

# week4\_lecture

July 13, 2020

## 1 Modules

To visualize data in Python, we will be using an external module called `plotnine`. To install `plotnine`, launch the "Anaconda Prompt (Anaconda3)" program and run the following command:  
`conda install -c conda-forge plotnine`

When importing `plotnine`, we will alias it as `p9`. We also need to import `numpy` and `pandas`, using the same aliases as before.

```
[1]: import plotnine as p9
import numpy as np
import pandas as pd
```

## 2 Basic visualization

With `plotnine`, you can build a plot using only 2 lines of code:

1. `ggplot()`: this creates a plot object, and holds the details of which variables will be displayed on which axes
2. a **geom**: this specifies the kind of plot that will be created

And that's it! This structure is called the grammar of graphics, and it is used by other plotting libraries in other languages, such as `ggplot` in R.

### 2.1 Input data

To keep things simple, we will use an artificial dataset to start with. We can make this with `pandas`, as covered in the previous session:

```
[2]: example = pd.DataFrame([
    {"City": "Manchester", "Year": 1980, "Rainfall": 200},
    {"City": "Manchester", "Year": 1990, "Rainfall": 190},
    {"City": "Manchester", "Year": 2000, "Rainfall": 160},
    {"City": "Manchester", "Year": 2010, "Rainfall": 170},
    {"City": "London", "Year": 1980, "Rainfall": 100},
    {"City": "London", "Year": 1990, "Rainfall": 90},
    {"City": "London", "Year": 2000, "Rainfall": 60},
```

```

    {"City":"London", "Year":2010, "Rainfall":70},
  ])
example

```

```

[2]:
      City Year Rainfall
0 Manchester 1980      200
1 Manchester 1990      190
2 Manchester 2000      160
3 Manchester 2010      170
4      London 1980      100
5      London 1990       90
6      London 2000       60
7      London 2010       70

```

## 2.2 Scatter plot

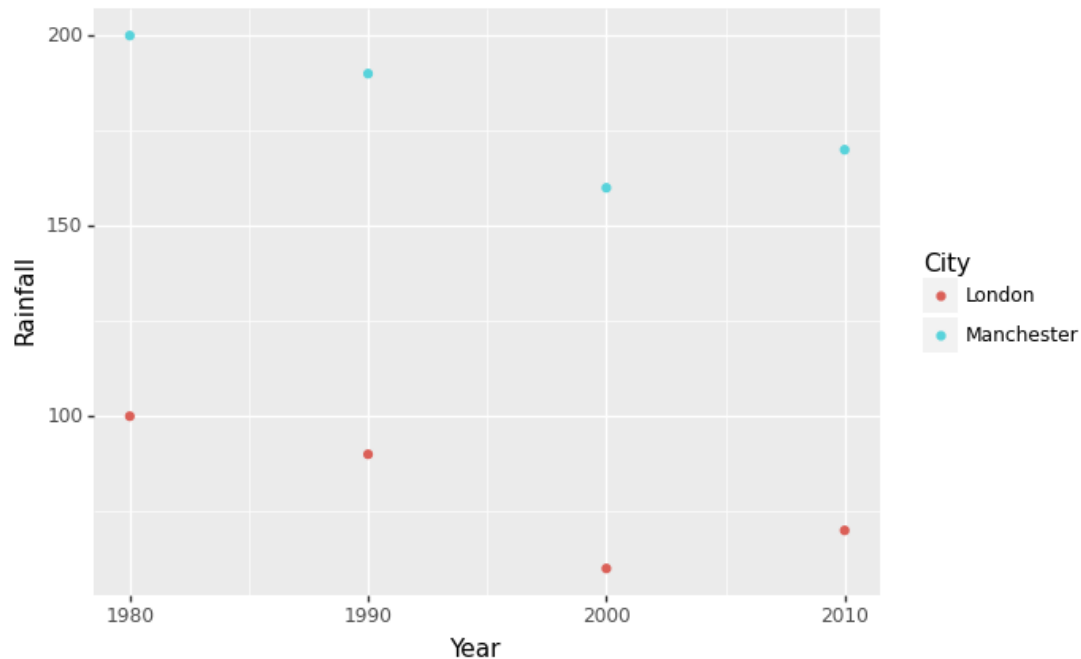
The first plot we will create is a scatter plot, which uses the `geom_point()` geom. The important points to note in the code below are:

- `ggplot()`, `aes()` and `geom_point()` all come from the `plotnine` module, so they need to have the `p9.` prefix
- the first argument to `ggplot()` is always the dataframe containing the data to be plotted
- the second argument to `ggplot()` is always the `aes()` object, which contains the variables to be used for the x axis, y axis, plot fill and plot colour. These should always be specified in that order.
- the `geom_point()` argument is preceded by a `+` because it is being added to the plot object created by the `ggplot()` function
- I have separated the commands across multiple lines to make it easier to read. When doing this, the whole chunk of code needs to be encapsulated in brackets

```

[4]: (
  # create the plot object, and specify which variables should be placed on the x_
  ↳ y axes, and which should be used to fill and colour the plot
  p9.ggplot(example, p9.aes("Year", "Rainfall", fill = "City", colour="City"))
    # specify a scatter plot
    + p9.geom_point()
)

```

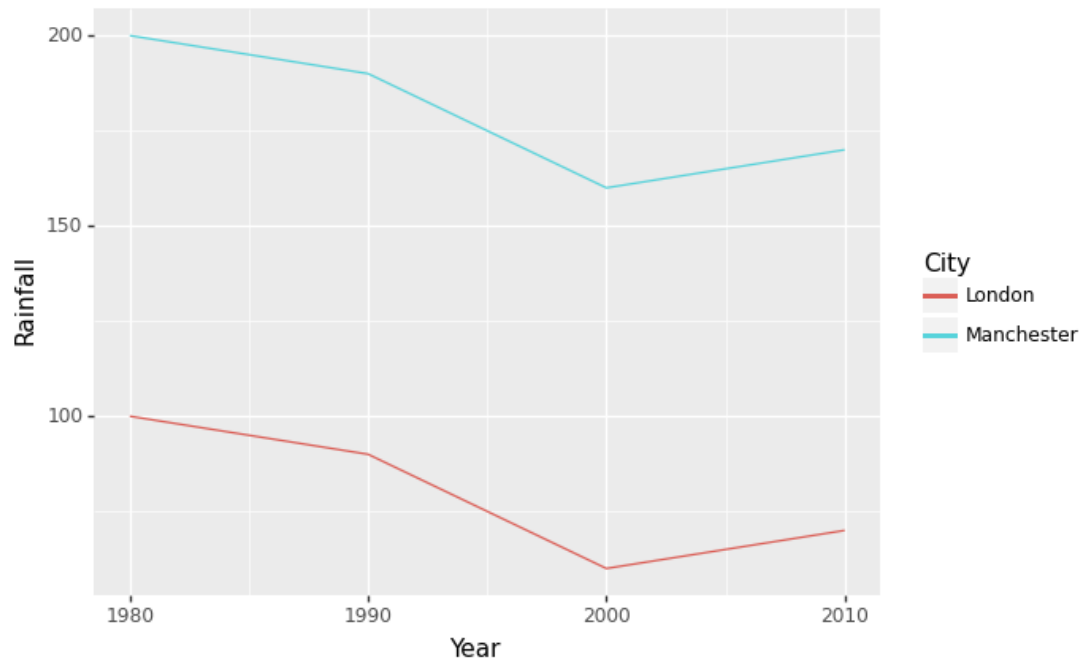


```
[4]: <ggplot: (-9223371888451666568)>
```

## 2.3 Line plot

To create a line plot, we can use the same `ggplot()` call as we used for the scatter plot above, and this will keep the same variables for the *x*, *y*, *fill* and *colour*. The only thing that needs to change about the code is the **geom**. In this case, we will use `geom_line()`:

```
[5]: (  
  # the ggplot call is exactly the same as the scatter plot  
  p9.ggplot(example, p9.aes("Year", "Rainfall", fill="City", colour="City"))  
    # specify a line plot  
    + p9.geom_line()  
)
```

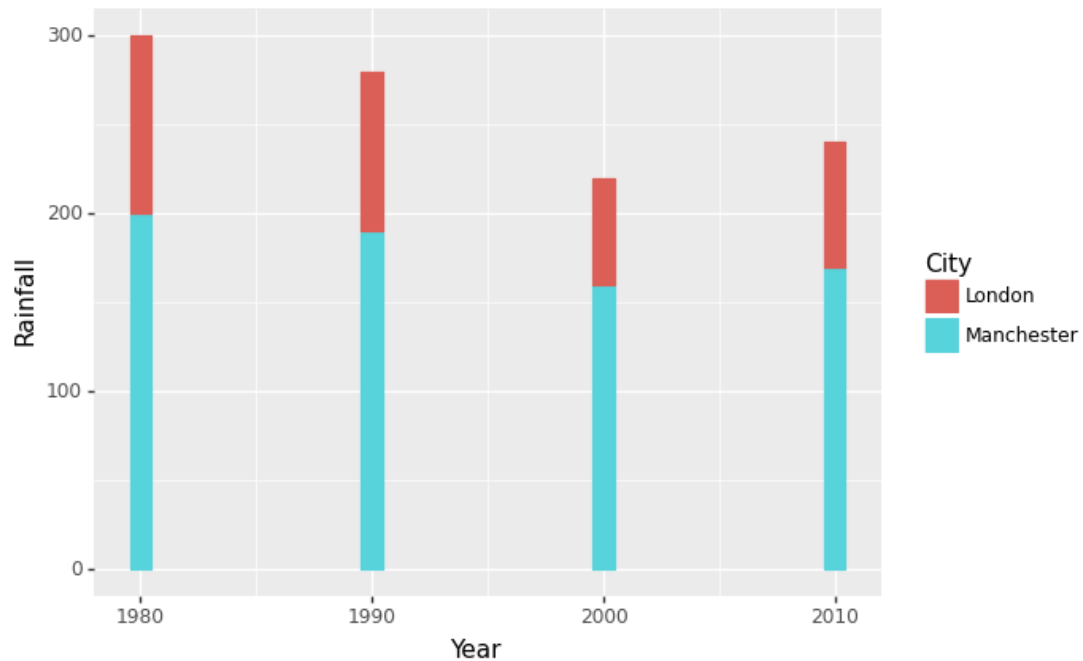


```
[5]: <ggplot: (-9223371888451934280)>
```

## 2.4 Bar chart

To create a bar chart, we again only need to change the **geom**. In this case, we will use `geom_col()`:

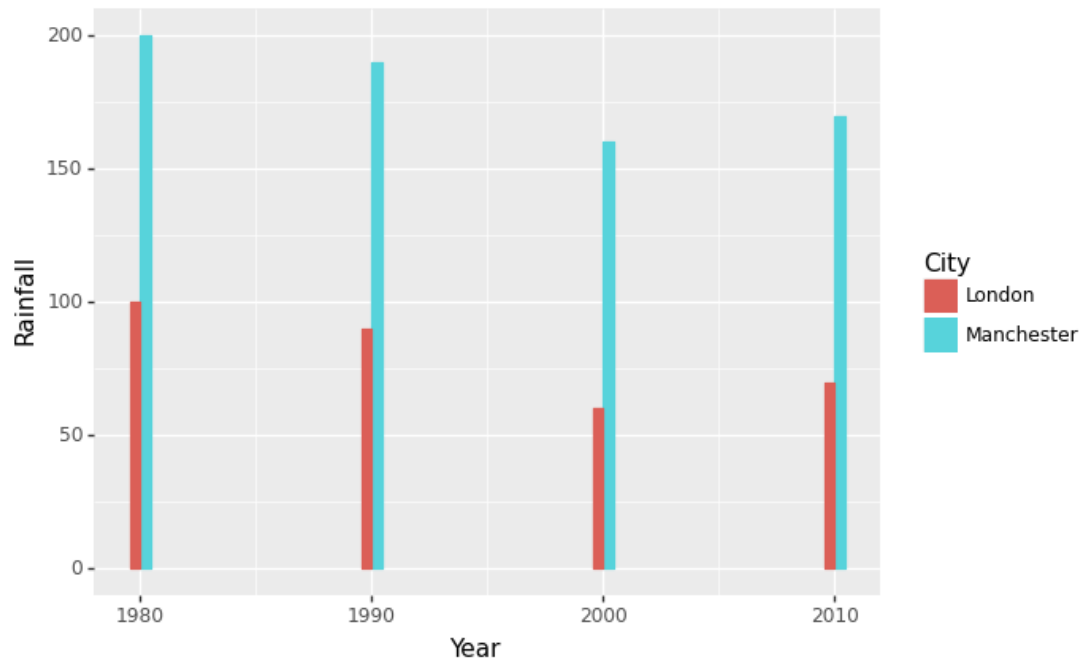
```
[6]: (  
  # the ggplot call is exactly the same as the scatter plot  
  p9.ggplot(example, p9.aes("Year", "Rainfall", fill="City", colour="City"))  
  # specify a bar chart  
  + p9.geom_col()  
)
```



```
[6]: <ggplot: (-9223371888451618188)>
```

By default `geom_col()` stacks the bars for the same x axis variable on top of each other, but we can place them next to each other by using the `position` argument:

```
[7]: (  
  # the ggplot call is exactly the same as the scatter plot  
  p9.ggplot(example, p9.aes("Year", "Rainfall", fill="City", colour="City"))  
  # specify a bar chart  
  + p9.geom_col(position="dodge")  
)
```

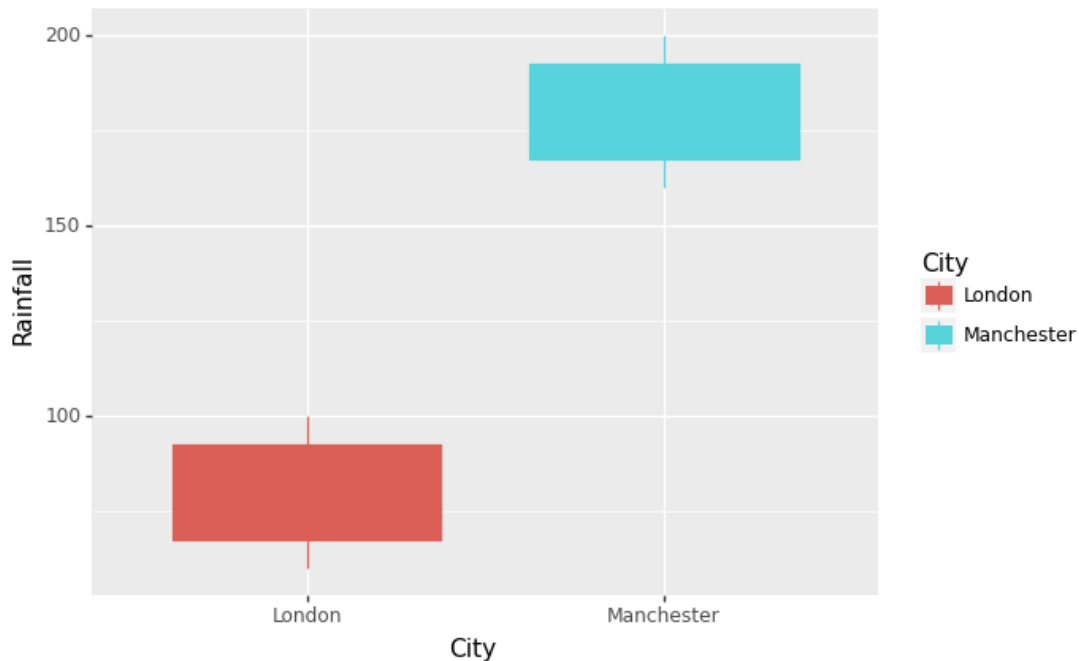


```
[7]: <ggplot: (-9223371888451494940)>
```

## 2.5 Box plot

To create a box plot that visualizes the rainfall for the two cities, we need to use a different **geom** and change the x axis argument. In this case, we will use `geom_box()` and replace `Year` with `City` in the `aes()` call:

```
[8]: (  
  # the ggplot call is exactly the same as the scatter plot  
  p9.ggplot(example, p9.aes("City", "Rainfall", fill="City", colour="City"))  
    # specify a box plot  
    + p9.geom_boxplot()  
)
```



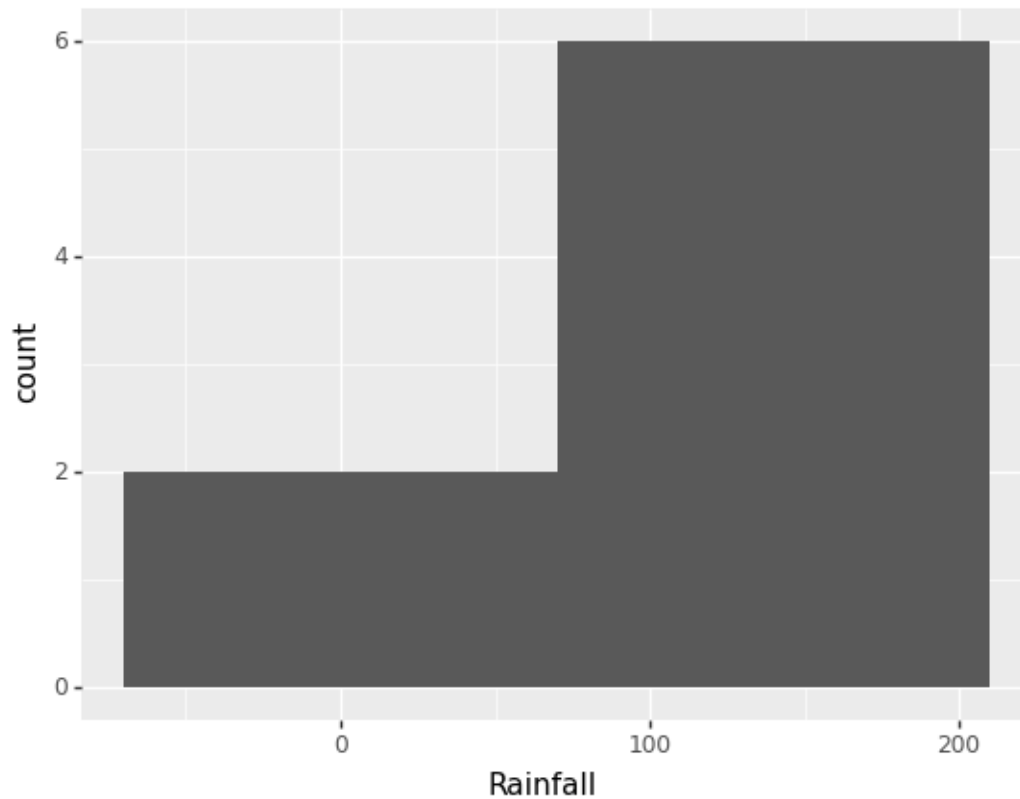
```
[8]: <ggplot: (-9223371888451462112)>
```

## 2.6 Histograms

Histograms can be very useful to visualize the distribution of a variable. To generate a histogram, we use `geom_histogram`. An important difference between this plot and the previous plots is that for histograms, we only specify the x axis variable within the `aes()` object, and we leave out the y axis variable:

```
[9]: (
  # the aes object contains only the variable that will be plotted on the x axis
  # i.e. the one for which we want to generate a distribution
  p9.ggplot(example, p9.aes("Rainfall"))
  # specify a histogram
  + p9.geom_histogram()
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\stats\stat_bin.py:93:
PlotnineWarning: 'stat_bin()' using 'bins = 2'. Pick better value with
'binwidth'.
  warn(msg.format(params['bins']), PlotnineWarning)
```



[9]: <ggplot: (-9223371888451329440)>

## 3 Advanced visualization

### 3.1 Input data

To create some more advanced plots, we will use a more complex dataset: the clinical and expression data from the metabric project, as used in the previous sessions. First we will read the data in as a pandas dataframe, using `dropna()` to remove any rows with missing data, and then print out the column names and the data types that they hold by calling the `dtypes` attribute.

```
[10]: metabric = pd.read_csv("../data/metabric_clinical_and_expression_data.csv").
      ↪ dropna()
      metabric.dtypes
```

```
[10]: Patient_ID          object
      Cohort              int64
      Age_at_diagnosis    float64
      Survival_time       float64
      Survival_status     object
```



Vital_status	object
Chemotherapy	object
Radiotherapy	object
Tumour_size	float64
Tumour_stage	float64
Neoplasm_histologic_grade	float64
Lymph_nodes_examined_positive	int64
Lymph_node_status	int64
Cancer_type	object
ER_status	object
PR_status	object
HER2_status	object
HER2_status_measured_by_SNP6	object
PAM50	object
3-gene_classifier	object
Nottingham_prognostic_index	float64
Cellularity	object
Integrative_cluster	object
Mutation_count	float64
ESR1	float64
ERBB2	float64
PGR	float64
TP53	float64
PIK3CA	float64
GATA3	float64
FOXA1	float64
MLPH	float64
dtype:	object

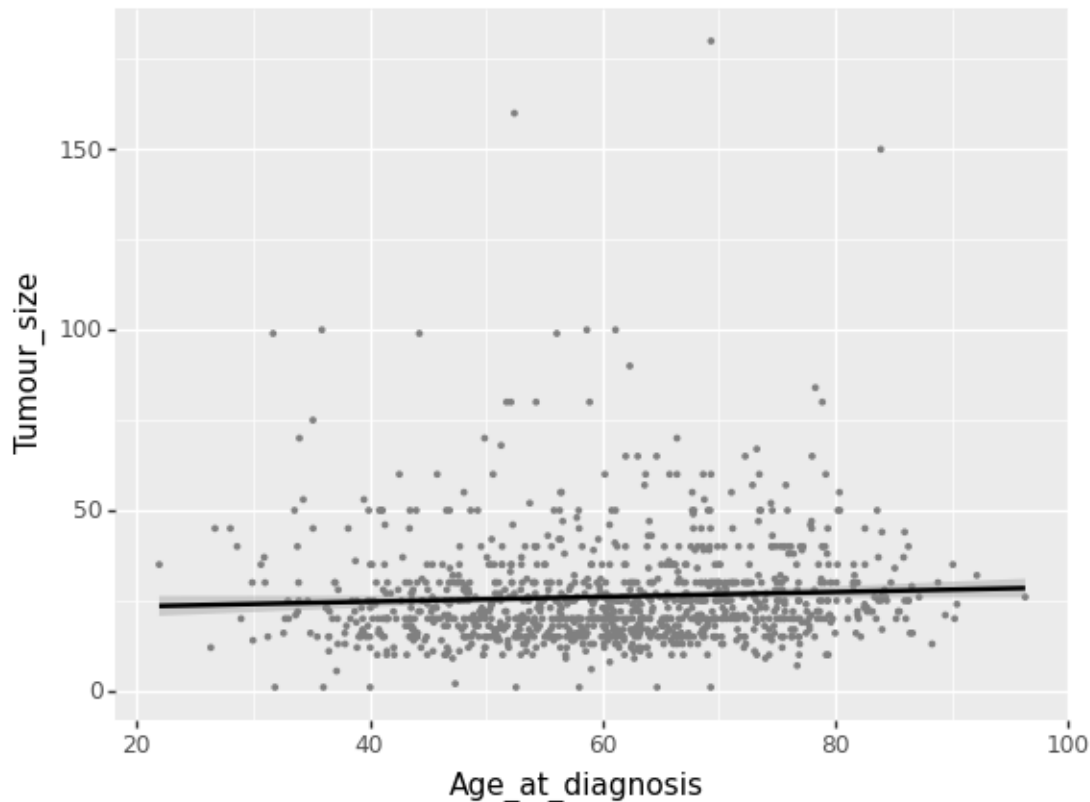
### 3.2 Layered plots

As shown in the last section, a variety of simple plots can be generated by changing just the **geom**. However, we can construct more complex plots by layering different geoms on top of each other. This is particularly useful when we want to highlight a broad pattern in a noisy dataset: we can use one layer to display a line to visualize a model fit to the data, and another layer to show the raw data in the background. There are two things worth noting here: 1. By passing the **colour** and **size** arguments to `geom_point()` we can specify the colour and size of the points 2. We are using the `stat_smooth()` layer to plot a line of model fit on top of the raw data

```
[11]: (
  p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
  + p9.geom_point(colour="grey", size=0.5)
  + p9.stat_smooth()
)
```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
FutureWarning: Method .ptp is deprecated and will be removed in a future

```
version. Use numpy.ptp instead.
return ptp(axis=axis, out=out, **kwargs)
```



```
[11]: <ggplot: (-9223371888450935172)>
```

### 3.3 Faceted plots

When a large dataset contains a discrete variable, it can be useful to generate separate plots for each value of the discrete variable, so that we can tease apart patterns that might only be present in a subset of the data. For example, patients in the metabric dataset are each assigned a vital status, but each vital status might have a different tumour size distribution. In plotnine, separate plots can be generated by creating a **faceted** plot, where each facet represents a different subset of the data. To do this, we add the `facet_wrap()` parameter to the plot. The arguments that we pass to `facet_wrap()` are very important: 1. The first argument must always be the variable that we want to generate separate plots for, and we must always add a `~` before it 2. We can use the `scales` argument to make the range of the x axis, y axis, or both axes independent of each other. The default is to use the same axis ranges for all facets, but when there is wide variation in a variable it can be useful to make them independent. In this case there are different numbers of patients in the different vital status classes, so specifying independent y axis ranges allows us to see their distributions more clearly 3. We can use the `ncol` argument to specify the number of columns

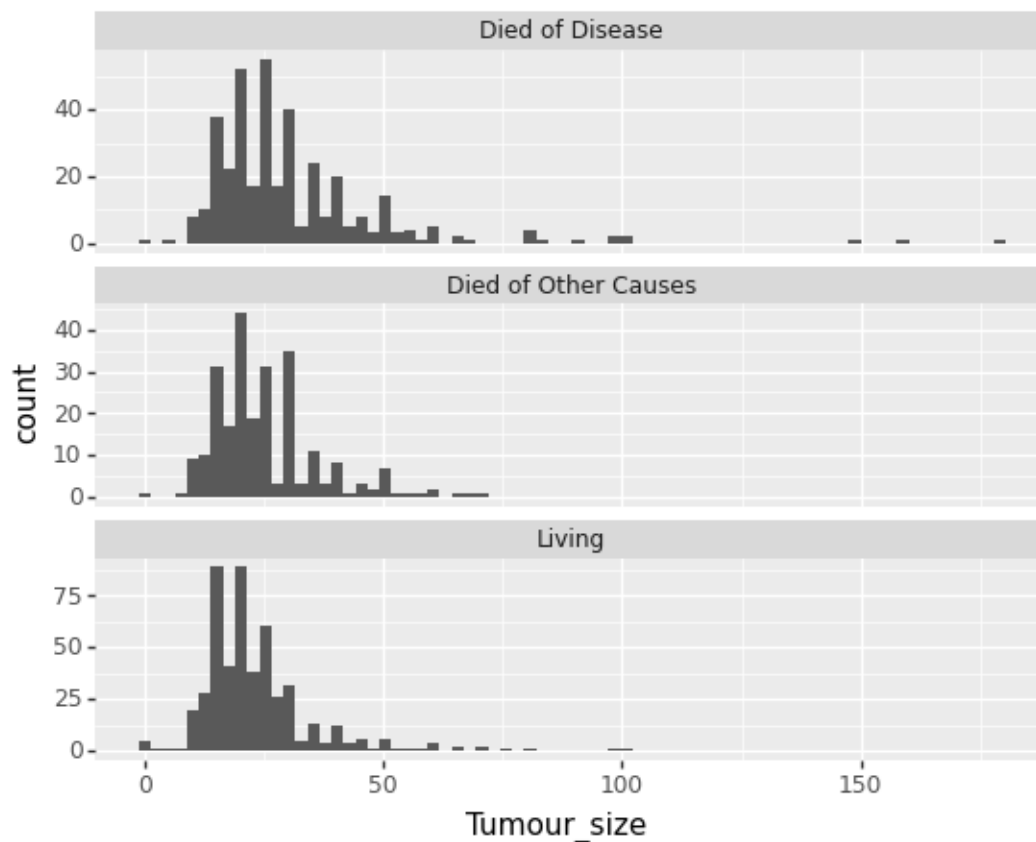
for the plot. By default the plots will be placed side-by-side until there is no more horizontal space, but we can place them on top of each other by specifying one column

```
[12]: (
  p9.ggplot(metabric, p9.aes("Tumour_size"))
  + p9.geom_histogram()
  + p9.facet_wrap("~Vital_status", scales="free_y", ncol=1)
)
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\stats\stat\_bin.py:93:

PlotnineWarning: 'stat\_bin()' using 'bins = 72'. Pick better value with 'binwidth'.

warn(msg.format(params['bins']), PlotnineWarning)



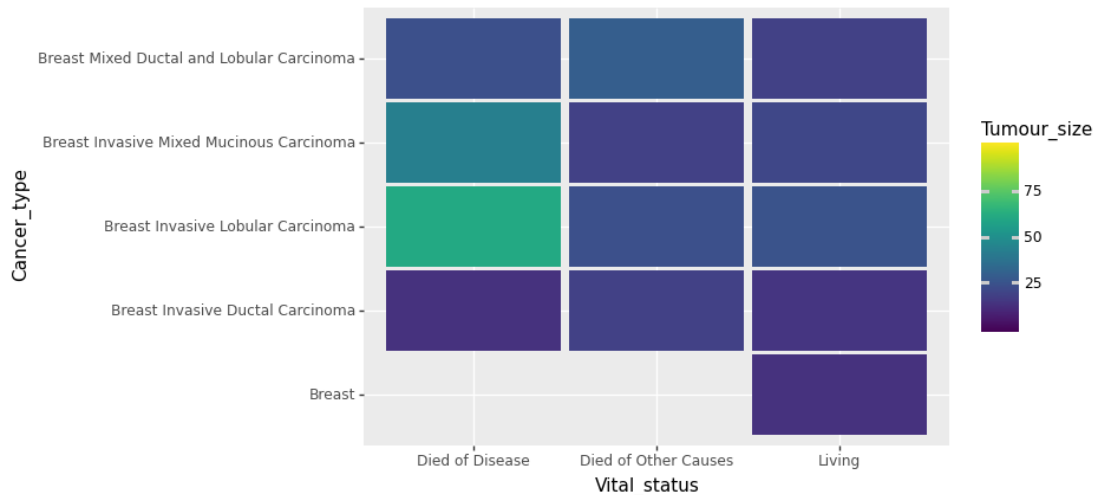
```
[12]: <ggplot: (-9223371888450936588)>
```

### 3.4 Heat maps

When we want to explore the relationship between 2 discrete variables and a continuous variable, a heatmap can be very useful. We use `geom_tile()` to generate a different block for each combination

of the discrete variables, and use the `fill` argument in the `aes()` object to specify which continuous variable to use for colouring the heatmap. It is worth noting in the example below that we can filter the dataset within the `ggplot()` call:

```
[13]: # we can see from the histograms above that there are a few very large tumours,
      ↪so we will filter these out as we pass the data into the plot
      (
p9.ggplot(metabric[metabric["Tumour_size"]<100], p9.aes("Vital_status",
      ↪"Cancer_type", fill="Tumour_size"))
      + p9.geom_tile(p9.aes(width=0.95, height=0.95))
      )
```

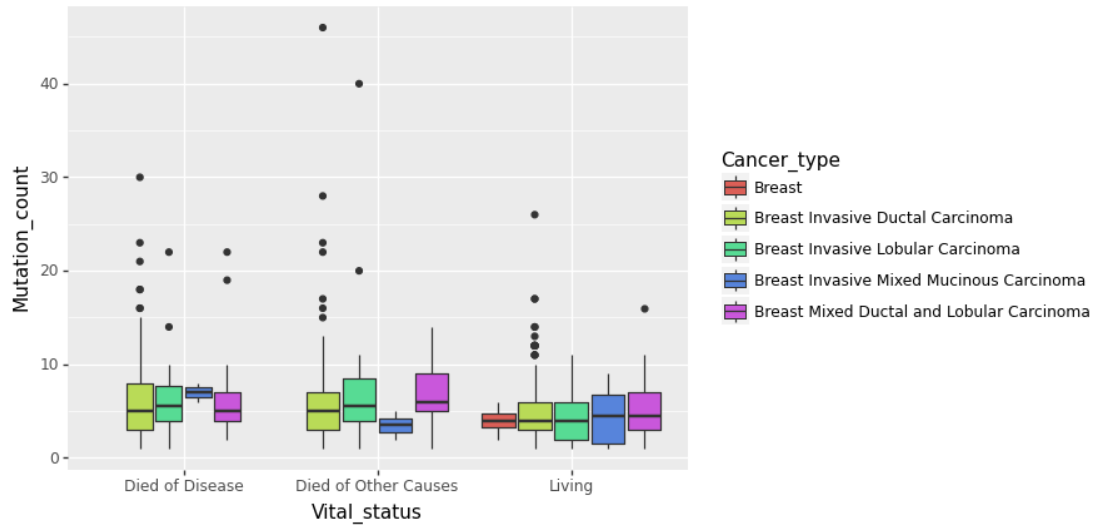


```
[13]: <ggplot: (-9223371888451293168)>
```

### 3.5 Nested plots

In complex datasets there may be a relationship between two variables that differs according to the value of a third variable. To visualize this, we can plot one variable on the x axis, another variable on the y axis, and use the `fill` parameter in the `aes()` object to split the x axis by a third variable. For example, we can compare the mutation count between different vital statuses, each subdivided by cancer type:

```
[14]: (
p9.ggplot(metabric, p9.aes("Vital_status", "Mutation_count",
      ↪fill="Cancer_type"))
      + p9.geom_boxplot()
      )
```

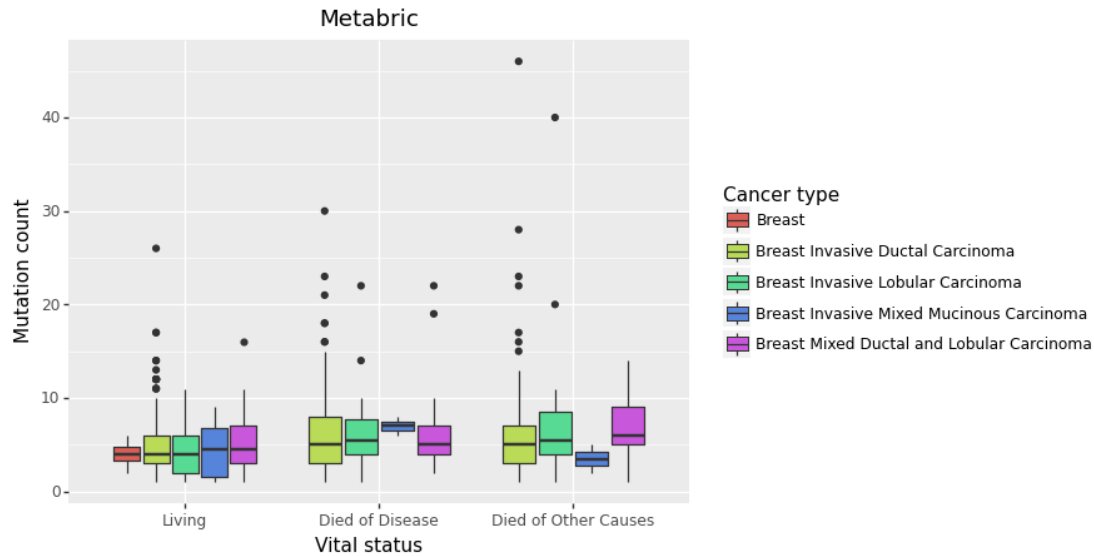


```
[14]: <ggplot: (-9223371888451337296)>
```

### 3.6 Plot formatting

We often want to make specific changes to the format of our plots, and `plotnine` offers enormous flexibility for this. In the nested plot above, we may want to add a title, remove the underscores in the axes and legend titles, and change the order of the vital statuses. We can make these changes by specifying the following objects: - `ggtitle`: add a title to the plot - `xlab`: specify the x axis label - `ylab`: specify the y axis label - `scale_fill_discrete`: control the name, order or colours of the variable used to fill the plot - `scale_x_discrete`: control the name, order or colours of the variable on the x axis. By setting the `limits` parameter to a list, we can specify the order of the vital statuses

```
[15]: (
  p9.ggplot(metabric, p9.aes("Vital_status", "Mutation_count",
    ↪fill="Cancer_type"))
    + p9.geom_boxplot()
    + p9.ggtitle("Metabric")
    + p9.xlab("Vital status")
    + p9.ylab("Mutation count")
    + p9.scale_fill_discrete(name = "Cancer type")
    + p9.scale_x_discrete(limits=["Living", "Died of Disease", "Died of Other_
    ↪Causes"])
)
```



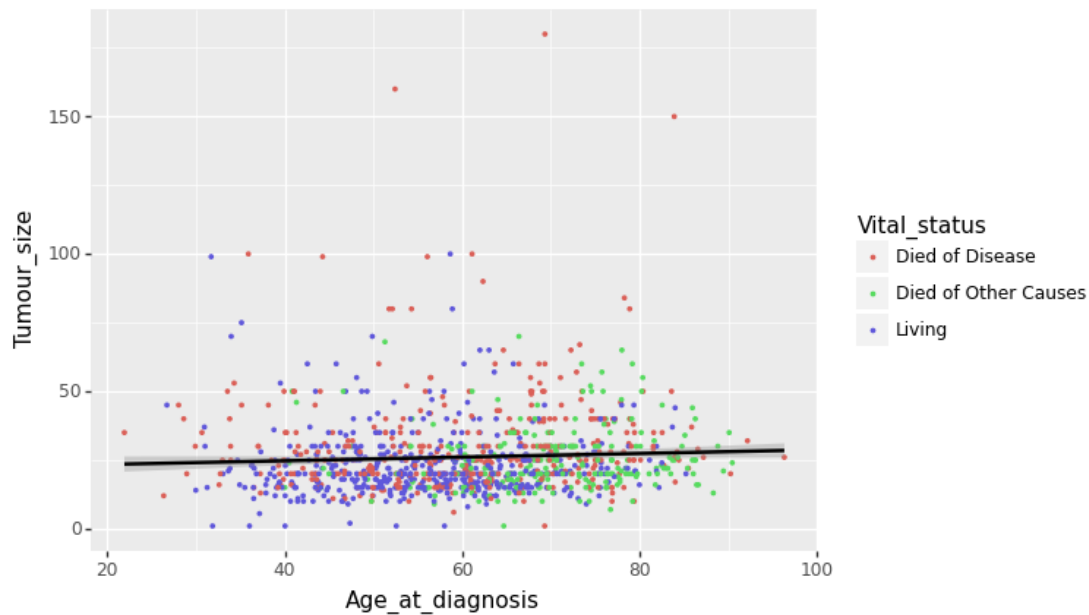
```
[15]: <ggplot: (-9223371888451894340)>
```

### 3.7 Themes

In any plot generated by plotnine, many cosmetic details are controlled by the **theme**, including the font, background colour, line thicknesses etc. plotnine comes with a number of pre-defined themes, which allow us to make dramatic changes to how a plot looks by adding just one more line to our existing plotting code. Let's go back to the plot of age at diagnosis versus tumour size, but this time colour the points by vital status:

```
[16]: (
  p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
  + p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
  + p9.stat_smooth()
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```



[16]: <ggplot: (-9223371888451268772)>

By using `theme_minimal()` we can remove the grey background and axis tickmarks, and make the minor gridlines much fainter:

```
[17]: (
  p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
  + p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
  + p9.stat_smooth()
  + p9.theme_minimal()
)
```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
 FutureWarning: Method .ptp is deprecated and will be removed in a future  
 version. Use numpy.ptp instead.  
 return ptp(axis=axis, out=out, \*\*kwargs)



```
[17]: <ggplot: (-9223371888451534544)>
```

`theme_matplotlib()` removes the gridlines and grey background, and add minor tickmarks and a border to the plot:

```
[18]: (
  p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
  + p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
  + p9.stat_smooth()
  + p9.theme_matplotlib()
)
```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
FutureWarning: Method .ptp is deprecated and will be removed in a future  
version. Use numpy.ptp instead.

return ptp(axis=axis, out=out, \*\*kwargs)

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:

The datapath rcparam was deprecated in Matplotlib 3.2.1 and will be removed two  
minor releases later.

rcParams[key] = val

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:

The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed  
in 3.3.

rcParams[key] = val

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:



MatplotlibDeprecationWarning:

The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and will be removed in 3.2.

```
rcParams[key] = val
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:

MatplotlibDeprecationWarning:

The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.

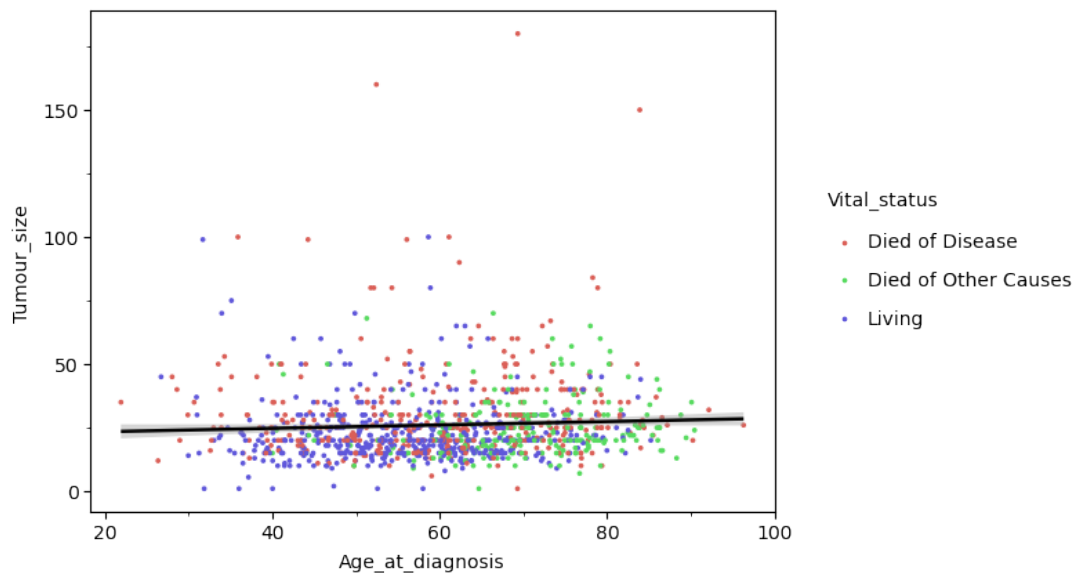
```
rcParams[key] = val
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:

MatplotlibDeprecationWarning:

The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.

```
rcParams[key] = val
```



```
[18]: <ggplot: (-9223371888451394240)>
```

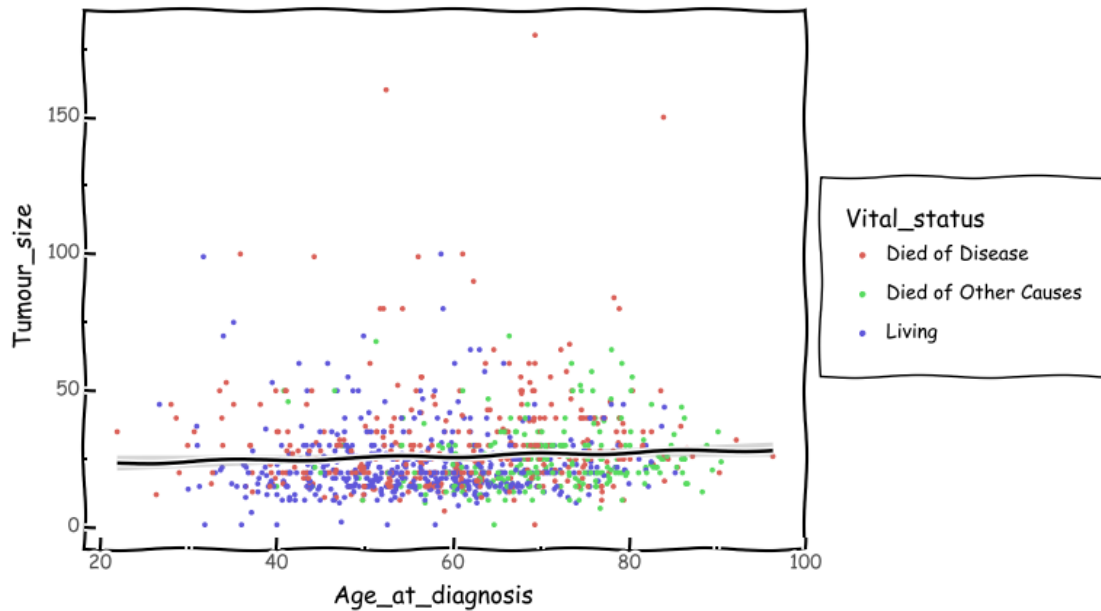
`theme_xkcd()` gives the plot a hand-drawn feel, following the style of the [xkcd webcomic](#):

```
[19]: (
  p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
  + p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
  + p9.stat_smooth()
  + p9.theme_xkcd()
)
```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:

FutureWarning: Method .ptp is deprecated and will be removed in a future version. Use numpy.ptp instead.

```
return ptp(axis=axis, out=out, **kwargs)
```

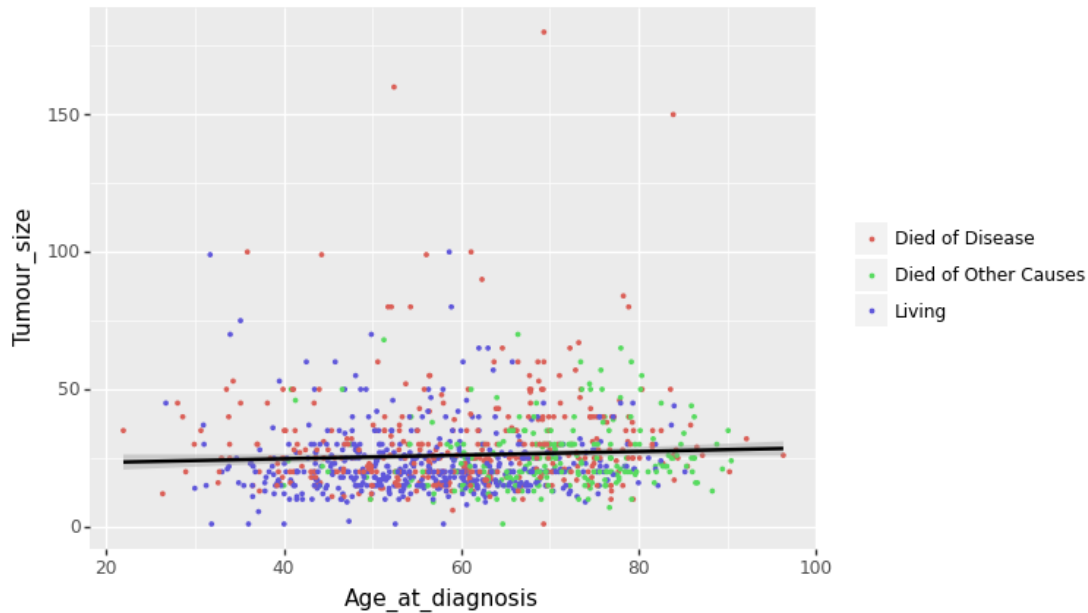


```
[19]: <ggplot: (-9223371888451119836)>
```

As well as using the settings from pre-defined themes, we can specify changes to the theme directly. To do this, we add the `theme()` object to the plot, and make changes to specific aspects of the plot within the call to the object. For example, we can create a version of the scatterplot above that doesn't have a legend title. To do this, we set the value of the `legend_title` parameter as `element_blank()`. `element_blank()` is a special object from the `plotnine` module that serves as an empty placeholder, and it can be used to remove the values for many different plot elements such as axis labels, axis values etc. In this case, it removes the legend title:

```
[20]: (
p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
+ p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
+ p9.stat_smooth()
+ p9.theme(legend_title = p9.element_blank())
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
return ptp(axis=axis, out=out, **kwargs)
```



[20]: <ggplot: (-9223371888451377732)>

We can combine specific changes and pre-existing themes by first specifying the theme, and then using the `theme()` object to specify changes to that theme. When doing this, we must always apply the pre-defined theme first, otherwise the specific changes we have made using the `theme()` object will be overwritten. For example, we can apply the `matplotlib` theme, and then remove the legend title by using the `theme()` object:

```
[21]: (
p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
+ p9.geom_point(p9.aes(colour="Vital_status"), size=0.5)
+ p9.stat_smooth()
+ p9.theme_matplotlib()
+ p9.theme(legend_title = p9.element_blank())
)
```

C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
FutureWarning: Method .ptp is deprecated and will be removed in a future  
version. Use numpy.ptp instead.

return ptp(axis=axis, out=out, \*\*kwargs)

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:

The datapath rcparam was deprecated in Matplotlib 3.2.1 and will be removed two  
minor releases later.

rcParams[key] = val

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:

The `savefig.frameon` rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.

```
rcParams[key] = val
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:

MatplotlibDeprecationWarning:

The `text.latex.unicode` rcparam was deprecated in Matplotlib 3.0 and will be removed in 3.2.

```
rcParams[key] = val
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:

MatplotlibDeprecationWarning:

The `verbose.fileo` rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.

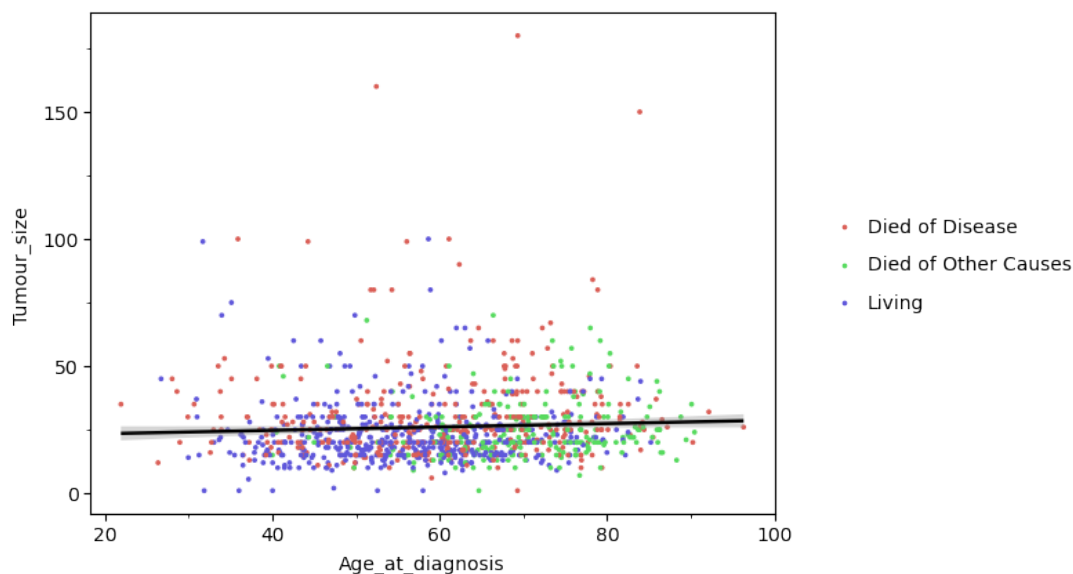
```
rcParams[key] = val
```

C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:

MatplotlibDeprecationWarning:

The `verbose.level` rcparam was deprecated in Matplotlib 3.1 and will be removed in 3.3.

```
rcParams[key] = val
```



```
[21]: <ggplot: (-9223371888451118212)>
```

### 3.8 Saving plots

Once we have produced a plot, we usually want to save it as a separate file. In Python, we can do this by assigning the `ggplot()` call to a variable, and then calling the `.save()` method on this variable. The first argument to `.save()` is always the file name (or full file path), with the file type determined by the file name ending. We can also specify the height and width of the plot, and the

units of measurement for these dimensions.

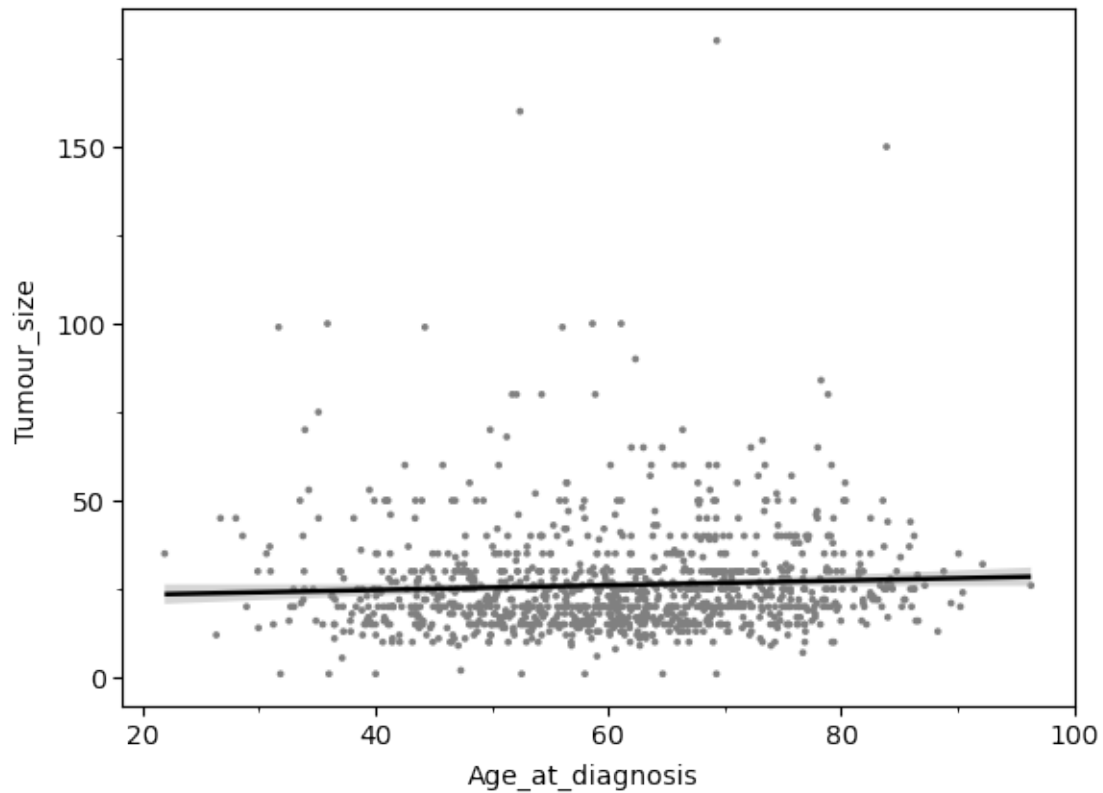
```
[22]: age_vs_size = (  
    p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))  
    + p9.geom_point(colour="grey", size=0.5)  
    + p9.stat_smooth()  
    + p9.theme_matplotlib()  
    )  
age_vs_size.save("metabric_age_size.svg", height=100, width=100, units="mm")  
age_vs_size
```

```
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\ggplot.py:729:  
PlotnineWarning: Saving 100.0 x 100.0 mm image.  
    from_inches(height, units), units), PlotnineWarning)  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\ggplot.py:730:  
PlotnineWarning: Filename: metabric_age_size.svg  
    warn('Filename: {}'.format(filename), PlotnineWarning)  
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
FutureWarning: Method .ptp is deprecated and will be removed in a future  
version. Use numpy.ptp instead.  
    return ptp(axis=axis, out=out, **kwargs)  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:  
The datapath rcparam was deprecated in Matplotlib 3.2.1 and will be removed two  
minor releases later.  
    rcParams[key] = val  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:  
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed  
in 3.3.  
    rcParams[key] = val  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:  
The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and will be  
removed in 3.2.  
    rcParams[key] = val  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:  
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed  
in 3.3.  
    rcParams[key] = val  
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:  
MatplotlibDeprecationWarning:  
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed  
in 3.3.  
    rcParams[key] = val  
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:  
FutureWarning: Method .ptp is deprecated and will be removed in a future
```

```

version. Use numpy.ptp instead.
    return ptp(axis=axis, out=out, **kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:
MatplotlibDeprecationWarning:
The datapath rcparam was deprecated in Matplotlib 3.2.1 and will be removed two
minor releases later.
    rcParams[key] = val
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:
MatplotlibDeprecationWarning:
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
    rcParams[key] = val
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:
MatplotlibDeprecationWarning:
The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and will be
removed in 3.2.
    rcParams[key] = val
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:
MatplotlibDeprecationWarning:
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
    rcParams[key] = val
C:\ProgramData\Anaconda3\lib\site-packages\plotnine\themes\theme.py:250:
MatplotlibDeprecationWarning:
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
    rcParams[key] = val

```



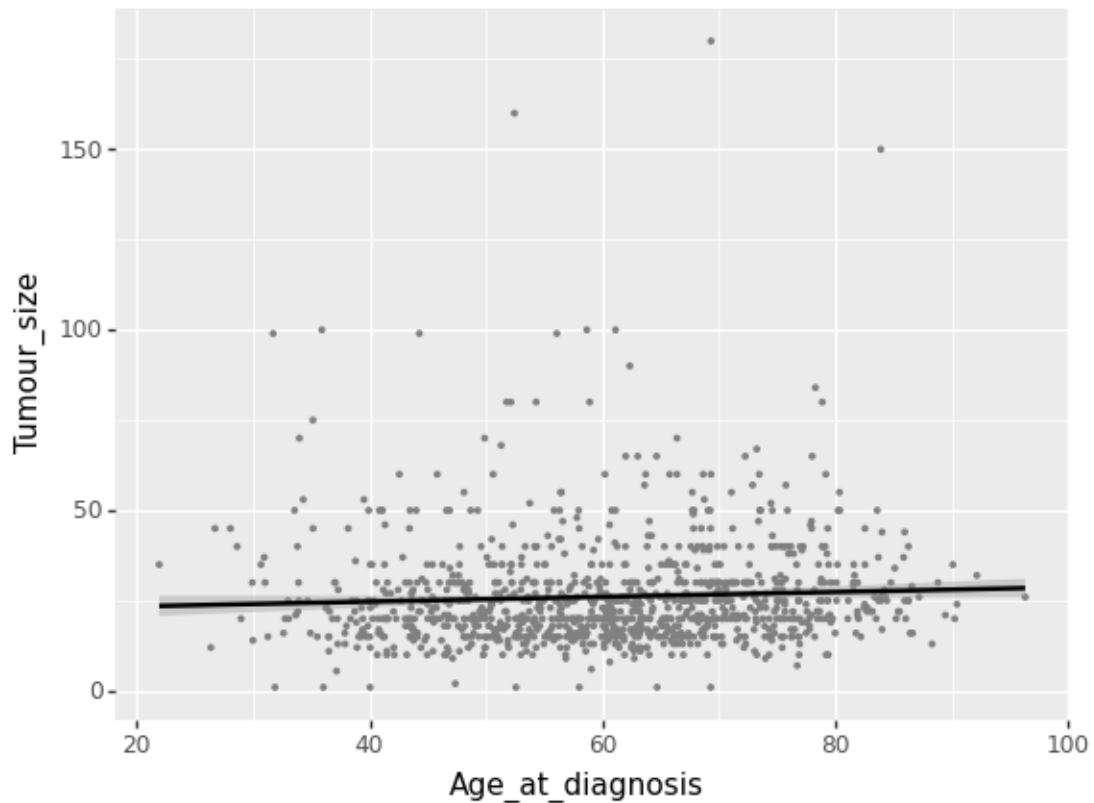
[22]: <ggplot: (-9223371888451400752)>

### 3.9 Suppressing warning messages

When using plotnine for data visualization, a lot of warning messages can be generated. For example,

```
[23]: (
p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
+ p9.geom_point(colour="grey", size=0.5)
+ p9.stat_smooth()
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
    return ptp(axis=axis, out=out, **kwargs)
```

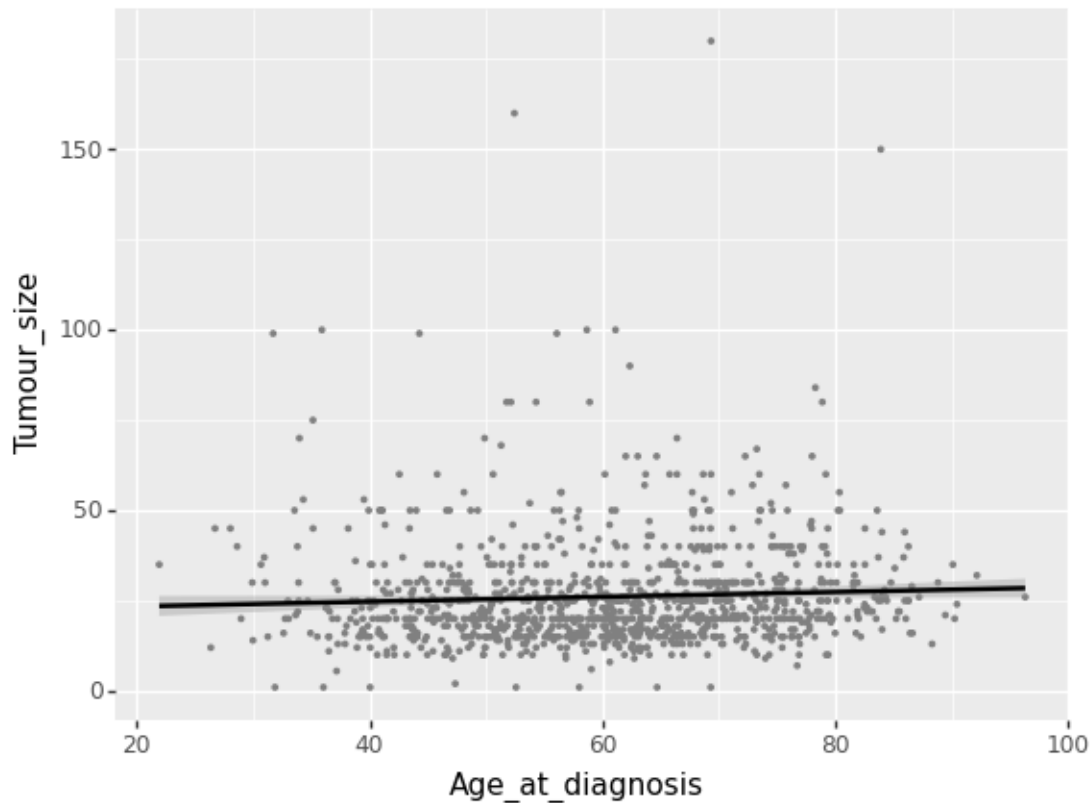


[23]: <ggplot: (-9223371888451580996)>

These messages are often informative, but they can be undesirable if we want to use the notebook to communicate our findings. To suppress these messages, we can use the `filterwarnings()` function from the `warnings` module. By setting this to "ignore", we will suppress warning messages from plotnine:

```
[24]: import warnings
warnings.filterwarnings('ignore')
(
p9.ggplot(metabric, p9.aes("Age_at_diagnosis", "Tumour_size"))
+ p9.geom_point(colour="grey", size=0.5)
+ p9.stat_smooth()
)
```





```
[24]: <ggplot: (-9223371888451005536)>
```

**Use this with caution:** all warnings from all Python modules will be suppressed, so it is usually sensible to run your code with warnings printed first, check the messages are safe to ignore, and then rerun your code with the messages suppressed.

### 3.10 Further reading

In this session we have seen that plotnine can be used to generate many different plots with very little code. However, we have just scratched the surface of what is possible with this package, and with data visualization in Python more generally. There are many resources that you can use to extend your Python data vizualization skills: - to see the full range of plot types that can be generated with plotnine, see the [plotnine gallery](#) - for full details of how each plotnine geom and object works, see the [plotnine API documentation](#) - if you want to generate bespoke plots with full control over each plotting element, take a look at the [matplotlib](#) library

## 4 Exercises

### 4.1 Exercise 1

Plot the distribution of survival time for all patients, and use the *bins* parameter to change the granularity of the distribution.

[ ]:

### 4.2 Exercise 2

Amend your code from exercise 1 to create interleaved distributions of survival time for ER positive and ER negative tumours.

[ ]:

### 4.3 Exercise 3

Generate a boxplot to compare survival time between different cancer types, incorporating the following features: 1. Add points behind the boxes to show the raw data points 2. Colour the boxes by Cancer type 3. Separate the plot into individual facets for ER+ and ER- 4. Rename the x axis to "Type of cancer" 5. Rename the y axis to "Survival time (months)" 6. Use a theme that removes the background colour and gridlines 7. Remove the legend title 8. Remove the X axis labels

[ ]: