# An R Markdown document converted from “traffic.ipynb”

# Assignment - British road network use

KATE expects your code to define variables with specific names that correspond to certain things we are interested in.

KATE will run your notebook from top to bottom and check the latest value of those variables, so make sure you don’t overwrite them.

- Remember to uncomment the line assigning the variable to your answer and don’t change the variable or function names.
- Use copies of the original or previous DataFrames to make sure you do not overwrite them by mistake.

You will find instructions below about how to define each variable.

Once you’re happy with your code, upload your notebook to KATE to check your feedback.

First of all, we will import `pandas` and `pandas_bokeh` and set them up:

```
library(tidyverse) #mainly for dplyr
library(lubridate) # date manipulation
library("glue") # literal string manipulation
library("collapse") # fast data transformation and aggregateion
library("fs") # file system operations
library("data.table") # fast data transformation and aggregation
library("ragg") # resizing graphics output
library("plotly") # making plots interactive
# library(reticulate) # python in Rstudio
reticulate::use_condaenv("py38", required = TRUE) # need python 3.8 for bokeh
```

```
import pandas as pd
```

```
import pandas_bokeh
```

```
from bokeh.plotting import show

from bokeh.plotting import output_notebook
output_notebook()
from bokeh.plotting import figure, output_file, save # this is needed when running
in
# an IDE not a notebook


import warnings
warnings.filterwarnings('ignore')
```

Use `.read_csv()` to get our dataset `data/region_traffic.csv` and assign to `df` :

```
import pandas as pd
df = pd.read_csv('data/region_traffic.csv')
df
```

```
##       year  region_id  ...      all_hgvs all_motor_vehicles
## 0     1993          1  ...  4.289609e+08       3.465840e+09
## 1     1993          1  ...  2.771219e+08       3.484710e+09
## 2     1993          1  ...  3.733318e+08       7.794004e+09
## 3     1993          1  ...  7.177956e+07       2.363717e+09
## 4     1993          1  ...  1.443973e+08       6.748291e+09
## ...    ...        ...  ...           ...                ...
## 1574  2018         11  ...  7.728592e+05       4.195688e+07
## 1575  2018         11  ...  1.954551e+08       2.763069e+09
## 1576  2018         11  ...  1.350186e+08       4.178892e+09
## 1577  2018         11  ...  8.763506e+06       7.132722e+08
## 1578  2018         11  ...  3.369251e+07       3.857152e+09
##
## [1579 rows x 14 columns]
```

```
df <- read_csv("data/region_traffic.csv")
```

```
## Rows: 1579 Columns: 14
## ── Column specification ────────────────────────────────────────────────────────
─
## Delimiter: ","
## chr  (2): name, ons_code
## dbl (12): year, region_id, road_category_id, total_link_length_km, total_lin...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this messag
e.
```

**Q1.** Use `.groupby()` to create a DataFrame called `year` which groups `df` by `year` and contains the columns `['pedal_cycles', 'cars_and_taxis', 'all_hgvs']`, with the `.sum()` of each of these for each year:

```
year = df.groupby('year')['pedal_cycles', 'cars_and_taxis', 'all_hgvs'].sum()
year.head()
```

```
##       pedal_cycles  cars_and_taxis      all_hgvs
## year
## 1993  2.489981e+09    2.100849e+11  1.507144e+10
## 1994  2.495693e+09    2.143886e+11  1.539442e+10
## 1995  2.573601e+09    2.181758e+11  1.581009e+10
## 1996  2.531690e+09    2.236457e+11  1.630137e+10
## 1997  2.536137e+09    2.272964e+11  1.668684e+10
```

```
# dplyr
year <- df %>%
    group_by(year) %>%
    summarise(across(.cols = c(pedal_cycles,
                               cars_and_taxis,
                               all_hgvs),
                     sum, na.rm = TRUE))
```

We want to look at the change over time of each of these forms of transport relative to the earliest values.

To do so, we will create an *index*. An index allows us to inspect the growth over time of a variable relative to some starting value (known as the *base*). By convention, this starting value is `100.0`. If the value of our variable doubles in some future time period, then the value of our index in that future time period would be `200.0`.

**Q2.** Create a new DataFrame called `year_index` as a `.copy()` of `year`. For our index, we will select 1993 as the base year. This means that all values for 1993 should be equal to `100.0`. All subsequent years should be relative to that.

Note that you do not need to apply any rounding to the index.

```
yr = year.copy()

base = yr.loc[1993]
year_index = (yr * 100)/ base
year_index.head()
```

```
##       pedal_cycles  cars_and_taxis    all_hgvs
## year
## 1993    100.000000      100.000000  100.000000
## 1994    100.229413      102.048581  102.143030
## 1995    103.358260      103.851256  104.900983
## 1996    101.675079      106.454909  108.160667
## 1997    101.853694      108.192646  110.718300
```

```
pd.set_option('plotting.backend', 'pandas_bokeh')
```

```
# use a custom function
normalise <- function(vec){
    (vec * 100) / vec[1]
}
# apply with mutate and across for multiple columns
year_index = year %>%
    mutate(across(.cols = -year, normalise))
# collapse
year_index <- year %>%
    ftransformv(pedal_cycles:all_hgvs, normalise)
head(year_index)
```

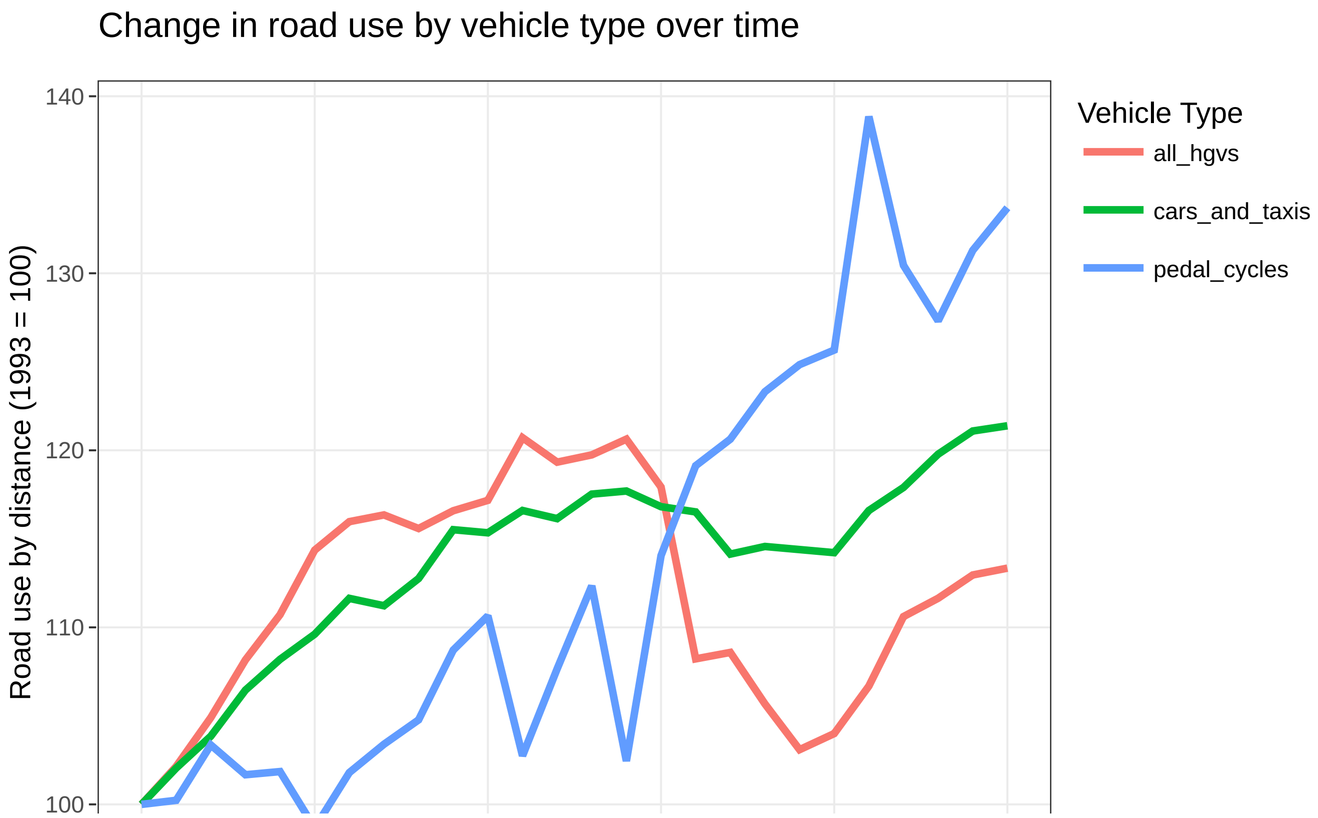
```
## # A tibble: 6 × 4
##    year pedal_cycles cars_and_taxis all_hgvs
##   <dbl>        <dbl>          <dbl>    <dbl>
## 1  1993        100             100      100
## 2  1994        100.            102.     102.
## 3  1995        103.            104.     105.
## 4  1996        102.            106.     108.
## 5  1997        102.            108.     111.
## 6  1998         98.7           110.     114.
```

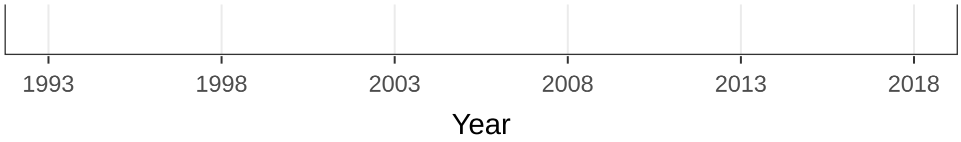
**Q3.** Having already imported and set up `pandas_bokeh` at the start of the notebook, we can now create a Bokeh plot of `year_index` simply using the `.plot()` method and setting to variable `yi_fig` .

**Do not pass any additional arguments to `.plot()`**

```
output_file(filename="bokeh.html", title="Bokeh Plot")
yi_fig = year_index.plot()
```

```
# using ggplot
yi_fig <- year_index %>%
    pivot_longer(-year) %>%
    ggplot(aes(x = year, y = value, group = name, colour = name)) +
    geom_line(lwd = 1) +
    labs(x = "Year",
         y = "Road use by distance (1993 = 100)",
         title = "Change in road use by vehicle type over time",
         colour = "Vehicle Type") +
    scale_x_continuous(breaks = seq(1993, 2018, 5)) +
    theme_bw()
#make interactive
yi_fig %>% ggplotly()
```






**Q4.** Now that you have created your `yi_fig` variable using just `.plot()` , make the following changes to the specified properties of `yi_fig` :

- change the `text` of the `title` to ‘Change in road use by vehicle type over time’
- change the `axis_label` of the `yaxis` to ‘Road use by distance (1993 = 100)’
- change the `axis_label` of the `xaxis` to ‘Year’
- remove the toolbar (by setting the `.toolbar_location` attribute to `None` )
- set the legend location to `top_left`
- change the `ticker` of the `xaxis` to use the values `[1993, 1998, 2003, 2008, 2013, 2018]`

```
yi_fig.title = "Change in road use by vehicle type over time"
yi_fig.yaxis.axis_label = 'Road use by distance (1993 = 100)'
yi_fig.xaxis.axis_label = 'Year'
yi_fig.toolbar_location = None
yi_fig.legend.location = "top_left"
yi_fig.xaxis.ticker = [1993, 1998, 2003, 2008, 2013, 2018]
```

Run the cell below to see that your changes have been implemented as expected:

```
yi_fig # opens in browser
```

```
## Figure(id='1003', ...)
```

**Q5.** Create a DataFrame called `green_2018` which: - uses only the data from `df` for 2018 - groups this 2018 data by `name` - contains the columns `['pedal_cycles', 'buses_and_coaches']` which have the `.sum()` for each group - is sorted in *descending* order by the values for `pedal_cycles` - divide all of the values in the resulting DataFrame by 1,000,000

```
green_2018 = df[df.year == 2018] \
.groupby('name')['pedal_cycles', 'buses_and_coaches'] \
.sum() \
.sort_values('pedal_cycles', ascending = False) \
/ 1000000
green_2018.head()
```

```
##                   pedal_cycles  buses_and_coaches
## name
## South East          556.344401         269.744934
## East of England     455.848666         203.142747
## London              444.469852         305.159744
## South West          357.875642         207.614416
## North West          326.663412         185.056717
```

```
# dplyr
green_2018 <- df %>%
    filter(year == 2018) %>%
    group_by(name) %>%
    summarise(across(.cols = c(pedal_cycles, buses_and_coaches),
                     ~sum(.x) %>% `/`(1000000), na.rm = TRUE)) %>%
    arrange(desc(pedal_cycles))

# collapse
green_2018 <- df %>%
    sbt(year == 2018) %>% #subset
    collap(~name, #group and summarise (collapse)
           FUN = \(x){sum(x) / 1000000},
           cols = c("pedal_cycles", "buses_and_coaches")) %>%
    roworder(-pedal_cycles) #reorder rows
head(green_2018)
```

```
## # A tibble: 6 × 3
##   name                     pedal_cycles buses_and_coaches
##   <chr>                           <dbl>             <dbl>
## 1 South East                       556.              270.
## 2 East of England                  456.              203.
## 3 London                           444.              305.
## 4 South West                       358.              208.
## 5 North West                       327.              185.
## 6 Yorkshire and The Humber         325.              185.
```

**Q6.** Use the `.plot()` method to create a *horizontal, stacked* bar chart from the `green_2018` DataFrame, assigning it to `green_bar`:

- you may find the documentation (https://patrikhlobil.github.io/Pandas-Bokeh/#barplot) useful

```
green_bar = green_2018.plot.barh(stacked = True)
```

**Q7.** Once you have created your `green_bar` variable (specifying only that it should be a stacked, horizontal bar plot), modify the following properties of your variable such that:

- the plot `.width` is `800` pixels
- the `axis_label` of the `xaxis` is ‘Vehicle miles (millions)’
- the `axis_label` of the `yaxis` is ‘Region’
- the `text` of the `title` is ‘Regional travel by bicycle and bus in 2018’

```
green_bar.width = 800
green_bar.xaxis.axis_label = 'Vehicle miles (millions)'
green_bar.yaxis.axis_label = 'Region'
green_bar.title = 'Regional travel by bicycle and bus in 2018'
```

```
green_bar <- green_2018 %>%
    pivot_longer(-name, names_to = "vehicle_type", values_to = "total_km") %>%
    ggplot(aes(x = total_km, y = fct_reorder(name, -total_km), fill = vehicle_typ
e)) +
    geom_col() +
    labs(title = 'Regional travel by bicycle and bus in 2018',
         x = 'Vehicle miles (millions)',
         y = 'Region',
         fill = "Vehicle Type")
# make file name
png_file <- fs::path(knitr::fig_path(), "green_bar.png")

# using the ragg device agg_png you can increase the scaling parameter
#to proportionately increase the scale of the geoms when the width and height are
#increased
ggsave(png_file, green_bar, device = agg_png, width = 800, height = 600, units = "
px", res = 300, scaling = 1)
```

Use `show()` to check that your changes have been made as expected:

```
green_bar
```

```
## Figure(id='1404', ...)
```

**Q8.** Create a DataFrame called `length_motor` as follows:

- group `df` by `['year', 'name']` with columns for `['total_link_length_miles', 'all_motor_vehicles']` containing the `.sum()` of these
- add a column called ‘million_vehicle_miles_per_road_mile’ which is equal to `(['all_motor_vehicles'] / 1000000) / 'total_link_length_miles'`

```
length_motor = df.groupby(['year', 'name'])['total_link_length_miles', 'all_motor_
vehicles'].sum()
length_motor['million_vehicle_miles_per_road_mile'] = \
(length_motor['all_motor_vehicles'] / 1000000) / length_motor['total_link_length_m
iles']
length_motor.head()
```

```
##                        total_link_length_miles  ...  million_vehicle_miles_per_r
oad_mile
## year name                                    ...
## 1993 East Midlands                    19064.77  ...
1.064395
##      East of England                  24052.30  ...
1.174043
##      London                            8916.95  ...
2.140143
##      North East                        9830.26  ...
1.043946
##      North West                       22339.91  ...
1.293883
##
## [5 rows x 3 columns]
```

```
#dplyr
length_motor <- df %>%
    group_by(year, name) %>%
    summarise(across(.cols = c(total_link_length_miles, all_motor_vehicles),
                     sum), .groups = "drop") %>%
    mutate(million_vehicle_miles_per_road_mile = (all_motor_vehicles / 1000000) /
total_link_length_miles)


# collapse
length_motor <- collap(df, by = ~year + ~name,
        cols = c("total_link_length_miles", "all_motor_vehicles"),
        FUN = fsum)
# by reference
settfm(length_motor,
       million_vehicle_miles_per_road_mile = (all_motor_vehicles / 1000000) / tota
l_link_length_miles)
```

**Q9.** From `length_motor`, create a new DataFrame called `reg_density` which has a row index of `year` (i.e. one row for each year 1993-2018), and a column for each region (i.e. each unique value in `name`), with the values within the DataFrame being the appropriate `million_vehicle_miles_per_road_mile` for that year in the given region:

- do not change the original `length_motor` DataFrame
- you may find `.reset_index()` and the `.pivot()` method useful
- you can refer to the documentation here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html)

```
reg_density = length_motor.copy().reset_index(level = ("name")).pivot(columns = 'n
ame')['million_vehicle_miles_per_road_mile']
reg_density.head()
```

```
## name  East Midlands  East of England  ...  West Midlands  Yorkshire and The Hum
ber
## year                                  ...
## 1993       1.064395         1.174043  ...       1.274398                   1.092
595
## 1994       1.087336         1.201897  ...       1.299053                   1.114
387
## 1995       1.107626         1.224337  ...       1.323180                   1.135
798
## 1996       1.140873         1.255611  ...       1.355891                   1.166
726
## 1997       1.163561         1.282051  ...       1.381401                   1.185
452
##
## [5 rows x 11 columns]
```

```
reg_density <- length_motor %>%
    pivot_wider(id_cols = year, names_from = name, values_from = million_vehicle_m
iles_per_road_mile)

glimpse(reg_density)
```

```
## Rows: 26
## Columns: 12
## $ year                       <dbl> 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2…
## $ `East Midlands`            <dbl> 1.064395, 1.087336, 1.107626, 1.140873, 1.1…
## $ `East of England`          <dbl> 1.174043, 1.201897, 1.224337, 1.255611, 1.2…
## $ London                     <dbl> 2.140143, 2.164728, 2.161265, 2.177550, 2.1…
## $ `North East`               <dbl> 1.043946, 1.060768, 1.076316, 1.096399, 1.1…
## $ `North West`               <dbl> 1.293883, 1.314797, 1.339661, 1.371051, 1.3…
## $ Scotland                   <dbl> 0.5968922, 0.6100507, 0.6211635, 0.6382594,…
## $ `South East`               <dbl> 1.514245, 1.547368, 1.577301, 1.625237, 1.6…
## $ `South West`               <dbl> 0.7875319, 0.8074692, 0.8231388, 0.8432015,…
## $ Wales                      <dbl> 0.6788615, 0.6939328, 0.7060715, 0.7227206,…
## $ `West Midlands`            <dbl> 1.274398, 1.299053, 1.323180, 1.355891, 1.3…
## $ `Yorkshire and The Humber` <dbl> 1.092595, 1.114387, 1.135798, 1.166726, 1.1…
```

**Q10.** As we did earlier when creating `year_index`, create a new DataFrame called `density_index`, which is the same as `reg_density` except the all values are relative to the 1993 value, which should equal `100`:

- do not modify `reg_density`

```
ninety3 = reg_density.copy().loc[1993]
density_index = (reg_density.copy() *100) / ninety3
```

```
# reuse the normalise function from before
density_index <- reg_density %>%
    mutate(across(.cols = -year, .fns = normalise))
```

```
# density_index.reset_index(inplace=True)
density_index.head()
```

```
## name  East Midlands  East of England  ...  West Midlands  Yorkshire and The Hum
ber
## year                                  ...
## 1993     100.000000       100.000000  ...     100.000000                    100.000
000
## 1994     102.155346       102.372441  ...     101.934657                    101.994
499
## 1995     104.061565       104.283762  ...     103.827818                    103.954
178
## 1996     107.185155       106.947597  ...     106.394598                    106.784
845
## 1997     109.316675       109.199620  ...     108.396338                    108.498
745
##
## [5 rows x 11 columns]
```

**Q11.** Assign to `density_plot` a figure created by using the `.plot()` method on `density_index`, with the parameter `hovertool=False`.

```
density_plot = density_index.plot(hovertool = False)
```

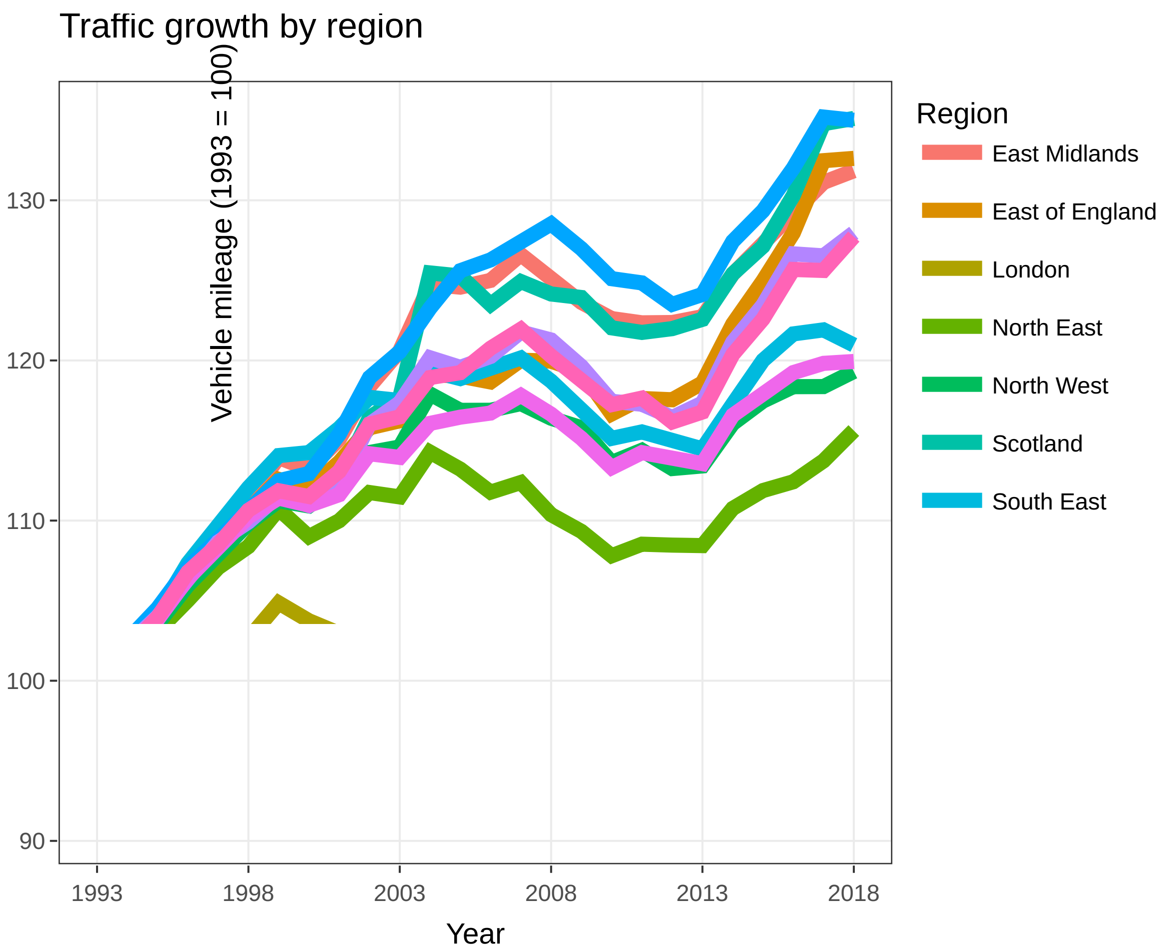
**Q12.** Make the following changes to `density_plot`:

- make the `height` and `width` both `800`
- remove the toolbar
- move the legend to the `top_left`
- use the following values on the x-axis: `[1993, 1998, 2003, 2008, 2013, 2018]`

```
density_plot.height = 800
density_plot.width = 800
density_plot.toolbar_location = None
density_plot.legend.location = "top_left"
density_plot.xaxis.ticker = [1993, 1998, 2003, 2008, 2013, 2018]
```

```
density_plot <- density_index %>%
    pivot_longer(cols = -year, names_to = "region", values_to = "miles_r_1993") %
>%
    ggplot() +
    geom_line(aes(x = year, y = miles_r_1993, colour = region),
              lwd = 2) +
    scale_x_continuous(breaks = seq(1993, 2018, 5)) +
    labs(title = "Traffic growth by region",
         x  = "Year",
         y = "Vehicle mileage (1993 = 100)",
         colour = "Region") +
    theme_bw()

density_plot %>% ggplotly()
```




Run the following cell to check your changes have been applied as expected:

```
density_plot
```

```
## Figure(id='1676', ...)
```