variable points to  $x_N$ .
- Noise/residual points to  $\epsilon$ .



# Exploratory Analysis

- Before we fit the models, it's useful to have some intuition about what we expect to find
- Task: reproduce the right hand plot using `facet_wrap()`
- Given the plots on the right, what coefficients do you expect from the model?

*Exploratory plots*



# Fitting the Linear Model

- We will now fit the linear model using the ``lm()`` function (which uses **ordinary least squares**)
- Task: fit the model **to the training data**
- We can use ``summary(model)`` to get some useful information about the model:
  - Summary statistics about the residuals
  - Coefficients and p-values
  - $R^2$  value
- From the broom package, ``tidy()`` can be used to summarise information about a model object in a tidy data frame
- How do the coefficients and p-values confirm/challenge your expectations?
- What can you say about the estimated MPG of 4 wheel-drive cars?

```
linear_model <- lm(hwy ~ ., data = train)
```

*broom::tidy(model) summarises a linear model in a tidy data frame*

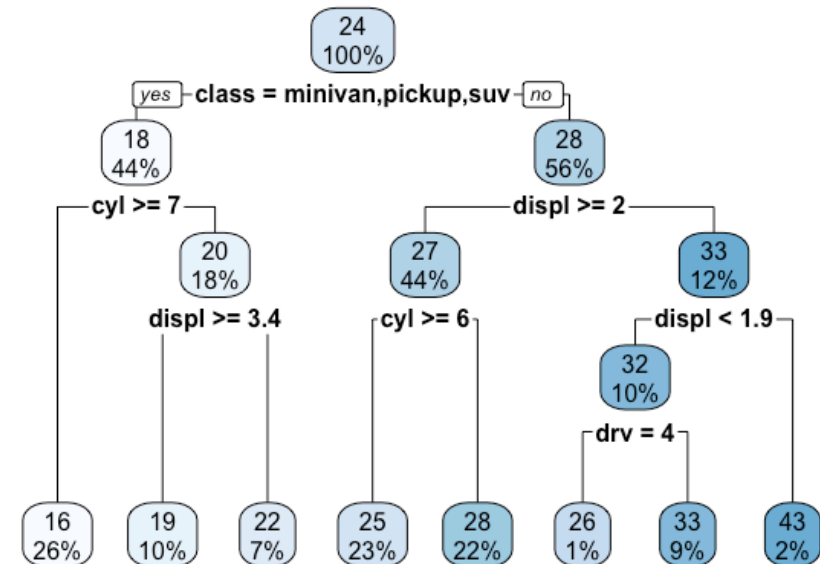
```
# A tibble: 11 × 5
```

	term <chr>	estimate <dbl>	std.error <dbl>	statistic <dbl>	p.value <dbl>
1	(Intercept)	37.0	2.08	17.8	3.21e-40
2	displ	-0.417	0.564	-0.739	4.61e- 1
3	cyl	-1.36	0.367	-3.71	2.79e- 4
4	drvf	3.89	0.748	5.20	5.71e- 7
5	drvr	1.39	0.886	1.57	1.19e- 1
6	classcompact	-4.19	1.75	-2.40	1.77e- 2
7	classmidsize	-4.93	1.73	-2.85	4.89e- 3
8	classminivan	-9.29	1.84	-5.05	1.15e- 6
9	classpickup	-8.71	1.67	-5.23	5.14e- 7
10	classsubcompact	-3.52	1.70	-2.07	4.00e- 2
11	classsuv	-8.04	1.55	-5.19	6.07e- 7

# Fitting the Decision Tree

- The mechanics of fitting a model can generally be applied to several model types
- Fitting a decision tree with `rpart()` uses the same syntax as `lm()`, although the other arguments are more important too:
- Task: fit a decision tree to the model with `rpart()` and:
  - `minsplit = 10`
  - `minbucket = 2`
  - `cp = 0.01`

*Use `rpart(formula, df)` to fit a decision tree for formula, using data df*



*Plot a decision tree with  
`rpart.plot(dt)`*

# Making Predictions

- In R, model objects can be operated on with similar functions:
  - `predict()` to make predictions
  - `summary()` to see summary statistics
- A fitted model object can be applied to make predictions using ``predict(model, data)``
- Task: use `predict()` to retrieve the predictions for the linear model and decision tree on the training data
- Task: use `mutate()` to add two columns to the ``train`` data frame with the linear model and decision tree predictions (like below)

## ***Actual***

hwy	displ	cyl	drv	class
<int>	<dbl>	<int>	<fct>	<fct>
25	5.3	8	f	midsize
20	4	6	4	pickup
17	4.7	8	4	suv
25	3.1	6	4	compact

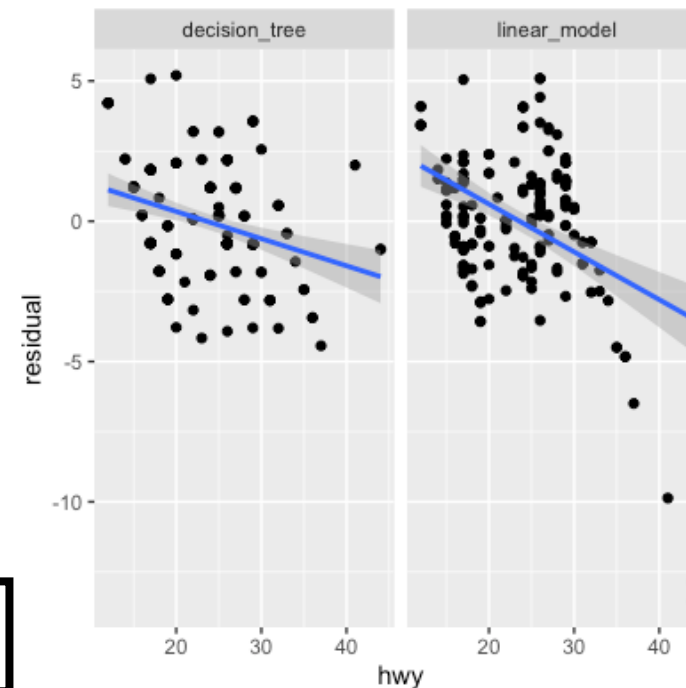
## ***Predictions***

linear_model	decision_tree
<dbl>	<dbl>
22.8	25.2
18.4	18.8
16.1	16.2
23.3	25.2

# Plotting Residuals

- It is important to understand the **residuals** (difference between predicted and actual values) of your model
- Task: use `pivot_longer()` to make `train` a tidy data frame like the one on the right
- Task: add a `residual` column to `train`  $\text{predicted} - \text{hwy}$
- Task: produce a scatterplot of residuals vs. hwy

hwy	displ	cyl	drv	class	model	predicted
<int>	<dbl>	<int>	<fct>	<fct>	<chr>	<dbl>
25	5.3	8	f	midsize	linear_model	22.8
25	5.3	8	f	midsize	decision_tree	25.2
20	4	6	4	pickup	linear_model	18.4
20	4	6	4	pickup	decision_tree	18.8
17	4.7	8	4	suv	linear_model	16.1
17	4.7	8	4	suv	decision_tree	16.2



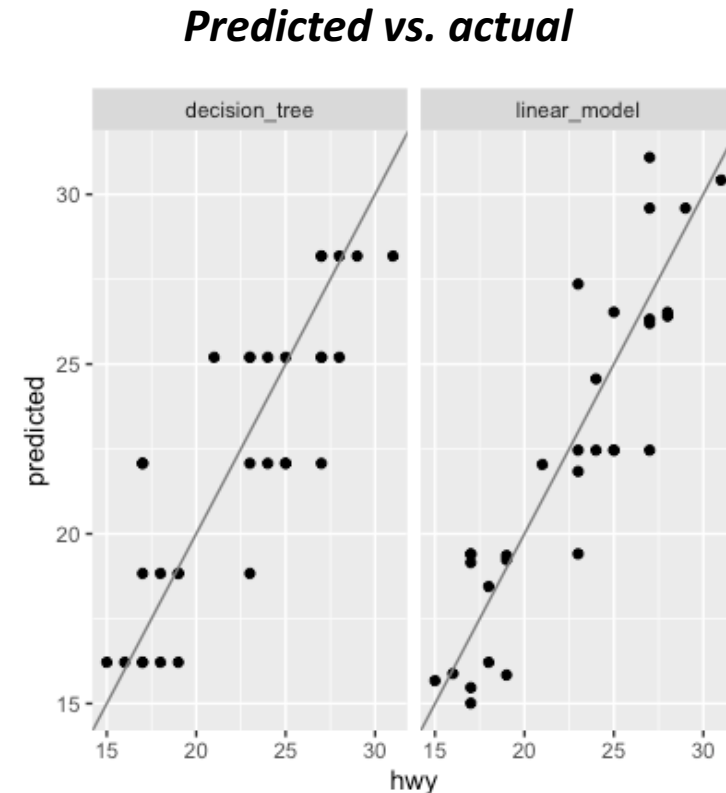
*Residuals are heteroskedastic*

Use `geom_smooth(method='lm')`  
to add the linear fit

Use `facet_wrap(~model)` to add a  
panel for each model

# Making Predictions on Test Data

- Now we will make predictions on the unseen test data and see which one has the best accuracy
- Task: add two columns to the `test` data frame giving predictions from linear model and decision tree using `predict()` (like we did for `train`)
- Task: pivot the data frame, creating a `model` column and `predicted` column
- Task: plot predicted versus actual values (as in the right hand plot)



*The line  $y=x$  on the predicted vs. actual graph indicates perfect predictions*

# Accuracy Metrics (Regression)

- To compare the two models, we need a quantitative accuracy metric
- Common metrics for regression include:
  - Mean absolute error (MAE)
  - Mean squared error (MSE)
  - Root mean squared error (RMSE)
  - R2 (usually valid for linear models only)
- Task: write functions for MAE, RMSE and MAE
- Task: use `group_by()`, `summarise()`, and your accuracy functions to create a table of accuracy metrics for each model

*Accuracy metrics for regression*

Metric	Equation
Mean absolute error (MAE)	$\frac{1}{N} \sum_{i=1}^N  y_i - \hat{y}_i $
Mean squared error (MSE)	$\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$
Root mean squared error (RMSE)	$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$
R <sup>2</sup>	$1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$

$y_i$  := predicted value for observation  $i$

$\hat{y}_i$  := actual value for observation  $i$

$\bar{y}$  := mean of actual values

# Conclusions

- Which model was most accurate?
- Which model would you choose to use in practice?
- How confident would you be in each model to generalise to new data?
- What were the limitations of our approach?
- How could our method be improved? **Topics for next week!**



Mentimeter