|  |
| --- |
| Spatial distribution of serious and fatal collisions in which people who were walking on the road network (‘pedestrians’) were hit by a car or other vehicle. |
| **Introduction**  stats19 is a new R package enabling access to and working with Great Britain’s official road traffic casualty database, [STATS19](https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data).  We started the package in late 2018 following three main motivations:   1. The release of the 2017 road crash statistics, which showed worsening road safety in some areas, increasing the importance of making the data more accessible. 2. The realisation that many researchers were writing ad hoc code to clean the data, with a huge amount of duplicated (wasted) effort and potential for mistakes to lead to errors in the labelling of the data (more on that below). 3. An understanding of the concept of ‘modularity’ in software design, following the [Unix philosophy](https://en.wikipedia.org/wiki/Unix_philosophy) that programs should ‘do one thing and do it well’. This realisation has led to code inside the rOpenSci-hosted package [stplanr](https://github.com/ropensci/stplanr) being split-out into two separate packages: [cyclestreets](https://github.com/Robinlovelace/cyclestreets) and [stats19](https://github.com/ropensci/stats19).   We have a wider motivation: we want the roads to be safer. By making data on the nature of road crashes more publicly accessible to inform policy, we hope this package saves lives.  stats19 been tested and peer reviewed thanks to rOpenSci and is now published in the Journal of Open Source Software[1](https://ropensci.org/blog/2019/02/26/stats19/#fn:1), making this an ideal time to present the package to the world 🚀.  In this post, we’ll provide a bit of context, show how the package works, and provide ideas for future work building on the experience. Version [0.2.0](https://cran.r-project.org/web/packages/stats19/index.html) has just been released on CRAN.  **Related work**  The large and complex STATS19 data from the UK’s Department for Transport, which is open access but difficult-to-use, represented a perfect opportunity for us to get stuck into a chunky data processing challenge.  **What is STATS19?**  The name comes from a UK police form called [STATS19](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/775149/Operational_Metrics_Manual.pdf) (note the capital letters). Another document called [STATS20](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/230596/stats20-2011.pdf) is the guidance for officers filling in a STATS19 form. The Department for Transport (DfT) also names the dataset [STATS19 on the main web page that links to open access road crash data](https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data).  We agree! However, the priority with the package is to remain faithful to the data, and alternative name options, such as stats19data, roadcrashesUK and roadSafetyData were not popular. Furthermore, the term ‘stats19’ is strongly associated with road crash data. On Wikipedia, the URL <https://en.wikipedia.org/wiki/STATS19> resolves to <https://en.wikipedia.org/wiki/Reported_Road_Casualties_Great_Britain>.  An important point is that the dataset omits crashes in which nobody was hurt, as emphasised by another government [document](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/743853/reported-road-casualties-gb-notes-definitions.pdf).  The importance of road safety and informed decision making based on crash data cannot be overstated. Deliberately avoiding the matter of life and death of road safety, two numbers from a strategy [document](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/8146/strategicframework.pdf) by the UK government (2011) are worth mentioning to show the scale of the numbers:  The economic welfare costs [of road collisions] are estimated at around £16 billion a year while insurance payouts for motoring claims alone are now over £12 billion a year.  While so many people die on the roads each year in the UK (1,793 people in 2017, 3 deaths per 100,000) and worldwide (1,250,000 people in 2015, 17 deaths per 100,000) and ‘vision zero’ remains a Swedish dream, we urge people researching STATS19 and other road safety datasets to focus on a more urgent question: how to stop this carnage?  **The road crash data in stats19**  There are three main different types of CSV files released by the DfT: accidents, vehicles and casualties tables. There is a schema covering these tables but a good amount of work is needed to understand it, let alone be able to process the data contained within the files and convert the integers they contain into meaningful data.  There are separate files for each of the above tables for certain years (e.g. 2016, 2017) but not for all of 1979 – 2017 or 2018 now. The largest chunk is the 1979 – 2004 data, which is made available in a huge ZIP file ([link](http://data.dft.gov.uk/road-accidents-safety-data/Stats19-Data1979-2004.zip)). Unzipped this contains the following 3 files, which occupy almost 2 GB on your hard drive:  721M Apr 3 2013 Accidents7904.csv  344M Apr 3 2013 Casualty7904.csv  688M Apr 3 2013 Vehicles7904.csv  # total 1.753 GB data  **How stats19 works**  With those introductions out of the way, lets see how the package works and how to access STATS19 data, back to 1979. First install the package in the usual way:  # release version - currently 0.2.0  install.packages("stats19")  Attach the package as follows:  library(stats19)  The easiest way to get STATS19 data is with get\_stats19(). This function takes two main arguments, year and type. The year can be any year between 1979 and 2017.  crashes\_2017 = get\_stats19(year = 2017, type = "Accidents", ask = FALSE)  nrow(crashes\_2017)  ## [1] 129982  What just happened? We just downloaded, cleaned and read-in data on all road crashes recorded by the police in 2017 across Great Britain. We can explore the crashes\_2017 object a little more:  column\_names = names(crashes\_2017)  length(column\_names)  ## [1] 32  head(column\_names)  ## [1] "accident\_index" "location\_easting\_osgr"  ## [3] "location\_northing\_osgr" "longitude"  ## [5] "latitude" "police\_force"  class(crashes\_2017)  ## [1] "spec\_tbl\_df" "tbl\_df" "tbl" "data.frame"  kableExtra::kable(head(crashes\_2017[, c(1, 4, 5, 7, 10)]))   | **accident\_index** | **longitude** | **latitude** | **accident\_severity** | **date** | | --- | --- | --- | --- | --- | | 2017010001708 | -0.080107 | 51.65006 | Fatal | 2017-08-05 | | 2017010009342 | -0.173845 | 51.52242 | Slight | 2017-01-01 | | 2017010009344 | -0.052969 | 51.51410 | Slight | 2017-01-01 | | 2017010009348 | -0.060658 | 51.62483 | Slight | 2017-01-01 | | 2017010009350 | -0.072372 | 51.57341 | Serious | 2017-01-01 | | 2017010009351 | -0.353876 | 51.43876 | Slight | 2017-01-01 |   The package contains the names of all “zip” files released by the DfT and hosted on Amazon servers to download. These file names have been included in the package and can be found under file\_names variable name. for example:  stats19::file\_names$dftRoadSafetyData\_Vehicles\_2017.zip  ## [1] "dftRoadSafetyData\_Vehicles\_2017.zip"  You can also get the raw data (if you really want!) to see how much more useful the data is after it has been cleaned and labelled by the stats19 package, compared with the data provided by government:  crashes\_2017\_raw = get\_stats19(year = 2017, type = "Accidents", ask = FALSE, format = FALSE)  ## Files identified: dftRoadSafetyData\_Accidents\_2017.zip  ## http://data.dft.gov.uk.s3.amazonaws.com/road-accidents-safety-data/dftRoadSafetyData\_Accidents\_2017.zip  ## Data already exists in data\_dir, not downloading  ## Data saved at /tmp/RtmpYh76XA/dftRoadSafetyData\_Accidents\_2017/Acc.csv  ## Reading in:  ## /tmp/RtmpYh76XA/dftRoadSafetyData\_Accidents\_2017/Acc.csv  kableExtra::kable(head(crashes\_2017\_raw[, c(1, 4, 5, 7, 10)]))  The first two columns are raw read, the next two are formatted by stats19 package:  kableExtra::kable(cbind(  head(crashes\_2017\_raw[1:2, c(7, 10)]),  head(crashes\_2017[1:2, c(7, 10)])  ))   | **Accident\_Severity** | **Date** | **accident\_severity** | **date** | | --- | --- | --- | --- | | 1 | 05/08/2017 | Fatal | 2017-08-05 | | 3 | 01/01/2017 | Slight | 2017-01-01 |   Note: the severity type is not labelled (this problem affects dozens of columns), the column names are inconsistent, and the dates have not been cleaned and converted into a user-friendly date (POSIXct) class.  **Creating geographic crash data**  An important feature of STATS19 data is that the “accidents” table contains geographic coordinates. These are provided at ~10m resolution in the UK’s official coordinate reference system (the Ordnance Survey National Grid, EPSG code 27700). stats19 converts the non-geographic tables created by format\_accidents() into the geographic data form of the sf package with the function format\_sf() as follows:  crashes\_sf = format\_sf(crashes\_2017)  ## 19 rows removed with no coordinates  # crashes\_sf = format\_sf(crashes\_2017, lonlat = TRUE) # provides the data in lon/lat format  An example of an administrative zone dataset of relevance to STATS19 data is the boundaries of police forces in England, which is provided in the packaged dataset police\_boundaries. The following code chunk demonstrates the kind of spatial operations that can be performed on geographic STATS19 data, by counting and plotting the number of fatalities per police force:  library(sf)  library(dplyr)  crashes\_sf %>%  filter(accident\_severity == "Fatal") %>%  select(n\_fatalities = accident\_index) %>%  aggregate(by = police\_boundaries, FUN = length) %>%  plot()    # as Layik and Robin are based in West Yorkshire  west\_yorkshire =  police\_boundaries[police\_boundaries$pfa16nm == "West Yorkshire", ]  crashes\_wy = crashes\_sf[west\_yorkshire, ]  nrow(crashes\_wy) # which is 3.36%  ## [1] 4371  **The big picture: road safety**  We can combine the three sets of tables (accidents, vehicles and casualties) to analyse the data further. Lets read the datasets first:  #crashes\_2017 = get\_stats19(year = 2017, type = "Accidents", ask = FALSE)  casualties\_2017 = get\_stats19(year = 2017, type = "Casualties", ask = FALSE)  nrow(casualties\_2017)  ## [1] 170993  vehicles\_2017 = get\_stats19(year = 2017, type = "Vehicles", ask = FALSE)  nrow(vehicles\_2017)  ## [1] 238926  Lets now read in casualties that took place in West Yorkshire as the authors are based in West Yorkshire (using crashes\_wy object above), and count the number of casualties by severity for each crash:  library(tidyr)  library(dplyr)  sel = casualties\_2017$accident\_index %in% crashes\_wy$accident\_index  casualties\_wy = casualties\_2017[sel, ]  cas\_types = casualties\_wy %>%  select(accident\_index, casualty\_type) %>%  group\_by(accident\_index) %>%  summarise(  Total = n(),  walking = sum(casualty\_type == "Pedestrian"),  cycling = sum(casualty\_type == "Cyclist"),  passenger = sum(casualty\_type == "Car occupant")  )  cj = left\_join(crashes\_wy, cas\_types)  What just happened?  We found the subset of casualties that took place in West Yorkshire with reference to the accident\_index variable in the accidents table. Then we used the dplyr function summarise(), to find the number of people who were in a car, cycling, and walking when they were injured. This new casualty dataset is joined onto the crashes\_wy dataset. The result is a spatial (sf) data frame of crashes in West Yorkshire, with columns counting how many road users of different types were hurt. The joined data has additional variables:  base::setdiff(names(cj), names(crashes\_wy))  ## [1] "Total" "walking" "cycling" "passenger"  In addition to the Total number of people hurt/killed, cj contains a column for each type of casualty (cyclist, car occupant, etc.), and a number corresponding to the number of each type hurt in each crash. It also contains the geometry column from crashes\_sf. In other words, joins allow the casualties and vehicles tables to be geo-referenced. We can then explore the spatial distribution of different casualty types. The following figure, for example, shows the spatial distribution of pedestrians and car passengers hurt in car crashes across West Yorkshire in 2017:  library(ggplot2)  crashes\_types = cj %>%  filter(accident\_severity != "Slight") %>%  mutate(type = case\_when(  walking > 0 ~ "Walking",  cycling > 0 ~ "Cycling",  passenger > 0 ~ "Passenger",  TRUE ~ "Other"  ))  ggplot(crashes\_types, aes(size = Total, colour = speed\_limit)) +  geom\_sf(show.legend = "point", alpha = 0.3) +  facet\_grid(vars(type), vars(accident\_severity)) +  scale\_size(  breaks = c(1:3, 12),  labels = c(1:2, "3+", 12)  ) +  scale\_color\_gradientn(colours = c("blue", "yellow", "red")) +  theme(axis.text = element\_blank(), axis.ticks = element\_blank())  Spatial distribution of serious and fatal collisions in which people who were walking on the road network ('pedestrians') were hit by a car or other vehicle. |
| To show what is possible when the data are in this form, and allude to next steps, let’s create an interactive map. We will plot crashes in which pedestrians were hurt, from the crashes\_types, using leaflet package:  library(leaflet)  crashes\_pedestrians = crashes\_types %>%  filter(walking > 0)  # convert to lon lat CRS  crashes\_pedestrians\_lonlat = st\_transform(crashes\_pedestrians, crs = 4326)  pal = colorFactor(palette = "Reds", domain = crashes\_pedestrians\_lonlat$accident\_severity, reverse = TRUE)  map = leaflet(data = crashes\_pedestrians\_lonlat, height = "280px") %>%  addProviderTiles(provider = providers$OpenStreetMap.BlackAndWhite) %>%  addCircleMarkers(radius = 0.5, color = ~pal(accident\_severity)) %>%  addLegend(pal = pal, values = ~accident\_severity) %>%  leaflet::addMiniMap(toggleDisplay = TRUE)  The R leaflet package is not easy to embed in an .md file for the rOpenSci blog entry. Therefore, we also wrote a “custom” LeafletJS chunk below and use htmltools to embed the interactive map here. Feel free to use the chunk in your own work by replacing the geojson content generated from the preceding code.  library(geojsonsf)  library(htmltools)  geojson = sf\_geojson(  crashes\_pedestrians\_lonlat[,c("accident\_severity")],  factors\_as\_string = FALSE)  template = paste0('  ')  path = file.path(tempdir(), "temp.html")  write(template, path)  includeHTML(path)  The result is:  **Conclusion**  [stats19](https://github.com/ropensci/stats19) provides access to a reliable and official road safety dataset. As covered in this post, it helps with data discovery, download, cleaning and formatting, allowing you to focus on the real work of analysis. In our experience, 80% of time spent using STATS19 data was spent on data cleaning. Hopefully, now the data is freely available in a more useful form, 100% of the time can be spent on analysis! We think it could help many people, especially, including campaigners, academics and local authority planners aiming to make the roads of the UK and the rest of the world a safe place for all of us.  There are many possible next steps, including:   * Comparing these datasets with interventions such as 20 mph zones and links with street morphology. * The creation of more general software for accessing and working with road crash data worldwide. * Making the data even more available by provide the data as part of an interactive web application |