forecast_timeseries_egjohnson

Elizabeth

March 27, 2019

1. Read in data

```
# read in data as character
tcdat <- fread("app_data.csv",colClasses=c("character"))
tcdat %>% head
```

begin_time	average_response_time	throughput
1508025600	203.20011539812356	119586
1508025660	144.14319113189268	112493
1508025720	127.0975367190018	108595
1508025780	134.4365023956849	104563
1508025840	127.95972196725687	119842
1508025900	138.5803969580254	111901

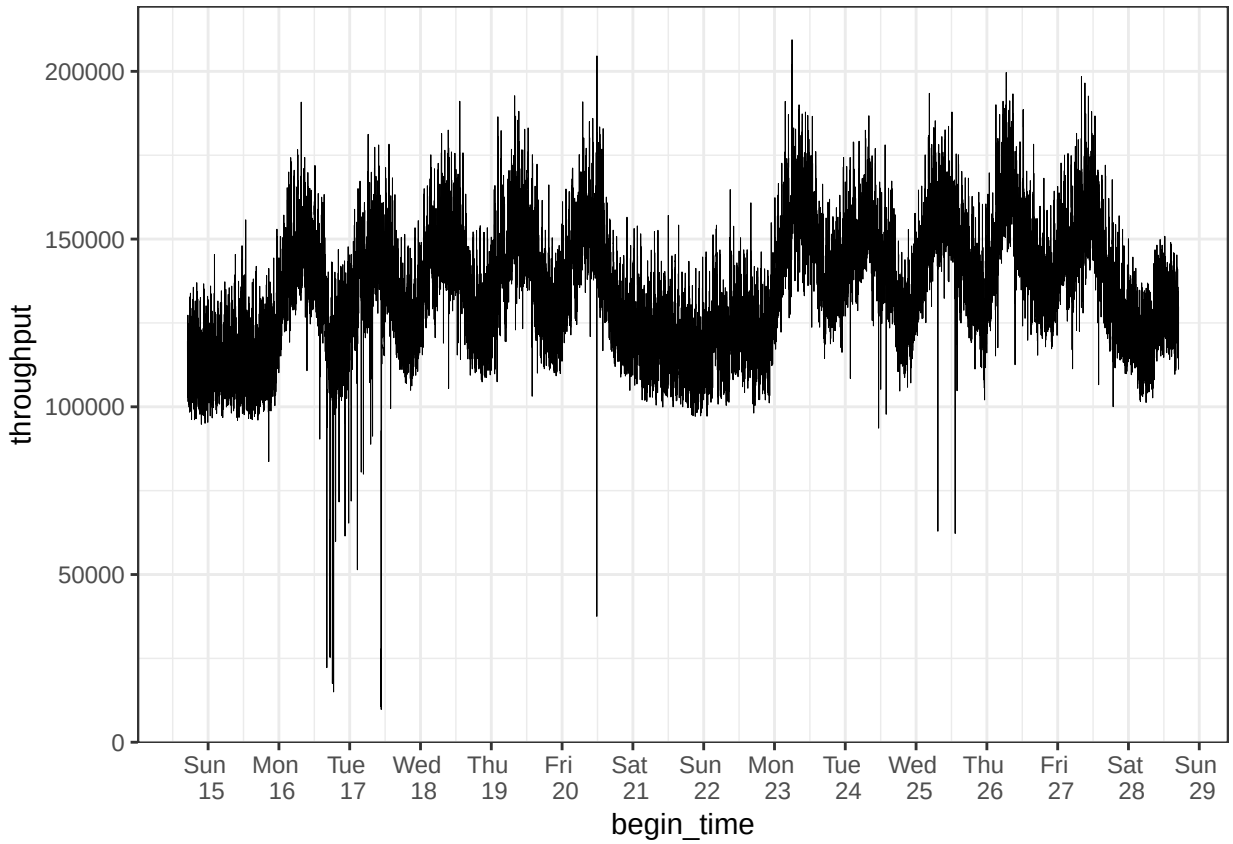
```
#change UNIX timestamp to actual date
#seconds since Jan 01 1970. (UTC)
#convert characters to numeric
library(tidyr)
tcdat %>% dplyr::mutate(begin_time=anytime(as.integer(begin_time))) %>% dplyr::mutate_if(is.character,as

tcdat %>% head
```

begin_time	average_response_time	throughput
2017-10-14 17:00:00	203.2001	119586
2017-10-14 17:01:00	144.1432	112493
2017-10-14 17:02:00	127.0975	108595
2017-10-14 17:03:00	134.4365	104563
2017-10-14 17:04:00	127.9597	119842
2017-10-14 17:05:00	138.5804	111901

2. Inspect Data

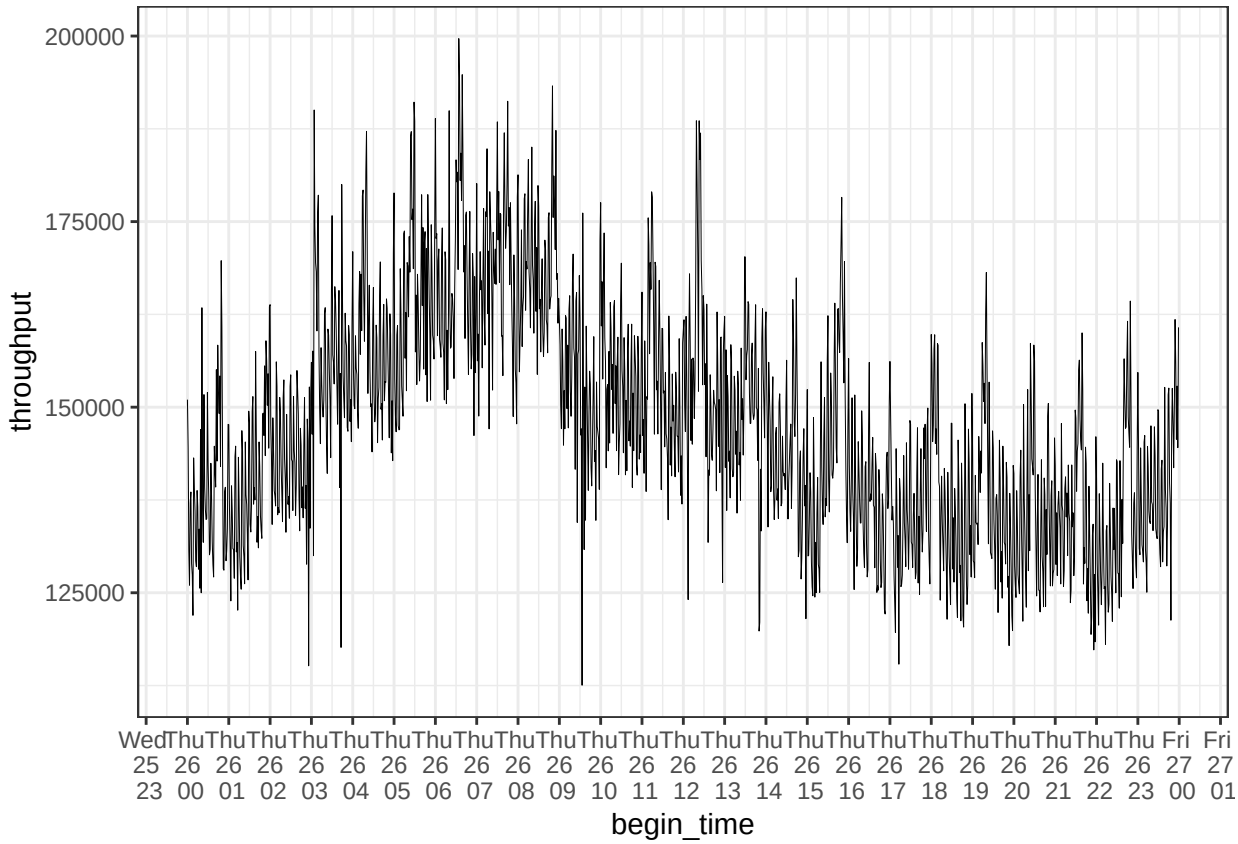
Throughput peaks in the middle of the day - but only on weekdays

```
ggplot(tcdat, aes(x = begin_time, y = throughput)) +
  geom_line(size =0.1) +
  scale_x_datetime(breaks = date_breaks("1 day"), labels=date_format("%a \n %d",tz=Sys.timezone()))+
  theme_bw()
```



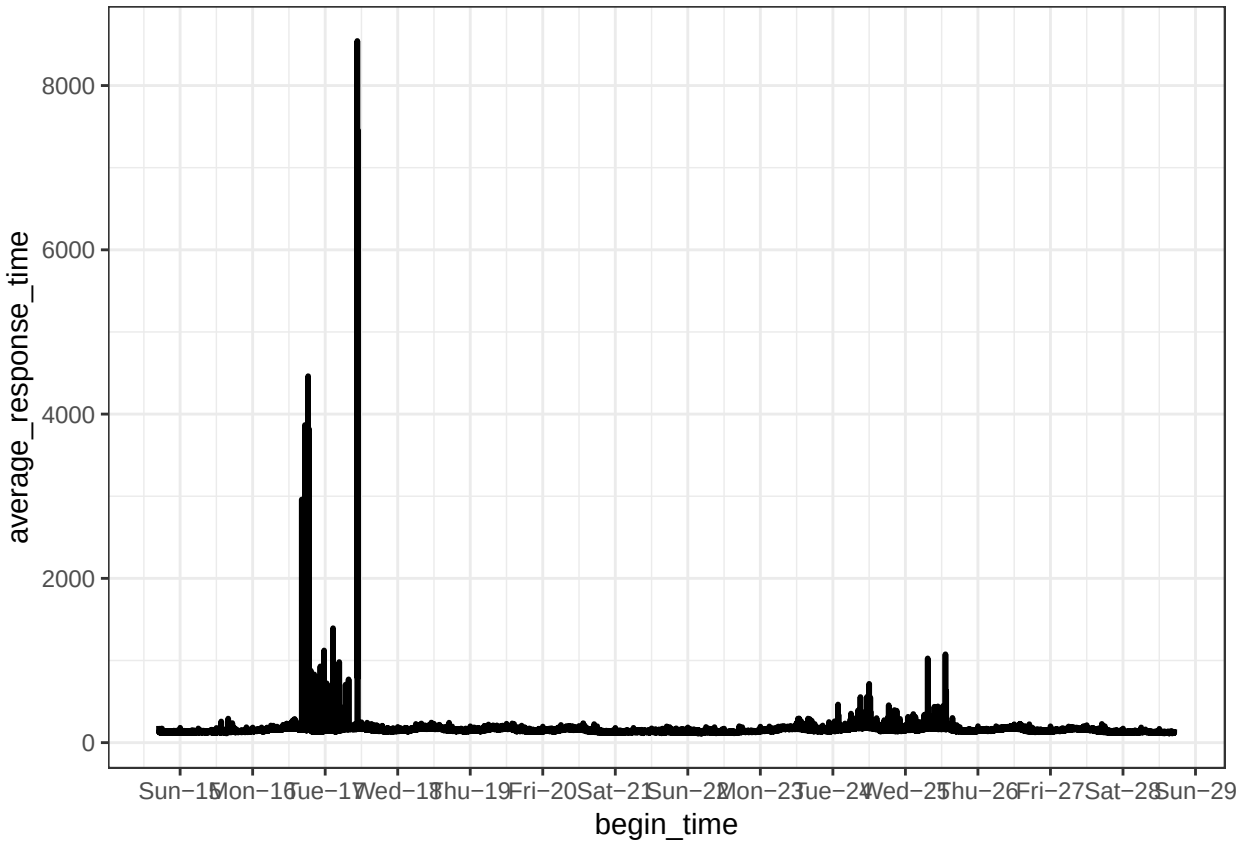
On a weekday throughput typically peaks prior to noon then tapers off.

```
tcdat %>% dplyr::filter(day(begin_time) ==26)->tc.plot

ggplot(tc.plot, aes(x = begin_time, y = throughput)) +
  geom_line(size =0.1) +
  scale_x_datetime(breaks = date_breaks("1 hour"),labels=time_format("%a \n %d \n %H",tz=Sys.timezone()))
  theme_bw()
```



response time spikes on some weekdays

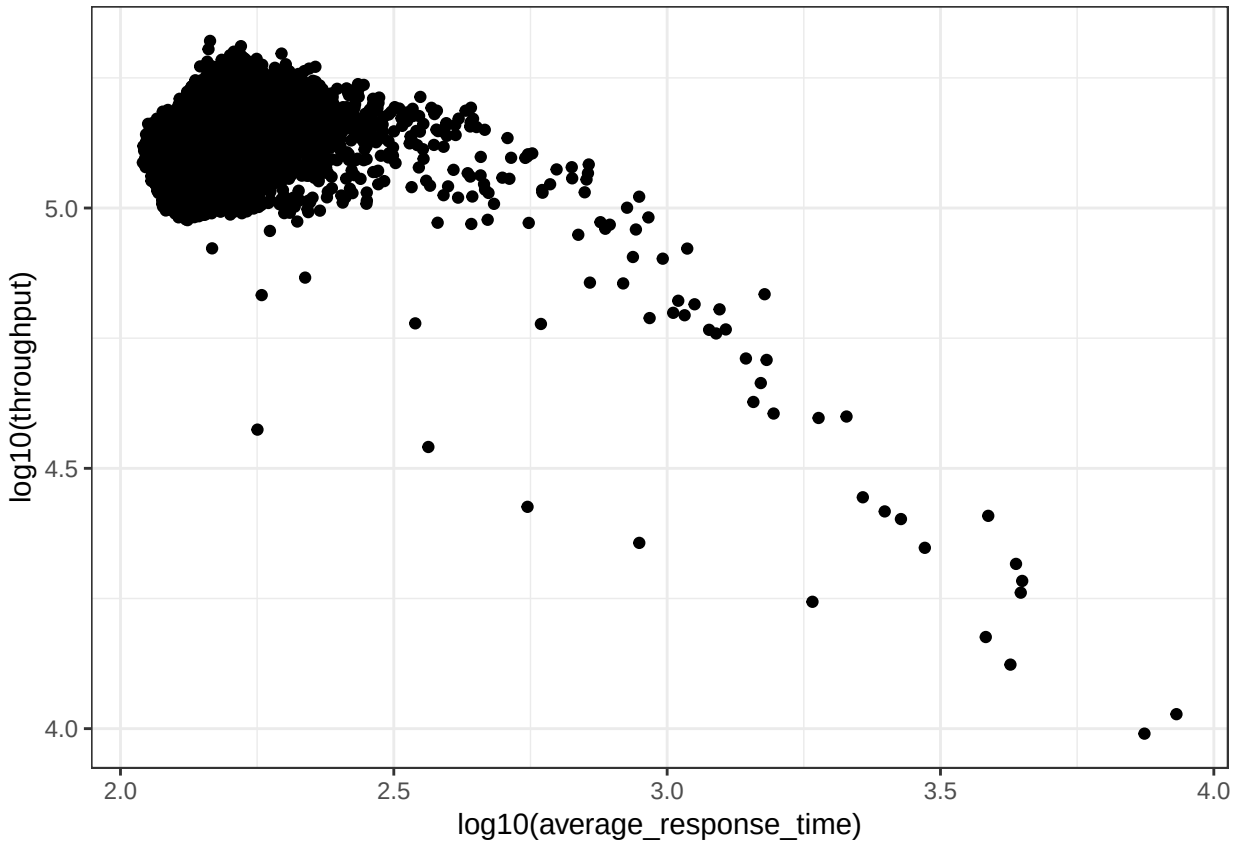
```
ggplot(tcdat, aes(x = begin_time, y = average_response_time)) +
  geom_line(size = 1) +
  scale_x_datetime(breaks = date_breaks("1 day"), labels=date_format("%a-%d",tz=Sys.timezone()))+
  theme_bw()
```



3. Characterize relationship/processes linking two data sets

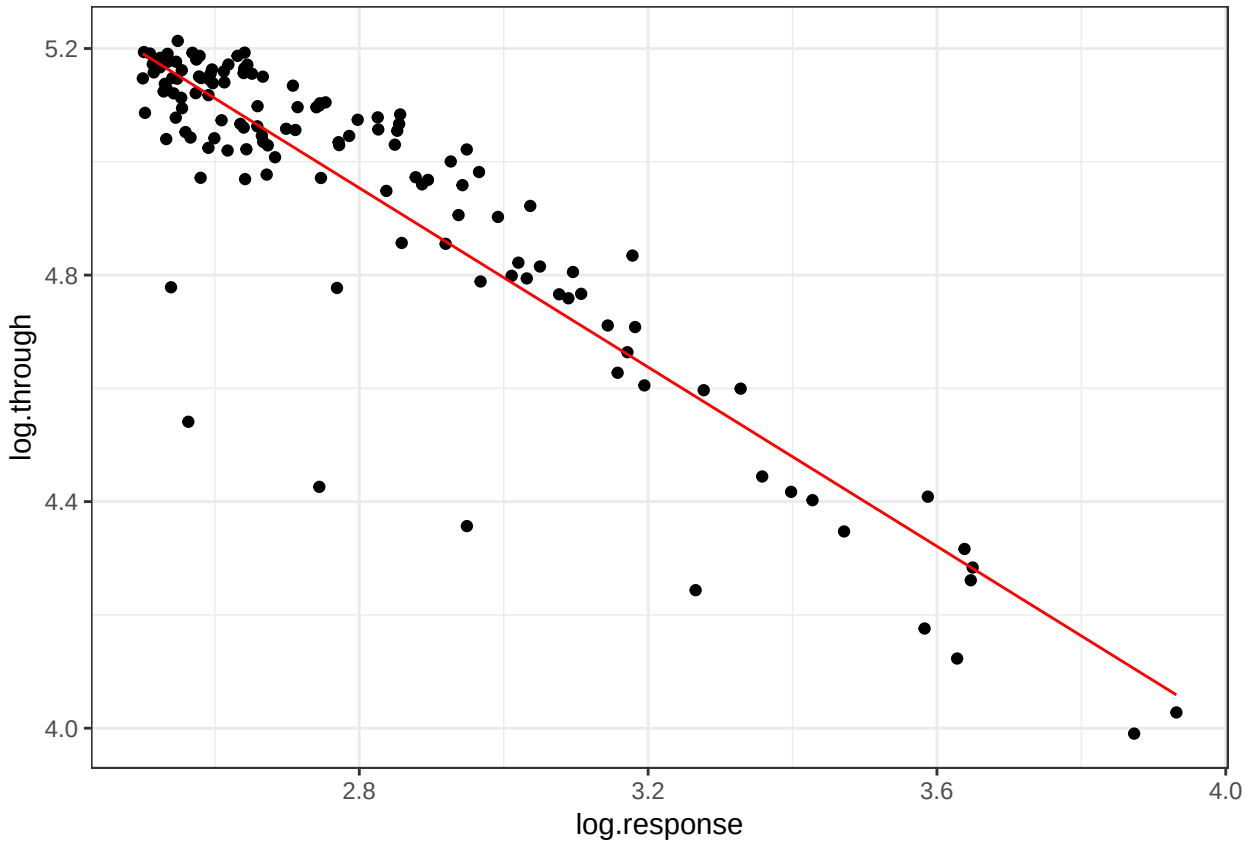
Graph of throughput and response time (log-log)

```
#plot model of both data sets vs eachother
ggplot(tcdat, aes(y=log10(throughput), x =log10(average_response_time))) + geom_point()+
  geom_point(size =0.5) +
  theme_bw()
```



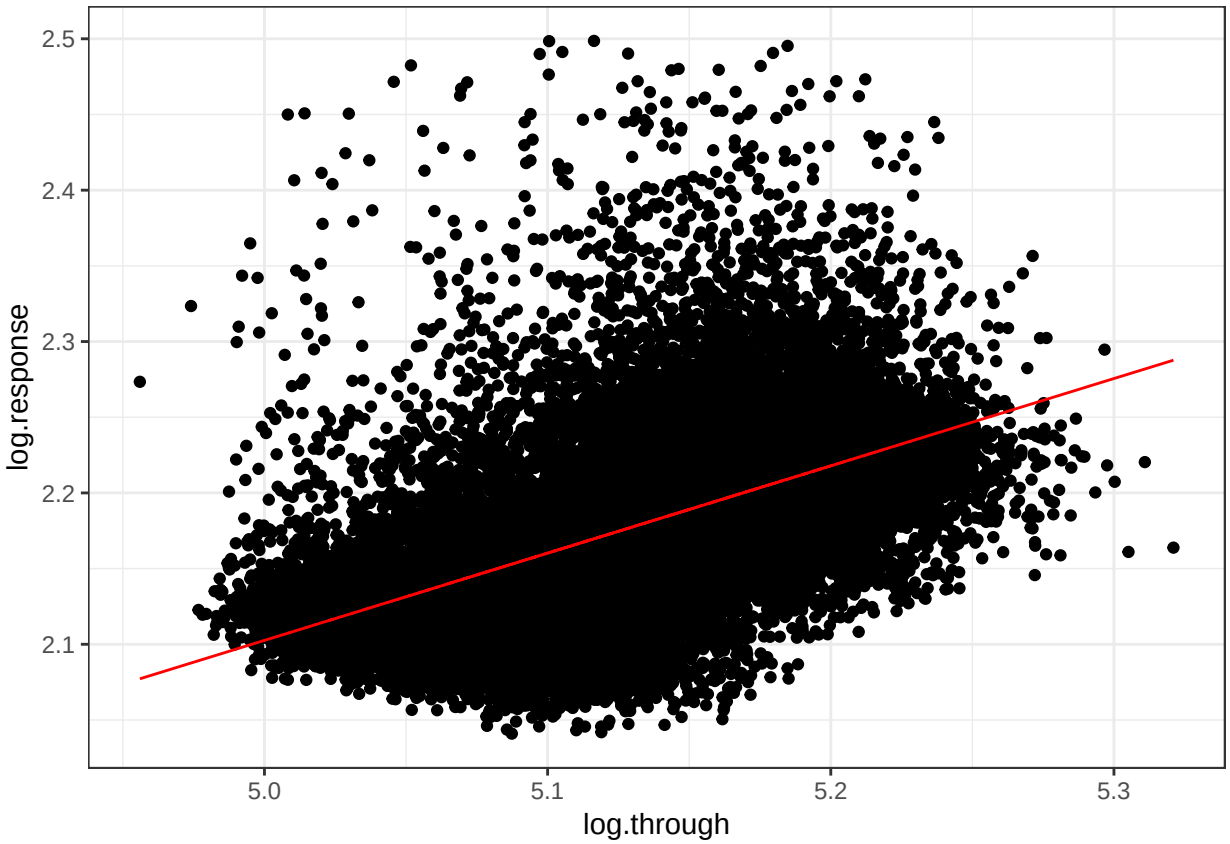
Process 1: extremely long response times bottleneck throughput (log-log)

```
#take log of both throughput and response time
tcdat %<>% mutate(log.response=log10(average_response_time),log.through=log10(throughput))
#create subset of data that has response time effect
tcdat.res<-tcdat %>% filter(log.response>2.5)
# estimate impact response has on throughput
res.impact.throughput.mod <-lm(log.through~ log.response,data=tcdat.res)
#plot model of responses impact on throughput
tcdat.res$predict.throughput<-predict(res.impact.throughput.mod,log.response=tcdat.res$log.response,data
ggplot(tcdat.res, aes(y=log.through, x =log.response)) + geom_point()+
  geom_point(size =0.5) +
  geom_line(color='red',aes(y=tcdat.res$predict.throughput,x=tcdat.res$log.response))+
  theme_bw()
```



Process 2: Increased throughput can slow down response time (log-log)

```
#create subset of data around throughput impact of throughput on response time
# (the more traffic the slower the response time)
tcdat.thr<-tcdat %>% filter(log.response<2.5 & log.through > 4.95)
# estimate impact response has on throughput
response.throughput.mod <-lm(log.response~ log.through,data=tcdat.thr)
#plot model of responses impact on throughput
tcdat.thr$predict.response<-predict(response.throughput.mod,log.through=tcdat.thr$log.through,data=tcdat
ggplot(tcdat.thr, aes(x=log.through, y =log.response)) + geom_point()+
  geom_point(size =0.5) +
  geom_line(color='red',aes(y=tcdat.thr$predict.response,x=tcdat.thr$log.through))+
  theme_bw()
```



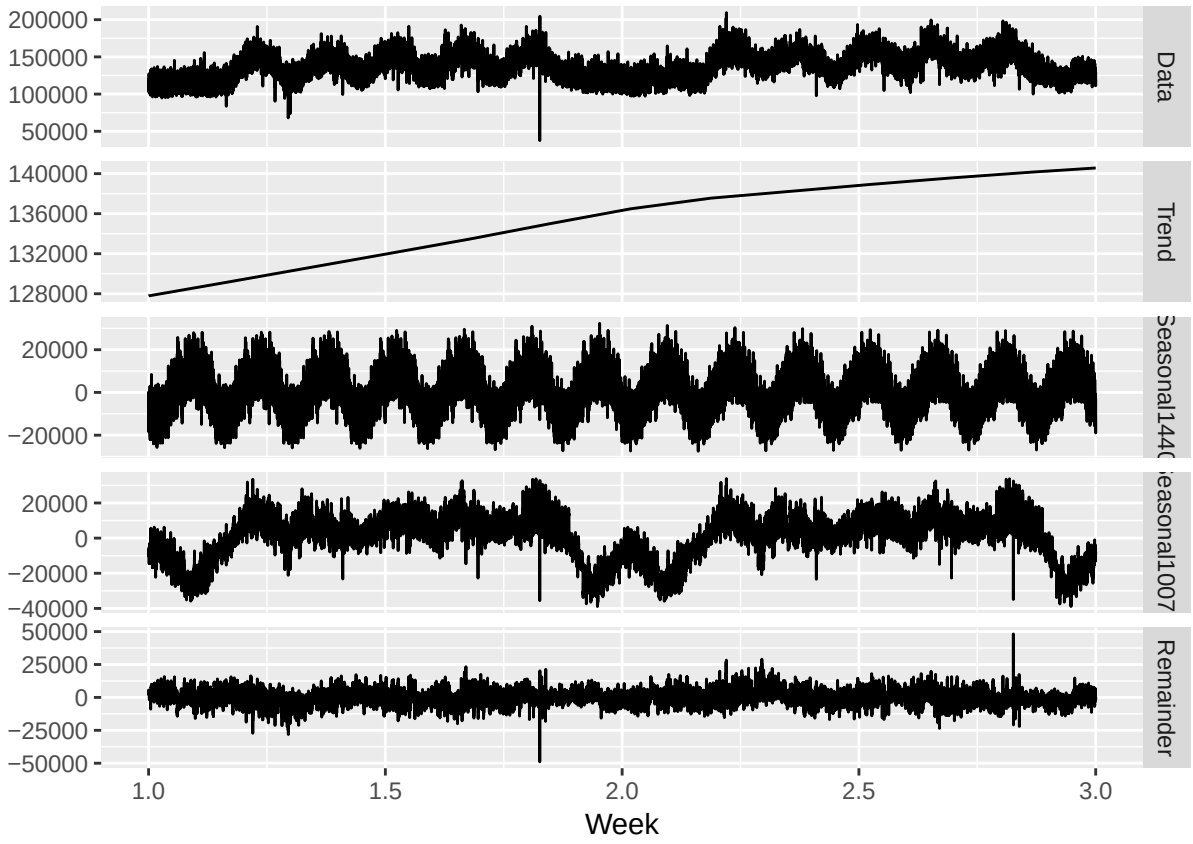
4. Decompose timeseries and account for any remaining structured noise

account for structured noise in forecasting model by subtracting out throughput spikes caused by extremely slow response times

```
# #1. put an NA in throughput outlier/spikes
tcdat %<>% mutate(throughput=case_when(log.response > 2.5  ~ NA_real_ ,TRUE ~ throughput))
# #2. smooth out these by interpolation
tcdat$throughput<-na.interpolation(tcdat$throughput, option = "linear")
```

Decompose Timeseries into a weekly period, a daily period, trend and random noise

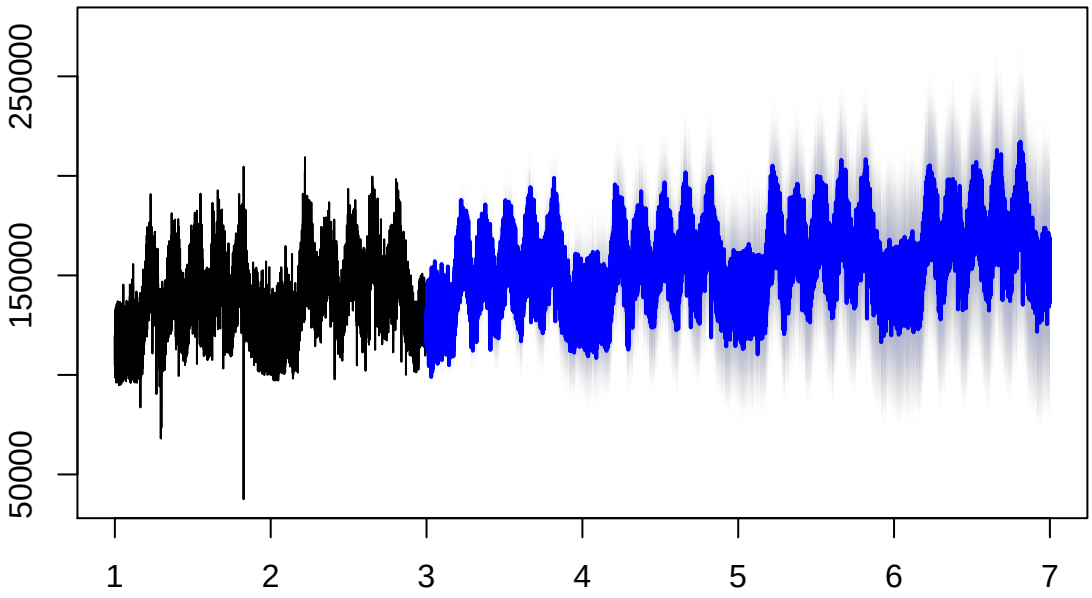
```
tcdat.ts <- forecast::msts(tcdat$throughput,seasonal.periods = c(24*60,24*60*7-1))
tcdat.ts %>% mstl(.) ->tcdat.mstl
tcdat.mstl %>% autoplot() + xlab("Week")
```



5. Forecast timeseries

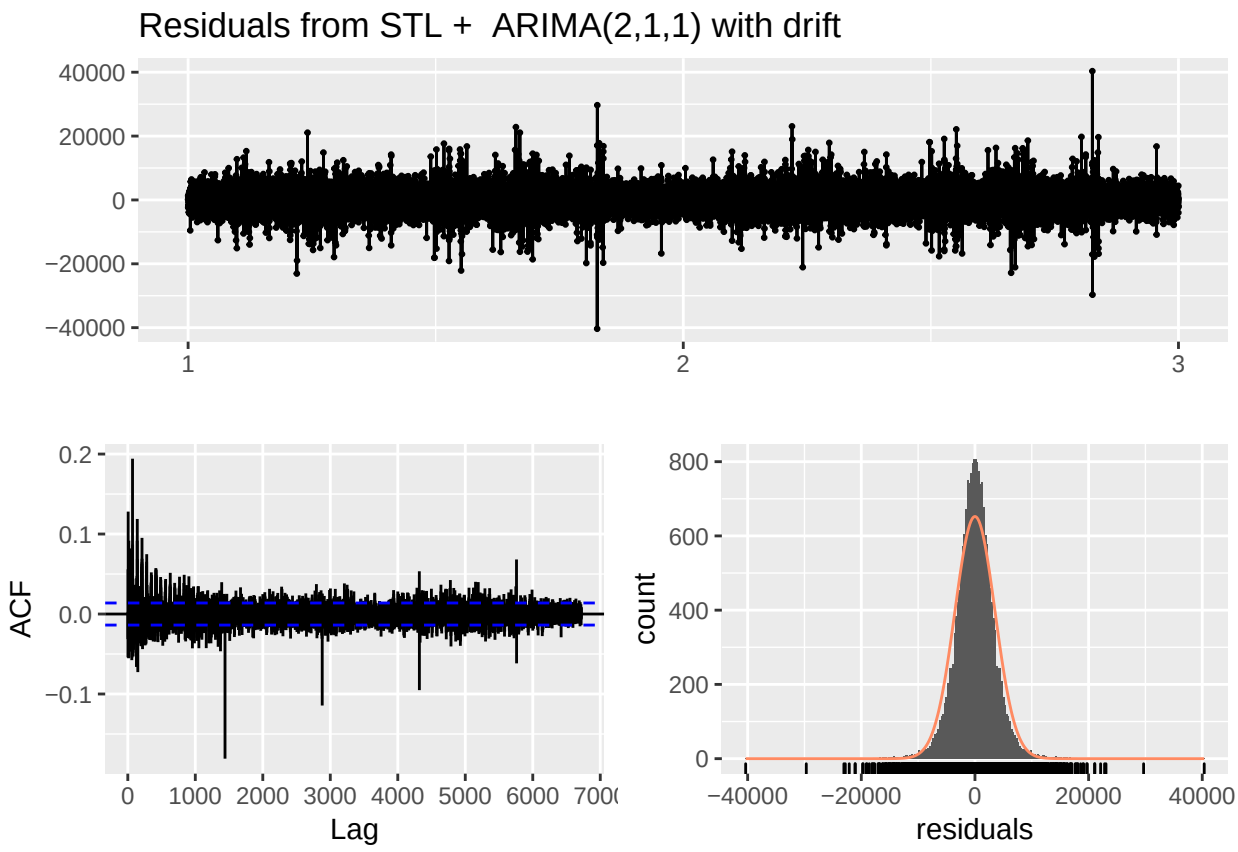
Initial forecast of timeseries using STL decomposition and ARIMA for trend and noise component

```
# STL: local regression for decomposition of cyclic components
# non seasonal arima model for trend and noise component
# 2 autoregressive terms (p)
# 1 difference needed for stationarity (d)
# 1 lagged forecast errors terms (q)

#create model that describes  4 weeks of data
tcdat.ts %>%stlf(method='arima',level=c(80,95),h =24*60*7*4) ->ts.model
#use model to forecast out 4  weeks
the.forecast<-forecast(ts.model,robust=TRUE)
plot(the.forecast)
```




```
checkresiduals(ts.model)
```

```
## Warning in checkresiduals(ts.model): The fitted degrees of freedom is based
## on the model used for the seasonally adjusted data.
```




```
##
```

```
##  Ljung-Box test
##
## data:  Residuals from STL +  ARIMA(2,1,1) with drift
## Q* = 21904, df = 4028, p-value < 2.2e-16
##
## Model df: 4.   Total lags used: 4032
```

create forecast dataframe

```
tcdat %>% mutate(begin_time2=begin_time+minutes(1)) %>%
filter(max(begin_time2)==begin_time2) %>% select(begin_time2) %>%
.$begin_time %>% as.character()->start.new.time.vec

forecast.date<-seq(from=ymd_hms(start.new.time.vec)+min(1),length.out=the.forecast$mean %>% length,by="
the.forecast %>% data.frame() %>% cbind(forecast.date) -> the.forecast
the.forecast %>% head
```

	Point.Forecast	Lo.80	Hi.80	Lo.95	Hi.95	forecast.date
3.000198	132488.8	127951.3	137026.2	125549.3	139428.2	2017-10-28 17:00:01
3.000298	122422.9	117457.7	127388.2	114829.2	130016.6	2017-10-28 17:01:01
3.000397	125373.1	119887.3	130858.9	116983.3	133762.9	2017-10-28 17:02:01
3.000496	115980.8	110241.6	121719.9	107203.5	124758.0	2017-10-28 17:03:01
3.000595	127887.7	121954.7	133820.8	118813.9	136961.5	2017-10-28 17:04:01
3.000695	125441.0	119382.8	131499.1	116175.8	134706.1	2017-10-28 17:05:01

6. Adjust for user behavior

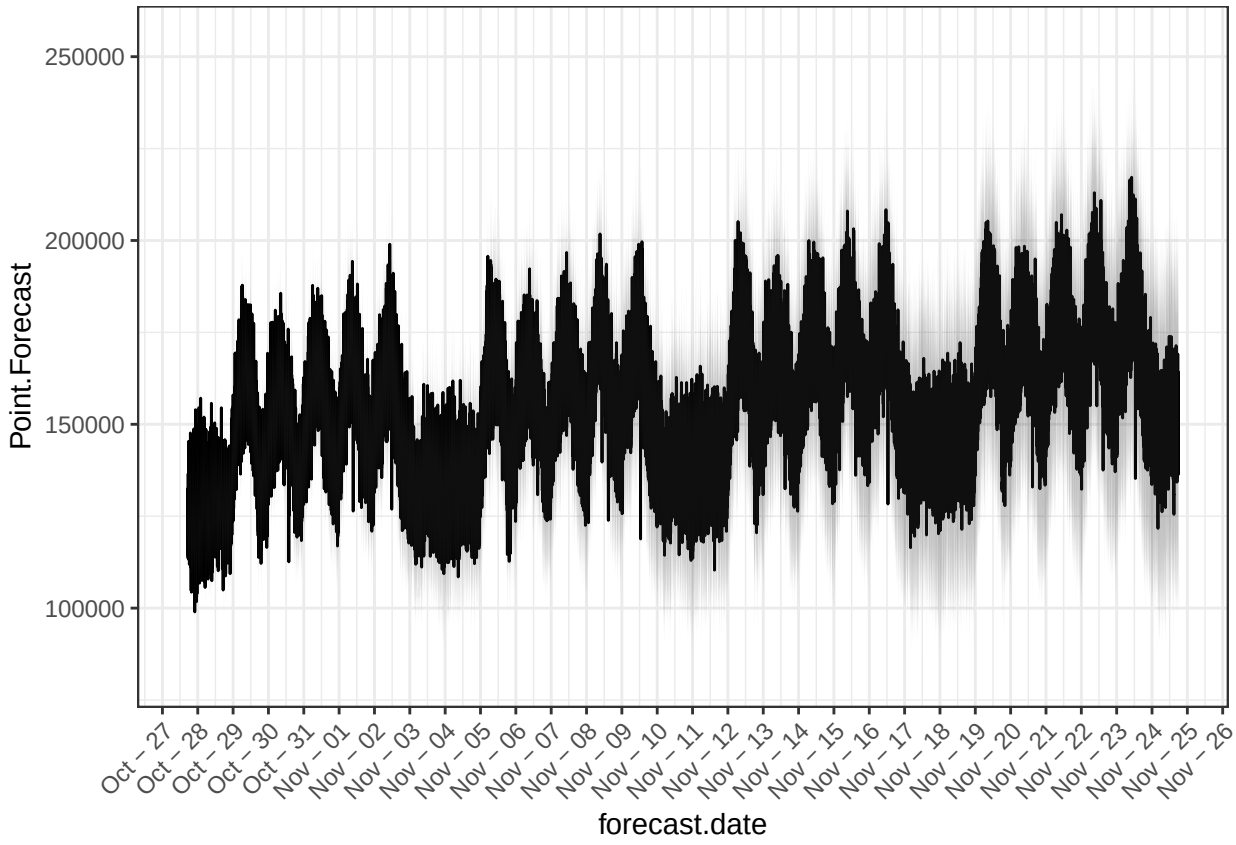
adjust for end of daylight savings time (app usage should be shifted)

```
# Sunday, November 5, 2017, 2:00:00 am clocks were turned backward 1 hour to 1:00am (US Canada)
# UTC does not change with a change of seasons
# but usage pattern of app will shift according to local time change

# for example: post time change, the 8am UTC usage will look like previous 7am UTC usage during dayligh
# during daylight savings people were getting to work an hour earlier in UTC time

# ID daylight savings observations and adjust throughput predictions (fall back)
the.forecast %<>% mutate(dst=case_when(forecast.date<ymd_hms("2017-11-05 01:00:00")~"yes",TRUE ~ "no"))
the.forecast %>% filter(dst=="no" & month(forecast.date)==11 & day(forecast.date)==5 & hour(forecast.da
the.forecast %>% filter(dst=="no") %>% mutate(forecast.date=ymd_hms(forecast.date,tz = "UTC")+60*60)->t
the.forecast.adj<-rbind(the.forecast %>% filter(dst=="yes"),fall.back.hr,the.forecast.dst.shift)
```

plot adjusted forecast

```
#the.forecast.adj %>% mutate(wkd=weekdays.POSIXt(forecast.date)) %>% filter(!wkd %in% #c("Saturday","Su

the.forecast.adj %>% mutate(wkd=weekdays.POSIXt(forecast.date))->the.forecast.adj.plot
```

```
p1 <- ggplot(the.forecast.adj.plot, aes(forecast.date,Point.Forecast))+
    geom_line()+geom_ribbon(data=the.forecast.adj,aes(ymin=Lo.80,ymax=Hi.80),alpha=0.3)+
  scale_x_datetime(breaks = date_breaks("1 day"), labels=date_format("%b - %d",tz=Sys.timezone()))+
  theme_bw()+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
p1
```



7. Add back in randomized structured noise

account for process 2

```
#use forecasted throughput timeline to forecast response time with fitted function describing process 1
the.forecast.adj$resp_p2_component_log<-predict(response.throughput.mod,data.frame(log.through=log10(the
the.forecast.adj %>% mutate(resp_p2_component=10^resp_p2_component_log) %>% select(-resp_p2_component_l
mutate(resp_p2_component=resp_p2_component+sample(10^(response.throughput.mod$residuals),size=length(the
```

account for process 1

```
set.seed(60)
# add back in process 1 outlier spikes

# 1.randomize response spike events locally to days (phenomena day/range specific)
```

```
#function to reshuffle spike distribution in a weekday specific manner
reshuffle.spikes<-function(shift.sec){tcdat %>% filter(log.response>2.5) %>% mutate(wkd=weekdays(begin_
  mutate(average_response_time=sample(average_response_time,size=length(average_response_time),replace=
    mutate(begin_time=begin_time+shift.sec)}

#events randomized
response.spikes.randomized<-rbind(reshuffle.spikes(0),reshuffle.spikes(60*60*24*7*2)) %>% select(begin_
  rename(average_response_time_spike=average_response_time,forecast.date=begin_time)
```

```
## Adding missing grouping variables: `wkd`
```

```
# 2. merge spike "events" into forecasted data frame
merge(response.spikes.randomized,the.forecast.adj,by="forecast.date",all=T) %>% arrange(forecast.date) %
```

forecast.date	wkd	average_response_time_spike	Point.Forecast	Lo.80	Hi.80	Lo.95	Hi.95	dst	re
2017-10-16 16:14:00	Monday	338.0993	NA	NA	NA	NA	NA	NA	
2017-10-16 16:15:00	Monday	1844.1379	NA	NA	NA	NA	NA	NA	
2017-10-16 16:16:00	Monday	410.3197	NA	NA	NA	NA	NA	NA	
2017-10-16 16:17:00	Monday	410.3197	NA	NA	NA	NA	NA	NA	
2017-10-16 16:18:00	Monday	924.3053	NA	NA	NA	NA	NA	NA	
2017-10-16 16:19:00	Monday	755.1039	NA	NA	NA	NA	NA	NA	