A novel way to visulaize event time data

true

2024-09-17

Continue on article repo

Abstract

Visualization is immensely important in statistical analysis and prediction. It helps in describing the data as well as in choosing the appropriate method of modeling and prediction. So far, however, there is no way to visualize event time data on a clock. In this paper we present a few novel ways to visualize various event time data on chart we call 'Data Clock'. Given events ocurring at various times in a 24-hour day, we show the events on a clock corresponding to the time of event. This makes it more useful to perceive exactly when the events occurred. In addition, we can easily compare the time of the event with other events. Time of events plotted on a clock chart is subtantially more revealing than a bar chart or a pie chart. It can demonstrate around which time events are less or more frequent as well the gap between the events. We also introduce **these functions**.

Introduction

Methodology

Broadly speaking, the construction of the clockcharts have two steps.

i. Construction of the 24-hour clock
ii. Plotting event times on the clock

Each steps has two substeps:

a. The mathematical idea
b. The programming

The Clock Skeleton

First we need to construct the skeleton, the clock on which event times are plotted. For this, we need a circle, on which we have to mark 24 dots to denote 24 hours, starting from 00:00 (12 AM) to 23:00 (11 PM).

It is enough to draw a unit circle, although a circle of any radius would do. In `R`, it is easiest to draw a circle using the complex equation. For a unit circle, the equation is $e^{i\theta}$, where e is Euler's number, an irrational constant having the approximate value of 2.71828; i stands for imaginary unit, equal to $\sqrt{-1}$; and θ refers to the trigonometric angle in radian. To make a complete circle, we need to allow θ to take values from 0 to 360 degrees (2π in radian).

Programmatically, we need to create some angles ranging from 0 to 2π and then connect them to make a complete circle. Let us create 100 angles and use them in the circle equation ($e^{i\theta}$). The trigonometric form of the equation is $sin^2\theta + cos^2\theta = 1$, while the algebraic form is $x^2 + y^2 = 1$. The complex form, however, is more convenient.

```r
k <- 100
timepoint <- exp(1i * 2 * pi * (k:1) / k)
plot(timepoint, pch = 19, type = "b")
```

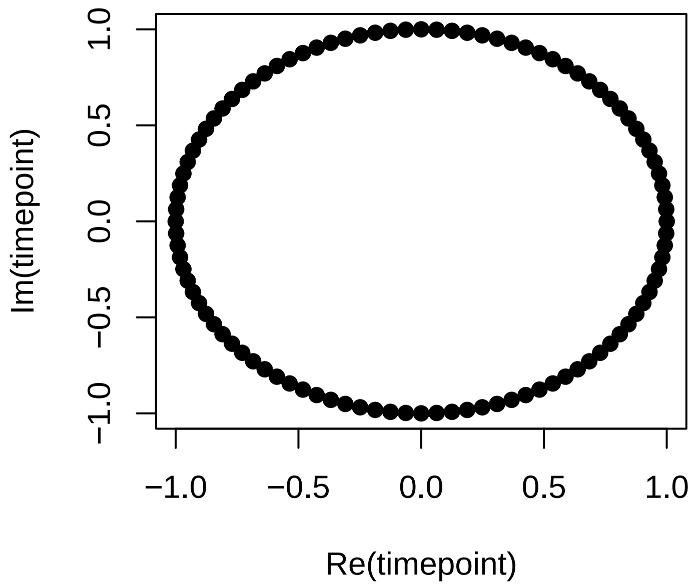


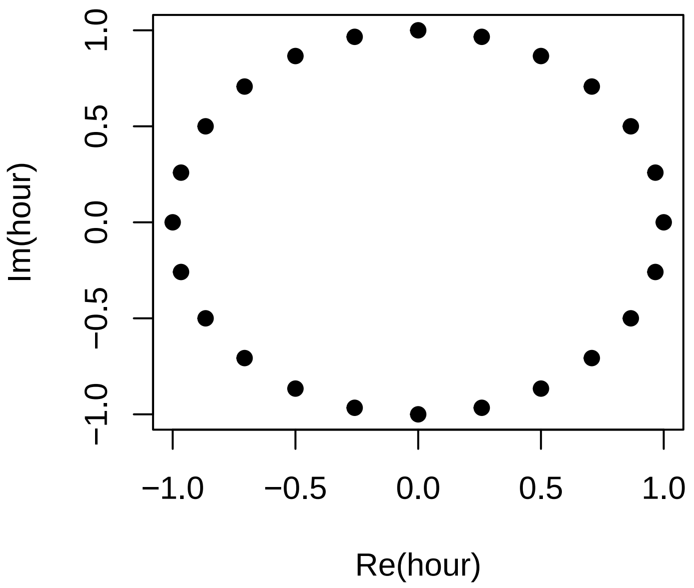
Figure 1: The Unit Circle

Here R just takes the complex numbers and plots the values of the real part on the X axis and the corresponding values of the imaginary part on the Y axis.

Now we have to mark the hours on the circle to make it look like a clock. We have to show 24 hours marks as well some marks for minute marks, corresponding to the fraction of the hour. This clock is slightly different from a traditional clock not only in that it has 24 hours instead of 12, but also that it has only one hand, the hour hand, with minute part added to the hour mark as the fraction; the second part, having minor contribution, is ignored.

First, let us see the output with 24 hour marks.

```r
k <- 24
hour <- exp(1i * 2 * pi * (k:1) / k)
plot(hour, pch = 19)
```



At this time, we move to the `R` package `ggplot2` to produce better graphics. We need to create an `R` data frame for plotting ggplots. We also mark some points for minutes.

First, let us create the required data frame

```
k <- 24 # Hours
subk <- 24*4 # Fraction of hours
times <- exp(1i * 2 * pi * (k:1) / k)
subtimes <- data.frame(SubT = exp(1i * 2 * pi * (subk:1) / subk))
ampm = c(rep(" AM",6), rep(" PM",12), rep(" AM",6))
library(tibble)
dfclock <- tibble(time = times,
                  hour = c(6:12, 1:12, 1:5), # May not be needed
                  label = paste0(c(6:12, 1:5), ampm))
```

Explanation

`k = 24` indicates there are 24 hours. Then in each hour is divided into 4 parts , corresponding to the elapse of successive 15 minutes. Thus, a small point on @ref(fig:clock24) in the middle of two larger point (hour) means half an hour has elapsed. For instance, the the point in the middle of 7 PM and 8 PM is 7:30 PM.

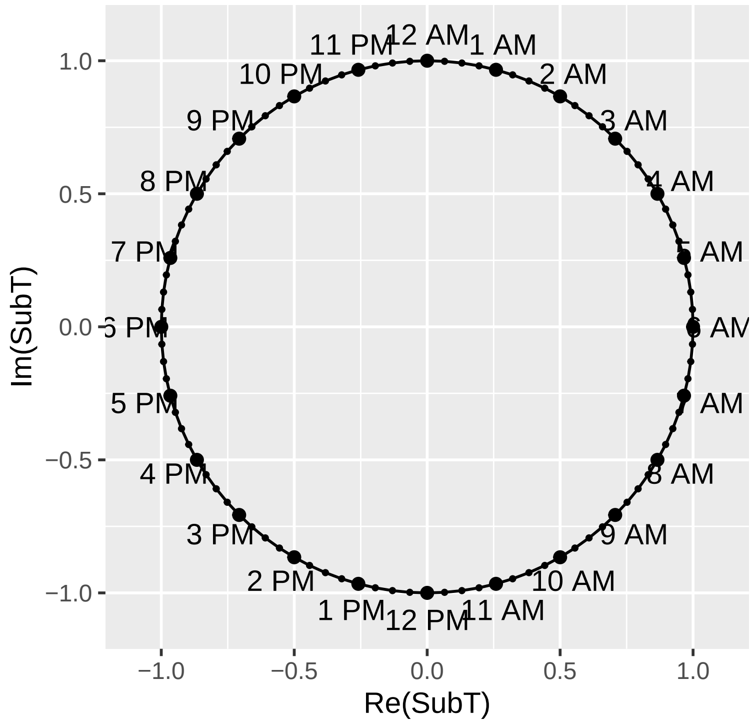
Now the code `exp(1i * 2 * pi * (k:1) / k)` gives us all the angles corresponding to 24 hours on the clock. Here, we have to note that the angles in trigonometry starts from 0^o (X-axis) and rotates counter-clockwise, the direction opposite to the rotation of clock hands (the significance of the difference will be seen shortly). But the clock moves clockwise. At 0 degree, we have 3 in a 12-hour clock and 6 in 24-hours clock.

Similarly, the code `SubT = exp(1i * 2 * pi * (subk:1) / subk)` gives us the angles corresponding to the fraction of an hour. As we have already mentioned, the clock hand rotates clock-wise, starting from the 0^o angle, so we name the labels 6 PM to 11 PM, then we have 12 PM to 11 PM, followed by 12 AM to 5 AM.

The code `c(rep(" AM",6), rep(" PM",12), rep(" AM",6))` ensures this sequence.

We then place the data into two data frames `subtimes` and `dfclock`, since they are unequal dimensions.

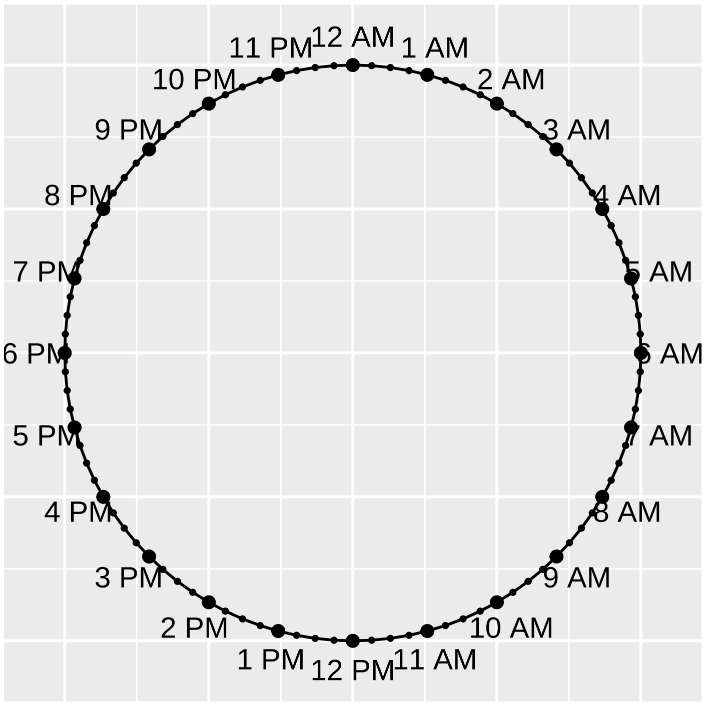
```r
library(ggplot2) # Load the package ggplot2
p1 <- dfclock %>% ggplot()+
  geom_path(data = subtimes, aes(Re(SubT), Im(SubT)))+
  # Connect Last two missing points
  geom_line(data = subtimes %>% dplyr::slice(-c(2:95)), aes(Re(SubT), Im(SubT)))+
  theme(aspect.ratio = 1)+
  geom_text(data = dfclock, aes(Re(time)*1.1, Im(time)*1.1, label = label))+
  geom_point(data = subtimes,
             aes(Re(SubT), Im(SubT)), shape = 19, color = "black", size = 0.6)+
  geom_point(aes(Re(time), Im(time)), color = "black", size = 1.8)
p1
```



Next, we plot the lines using the `ggplot2` package. The functions `geom_path()` does the job, while `geom_line()` is used to fill the gap between the first and the last point. `aspect.ratio = 1` is set to ensure the chart size do not depend on resizing of `Rstudio` Viewer size.

The points corresponding to the hours are labelled AM or PM using the `geom_text()` function. We do not need the axis labels and ticks. Let us remove them:

```r
p1 + theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank())+
  labs(x = "", y = "")
```



This completes our first part of the plot; we get the skeleton of the clock.

The second and final part of the task involves plotting event times on the appropriate points. Now, this task has several variations. In the simplest form, we just have a chart, so called `clock_chart`, which plots the event times on the 24-hour clock. An optional argument `Col` can be used to change the color of the clock hands. The other variations will be discussed later, when we complete the discussion on how the event times are prepared for plotting.

Preparation of Event Times Data We have data in the format `HH:MM:SS`. We need to convert this data to `(x,y)` coordinates. In the unit circle, for example, the 12 AM should be plotted on the Y-axis (x = 0, y = 1) and 6 AM on the X axis (x = 1, y = 0), although, to avoid plotting on the circumference, we decrease the length a little (more on this later).

To convert the `HH:MM:SS` to coordinates of the unit circle, we first separate the parts of time into `HH`, `MM`, and `SS`. We accomplish that using the `dplyr` and `tidyr` package.

```
# Data to check on
chkdf <- data.frame(time = c("06:00:00", "12:00:00", "17:30:00", "00:05:25"),
                    value = sample(10,4))
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
chkdf %>% tidyr::separate_wider_delim(cols = time,
                          names = c("hour", "minute", "second"),
                          cols_remove = FALSE,
                          delim = ":")
```

```
## # A tibble: 4 x 5
##   hour  minute second time     value
##   <chr> <chr>  <chr>  <chr>    <int>
## 1 06    00     00     06:00:00     4
## 2 12    00     00     12:00:00    10
## 3 17    30     00     17:30:00     5
## 4 00    05     25     00:05:25     3
```

This gives us `hour`, `minute`, and `second` as character (`chr`) vectors, but we need them as numeric vectors. The conversion is made with `as.numeric` function.

Here the challenge now is to combine the hour and the minute into one singe time, which we would later convert to coordinate. We intentionally ignore the second (`SS`), since it has too negligible contribution on the plot.

Here a short algorithm is to be used:

Note here that 30 minutes is equivalent to half an hour, so `1:30:00` corresponds to 1.5 hour. We need to recall that time values range from 0 to 60, while our decimal number system gives us the suitable range 1 to 100. In our number system, reaching 100 indicates we have completed some work, while clockworks say reaching 60 completes one cycle of the task.

The hour part does not need conversion. 23 is 23 and 22 is 22.

Thus, 15 minutes on the clock says $\frac{15}{60} = \frac{1}{4}$ th or 25% work, 30 means $\frac{30}{60} = \frac{1}{2} = 0.5$ or 50% work, and 50 means $\frac{50}{60}$ or 83% work have been completed.

The task is further complicated due to the clock and angles rotating in opposite directions.

////THIS CONVERSION IS A BUG//// ifelse(.data$minute < 10, .data$minute * 5/30, .data$minute * 5/300) ////MAY DEPEND ON OLD TIME FORMAT HHMM hrs/////

Clock Skeleton

Plotting Times on the Clock

Discussion

Let us explore US Accidents data. Although the main focus of these data seem to be on accident times, we cna employ it to visualize the comparison between the existing line chart and our proposed clock chart.

```r
library(clockplot)
library(ggplot2)
acdt <- read.csv("https://raw.githubusercontent.com/mahmudstat/open-analysis/main/data/usacc.csv")
acdt <- acdt[,c(9:12)]
head(acdt)
```

```
##   Temperature.F. Humidity... Weather_Condition     Time
## 1           36.9          91        Light Rain 05:46:00
## 2           37.9         100        Light Rain 06:07:59
## 3           36.0         100          Overcast 06:49:27
## 4           35.1          96     Mostly Cloudy 07:23:34
## 5           36.0          89     Mostly Cloudy 07:39:07
## 6           37.9          97        Light Rain 07:44:26
```

Let us plot temperature against time.

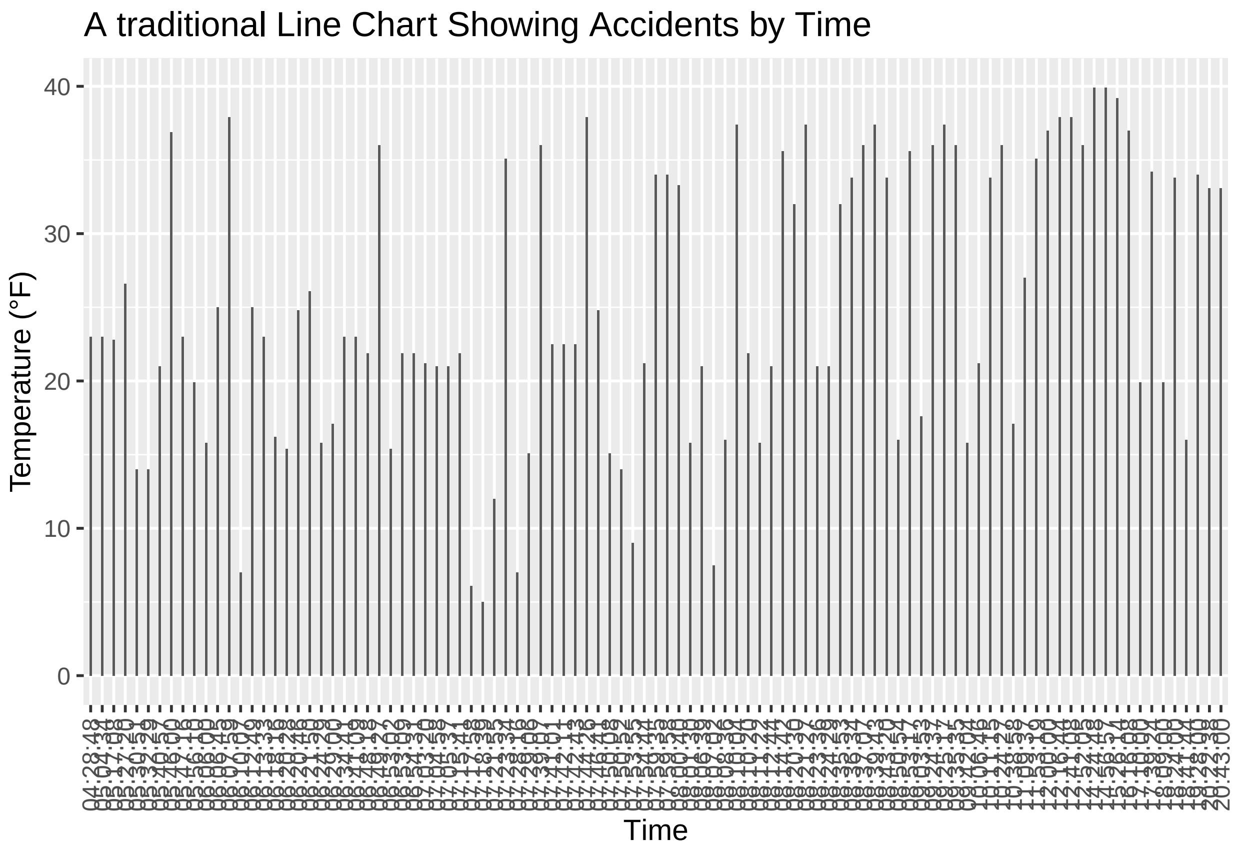
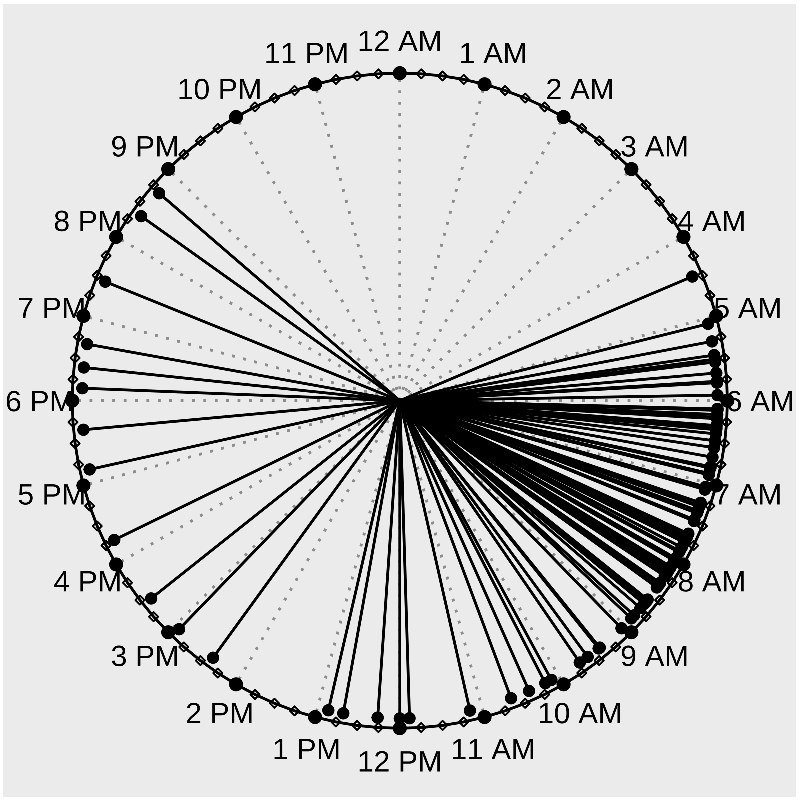


Figure 2: CAP

We have no idea which temperature belongs to which time, although we have changed X- axis labels at right angles (i.e., vertically).

Now let us take a look into a clock plot.



Although this is the simplest kind of clock chart that we propose, this still makes a lot more sense. We clearly see a huge gap between around 9 PM to around 4:30 AM, during which period no reading of temperature was made (in other example, this would mean no event occurred, which is actually relevant to this data set as well, which records times of accidents. Why no accidents occurred during this period is subject to further discussion; perhaps, the traffic was less during this time.)

We can easily observe that most incidents occurred from 5 AM to 9 AM. Between other hours, roughly two accidents occurred.

Let us try plotting 20 values only.

When a small number of values are plotted, the line chart becomes much better than its previous counterpart. There are, however, several limitations compared to its clockplot counterpart.

a. It is hard to discern exactly when an event occurred (or recorded). What we observe is just the trend of the temperature.

b. It gives us an idea that recordings are placed at equal intervals. Actually, the time gap between the first and the second recordings is around 8 hours, while the next gap is worth 36 minutes only. Thus, any trend, either visual or analytical, would be extremely faulty.
c. Unlike the clockplot, we cannot single out the duration when there are no, less, or large number of occurrences. Occurrences, as we have seen in the previous clockplot, may be rare during night and early morning. We, however, cannot extract that information from the line chart.
d. When we plot other periods (day, month, year, and so on), they are not necessarily cyclic. We get a continuous flow of time, going ahead linearly: year 2015, year 2016, year 2017 etc; no year is repeated. Time data are, however, cyclic; after 24 hours, each hour starts repeating itself. A linear line chart fails to reveal this cyclic behavior. As an example, the first and second value might represent the end of a day (a cycle) and the start of a new day. A legend may help see this; clockplot still serves the purpose

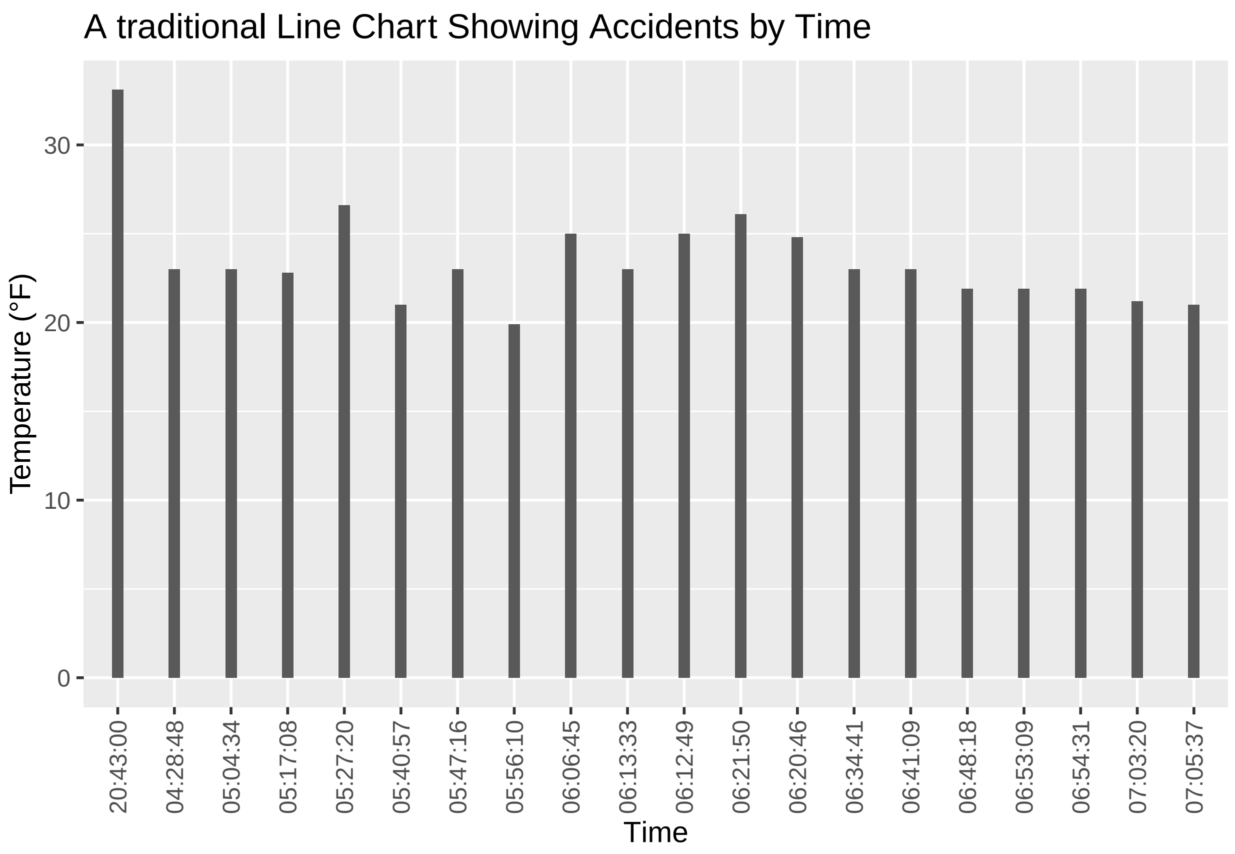

A traditional Line Chart Showing Accidents by Time
Temperature (°F)
0
10
20
30
20:43:00
04:28:48
05:04:34
05:17:08
05:27:20
05:40:57
05:47:16
05:56:10
06:06:45
06:13:33
06:12:49
06:21:50
06:20:46
06:34:41
06:41:09
06:48:18
06:53:09
06:54:31
07:03:20
07:05:37
Time

Figure 3: CAP

better. In short, the line chart cannot reveal the gap between the last and first.

In the hospital data, for example, all the admissions take place between 2 Am and 1 PM. A line chart will just plot the events between this period, without revealing there were no admissions between 1 PM and 2 AM.

e. The line chart fails to show that the events are actually clustered around 5 AM to 7 AM, as seen in the Figure @ref{fig:acdt20}.
f. If we have no values against times, for example, we only have timings of events, such as when patients were admitted in a hospital, we cannot present this on a line chart. To circumvent the problem, we may choose an arbirary value against each time, but this does not allow us to focus on the timings of the events.

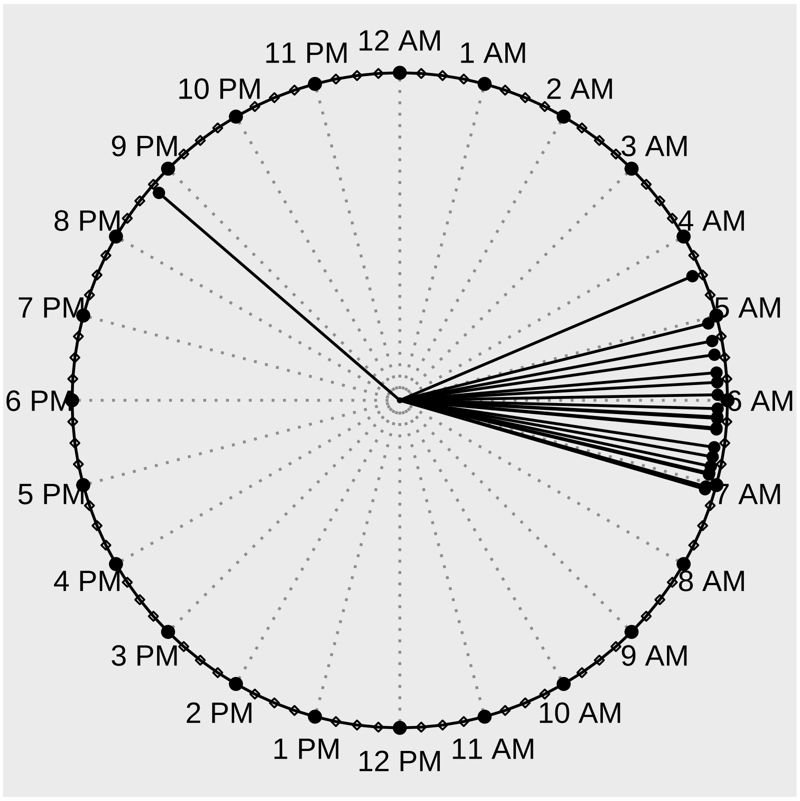


Figure 4: CAP

Need cyclic

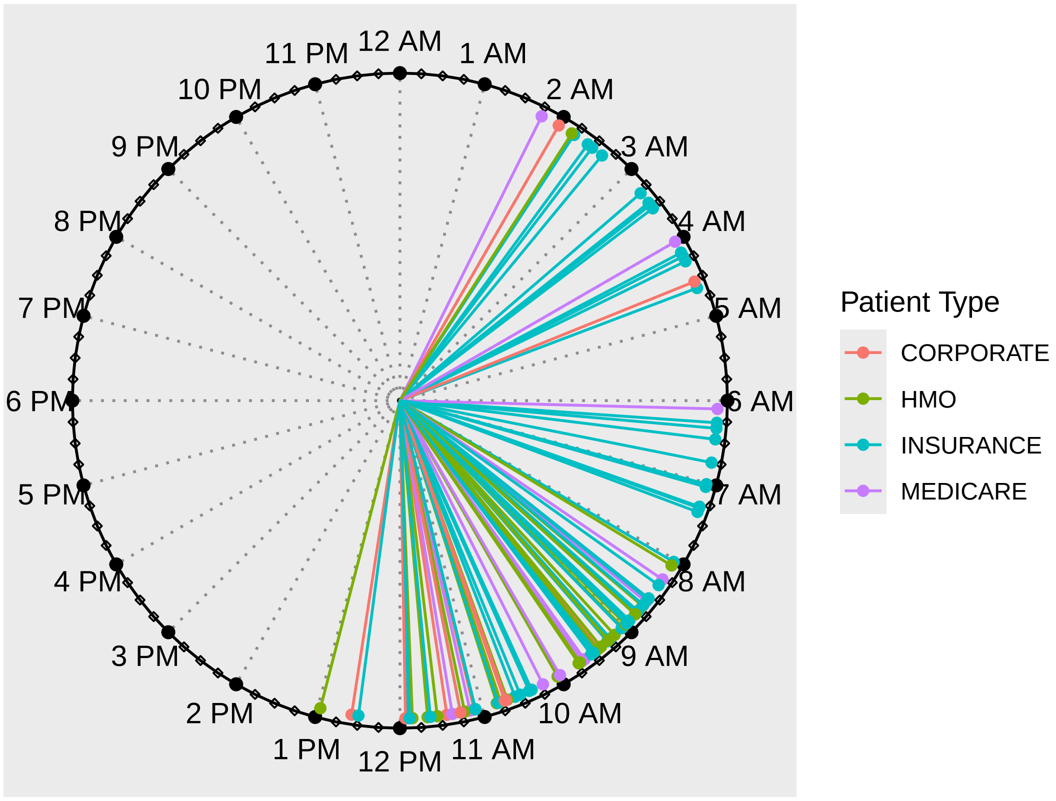

12 AM
1 AM
2 AM
3 AM
4 AM
5 AM
6 AM
7 AM
8 AM
9 AM
10 AM
11 AM
12 PM
1 PM
2 PM
3 PM
4 PM
5 PM
6 PM
7 PM
8 PM
9 PM
10 PM
11 PM
Patient Type
CORPORATE
HMO
INSURANCE
MEDICARE

Figure 5: CAP