COVID19 Prediction

Ahmad Abboud

5/31/2020

Introduction

The White House Office of Science and Technology Policy (OSTP) pulled together coalition research groups and companies (including Kaggle) to prepare the COVID-19 Open Research Dataset (CORD-19) to attempt to address key open scientific questions on COVID-19. Those questions are drawn from the National Academies of Sciences, Engineering, and Medicine's (NASEM) and the World Health Organization (WHO). This document uses the Kaggle challenge dataset to predict the daily number of confirmed COVID19 cases in various locations across the world, as well as the number of resulting fatalities. To tackle the problem, we analyze a set of regression models that seems to work with this kind of data. For instance, we fit auto ARIMA, Gradient Boosting Machine, Generalized Linear Regression, Random Forest, and Stack Ensemble models. Finally, we chose Gradient Boosting Machine as the best regression model for our data. The evaluation was run based on the Pinball evaluation method which was suggested by the challenge managers. The results obtained are modest, but further optimization could be done by integrating more useful predictor

Installing required Libraries

```
# Install required Libraries if necessary
list.of.packages <- c("caret", "dplyr","Boruta","mlbench",
                      "tidyr","fUnitRoots","FitAR","forecast",
                      "stringr","Metrics","tictoc","MLmetrics","h2o","opera","urca")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)
```

Load Required Libraries

```
#Load required libraries
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library("fUnitRoots")
```

```
## Loading required package: timeDate
```

```
## Loading required package: timeSeries
```

```
##
```

```
## Attaching package: 'timeSeries'

## The following object is masked from 'package:zoo':
##
##     time<-

## Loading required package: fBasics
```

```
library(FitAR)
```

```
## Loading required package: lattice

## Loading required package: leaps

## Loading required package: ltsa

## Loading required package: bestglm
```

```
library("forecast")
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

##
## Attaching package: 'forecast'

## The following object is masked from 'package:FitAR':
##
##     BoxCox
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
library(dplyr)
```

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

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

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

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

```
library(Boruta)
```

```
## Loading required package: ranger
```

```
library(mlbench)
library(tidyr)
library(stringr)
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
## The following object is masked from 'package:forecast':
##
##     accuracy
library(opera)
library(urca)
##
## Attaching package: 'urca'
## The following objects are masked from 'package:fUnitRoots':
##
##     punitroot, qunitroot, unitrootTable
library(tictoc)
library(rlist)
library(MLmetrics)
##
## Attaching package: 'MLmetrics'
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
## The following object is masked from 'package:base':
##
##     Recall
library(doParallel)
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
library(h2o)
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------
##
## Attaching package: 'h2o'
## The following objects are masked from 'package:timeSeries':
```

```
##
##     apply, colnames, colnames<-
```

```
## The following object is masked from 'package:timeDate':
##
##     dayOfWeek
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

Data Preparation

Import Dataset

```
train_KagCsv <- read.csv("Datasets/Week5/train.csv", na.strings = c("", "NA"))
train_Kag<- as.data.frame(train_KagCsv)
train_Kag <- train_Kag %>% mutate(Date=as.Date(Date))

test_KagCsv <- read.csv("Datasets/Week5/test.csv", na.strings = c("", "NA"))
test_Kag <- as.data.frame(test_KagCsv)
test_Kag <- test_Kag %>% mutate(Date=as.Date(Date))

sub_KagCsv <- read.csv("Datasets/Week5/submission.csv", na.strings = c("", "NA"))
sub_Kag <- as.data.frame(sub_KagCsv)
```

Data Cleaning

```
# Clean Data
print(paste("number of targetValue <0 :",sum(train_Kag$TargetValue < 0)))
```

```
## [1] "number of targetValue <0 : 3000"
```

```
train_Kag$TargetValue <- ifelse(train_Kag$TargetValue<0,abs(train_Kag$TargetValue),train_Kag$TargetValu
print(sum(train_Kag$TargetValue < 0))
```

```
## [1] 0
```

```
#Add lifting to the to all Target Values to insure it is away from zero and away from constant
lift<-10
train_Kag$TargetValue <-train_Kag$TargetValue +  rnorm(length(train_Kag$TargetValue),mean=lift,sd=1)

train_Kag_clean <- train_Kag %>%
  mutate(CPR=factor(paste(County,Province_State ,Country_Region,sep = "_"))) %>%
  mutate(days = as.numeric(Date-min(Date)), Wday=factor(weekdays(Date)),month=factor(as.numeric(format(
levels(train_Kag_clean$Wday) <-c(6,2,7,1,5,3,4) # Start from sunday =1

train_Kag_clean%>% head(10)%>% knitr::kable()
```

Id	County	Province_State	Country_Region	Population	Weight	Date	Target	TargetValue
1	NA	NA	Afghanistan	27657145	0.0583587	2020-01-23	ConfirmedCases	10.358999
2	NA	NA	Afghanistan	27657145	0.5835874	2020-01-23	Fatalities	9.800538
3	NA	NA	Afghanistan	27657145	0.0583587	2020-01-24	ConfirmedCases	11.018083
4	NA	NA	Afghanistan	27657145	0.5835874	2020-01-24	Fatalities	9.525160
5	NA	NA	Afghanistan	27657145	0.0583587	2020-01-25	ConfirmedCases	10.806071
6	NA	NA	Afghanistan	27657145	0.5835874	2020-01-25	Fatalities	10.583564
7	NA	NA	Afghanistan	27657145	0.0583587	2020-01-26	ConfirmedCases	10.638240
8	NA	NA	Afghanistan	27657145	0.5835874	2020-01-26	Fatalities	10.220815
9	NA	NA	Afghanistan	27657145	0.0583587	2020-01-27	ConfirmedCases	9.858977
10	NA	NA	Afghanistan	27657145	0.5835874	2020-01-27	Fatalities	7.758841

```
test_Kag_clean <- test_Kag %>%
  mutate(CPR=factor(paste(County,Province_State ,Country_Region,sep = "_"))) %>%
  mutate(days = as.numeric(Date-min(Date)), Wday=factor(weekdays(Date)),month=factor(as.numeric(format(
levels(test_Kag_clean$Wday) <-c(6,2,7,1,5,3,4) # Start from sund =1

test_Kag_clean%>% head() %>% knitr::kable()
```

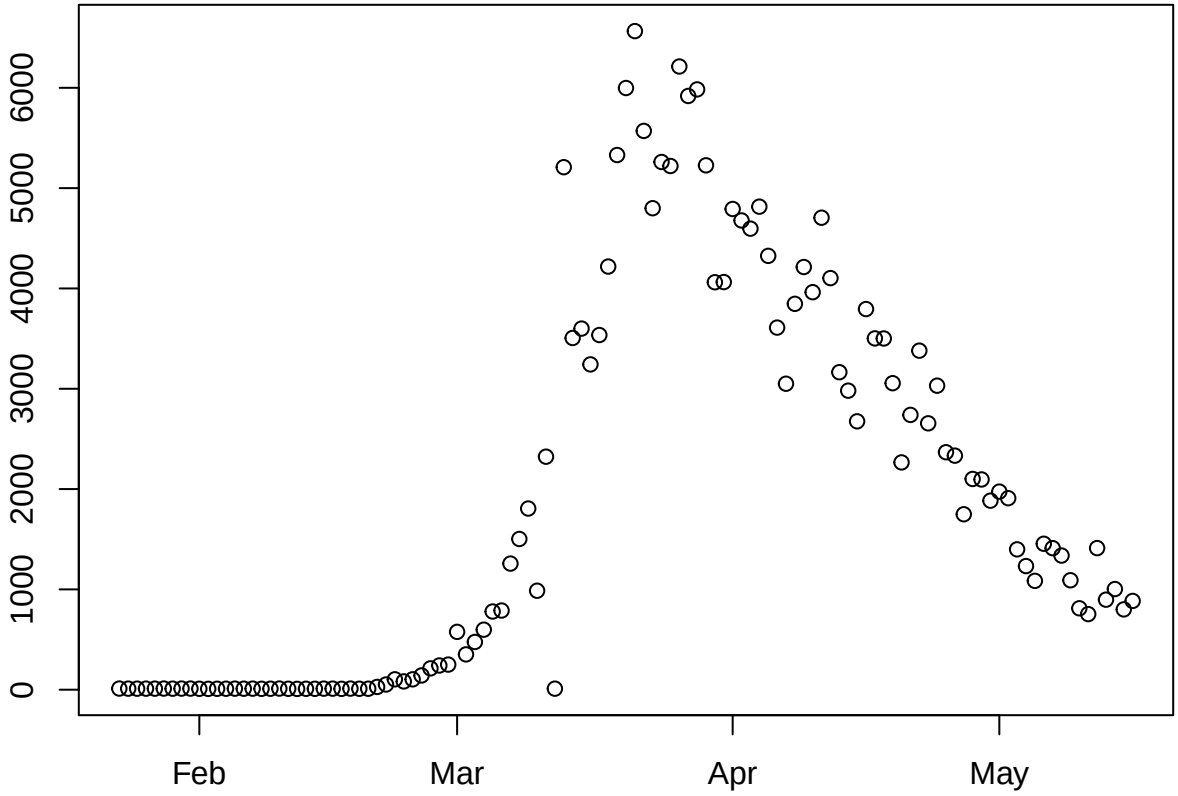
ForecastId	County	Province_State	Country_Region	Population	Weight	Date	Target
1	NA	NA	Afghanistan	27657145	0.0583587	2020-04-27	ConfirmedCase
2	NA	NA	Afghanistan	27657145	0.5835874	2020-04-27	Fatalities
3	NA	NA	Afghanistan	27657145	0.0583587	2020-04-28	ConfirmedCase
4	NA	NA	Afghanistan	27657145	0.5835874	2020-04-28	Fatalities
5	NA	NA	Afghanistan	27657145	0.0583587	2020-04-29	ConfirmedCase
6	NA	NA	Afghanistan	27657145	0.5835874	2020-04-29	Fatalities

Explorator y Data An alysis Plot confirme d cases f or Italy

```
#
Conf_US <- train_Kag_clean %>% dplyr::filter(Target=="ConfirmedCases" & CPR=="NA_NA_Italy") %>%
  group_by(CPR,Date) %>% summarise(TargetValue=sum(TargetValue)) %>%
  select(Date,TargetValue)
```

```
## Adding missing grouping variables: `CPR`
```

```
par(mar = c(2, 2, 2, 2))
plot(Conf_US$Date,Conf_US$TargetValue)
```

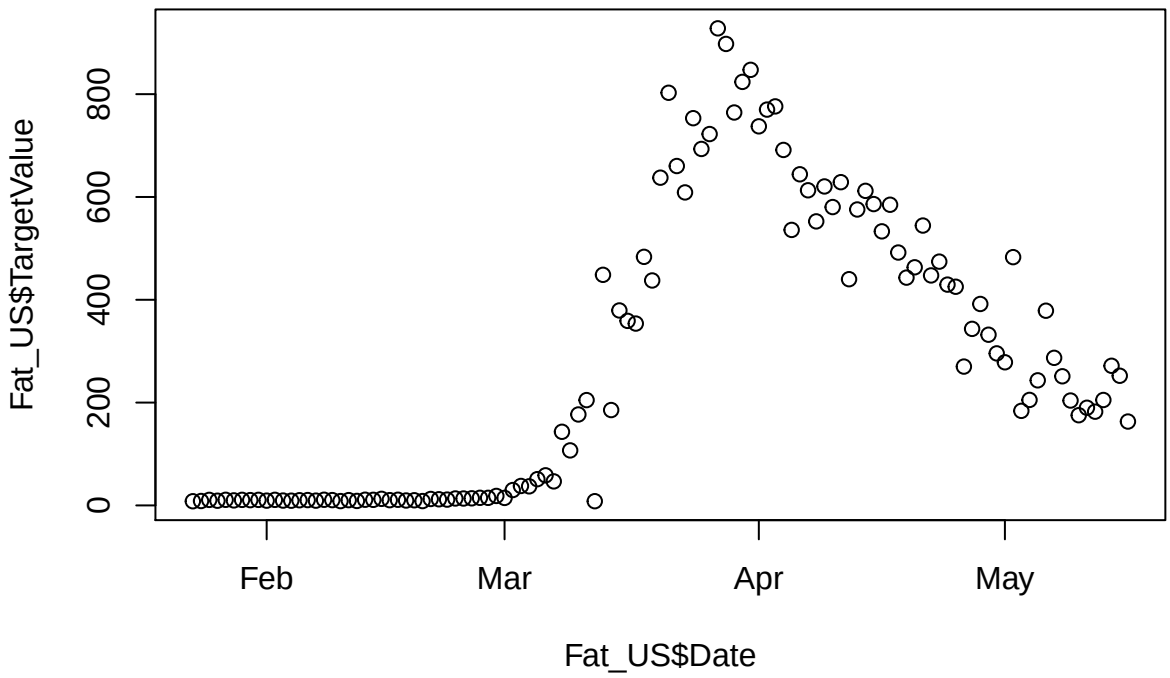


Plot Fat cases for Italy

```
Fat_US<-train_Kag_clean %>% dplyr::filter(Target=="Fatalities" & CPR=="NA_NA_Italy") %>%
  group_by(CPR,Date) %>% summarise(TargetValue=sum(TargetValue)) %>%
  select(Date,TargetValue)
```

```
## Adding missing grouping variables: `CPR`
```

```
plot(Fat_US$Date,Fat_US$TargetValue)
```



Split Confirmed and Fatality cases and add accumulated sum

```
# Split Confirmed and Fatality cases and add accumulated sum
# Split Kaggle train Data for local train test
ForecastDate <- as.Date("2020-05-08") # one weeks forecast
StartDate <- as.Date("2020-03-01") # since there is a tall tail at the begining for most counties
indx <- train_Kag_clean$Date < ForecastDate & train_Kag_clean$Date>StartDate
ts_indx <-train_Kag_clean$Date > ForecastDate
train <- train_Kag_clean[indx,]
test <- train_Kag_clean[ts_indx,]

train_Conf <- train %>% dplyr::filter(Target=="ConfirmedCases") %>%
  group_by(CPR) %>% mutate(accSum=cumsum(TargetValue))%>% ungroup()

train_Fat <- train %>% dplyr::filter(Target=="Fatalities") %>%
  group_by(CPR) %>% mutate(accSum=cumsum(TargetValue))%>% ungroup()

test_Conf <- test%>% dplyr::filter(Target=="ConfirmedCases") %>%
  group_by(CPR) %>% mutate(accSum=cumsum(TargetValue))%>% ungroup()

test_Fat <- test %>% dplyr::filter(Target=="Fatalities") %>%
  group_by(CPR) %>% mutate(accSum=cumsum(TargetValue))%>% ungroup()
```

Lets see top 10 Confirmed cases county

```
# Lets see top 10 Confirmed cases county
top10 <- train_Conf %>% group_by(CPR) %>% summarise(TargetValue=sum(TargetValue)) %>%
```

```
  arrange(desc(TargetValue)) %>% dplyr::filter(str_detect(CPR,"_US\\b"))%>% head(10)
top10%>% knitr::kable()
```

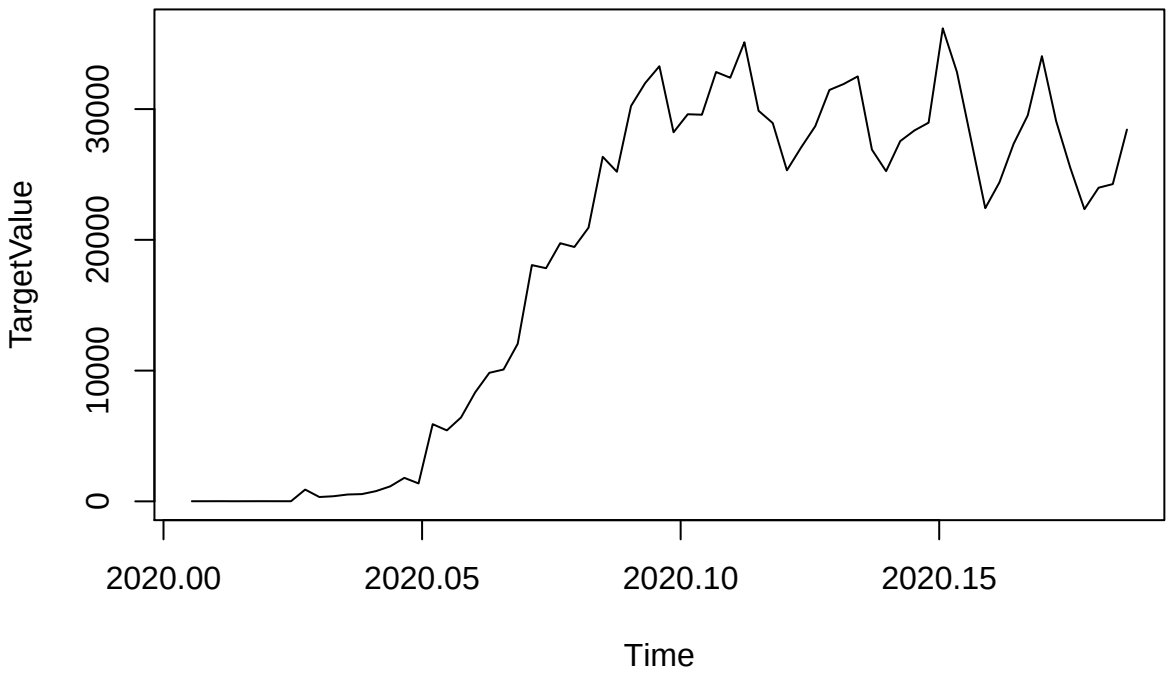
CPR	TargetValue
NA_NA_US	1257525.23
NA_New York_US	328144.21
New York_New York_US	180879.38
NA_New Jersey_US	134663.97
NA_Massachusetts_US	74388.02
NA_Illinois_US	71536.44
NA_California_US	62827.18
NA_Pennsylvania_US	56634.51
Cook_Illinois_US	49023.80
NA_Michigan_US	46424.03

Lets View Bottom 10 with at least 100 Target Value

```
# Lets View Bottom 10 with at least 100 Target Value
bot10 <- train_Conf%>% dplyr::filter(TargetValue > 100)  %>% group_by(CPR) %>% summarise(TargetValue=su
  arrange(TargetValue)
bot10%>%head(10) %>%knitr::kable()
```

CPR	TargetValue
Mecklenburg_North Carolina_US	100.1854
Jefferson_Colorado_US	100.5263
NA_NA_Iraq	100.5893
NA_NA_Bosnia and Herzegovina	101.1854
Lancaster_Pennsylvania_US	101.3820
Monroe_New York_US	102.0161
Hillsborough_Florida_US	102.5598
NA_NA_Congo (Kinshasa)	102.9801
Harrisonburg_Virginia_US	103.5942
Iberville_Louisiana_US	103.6552
Lets consider USA as case Study	

```
#######################################
# Case study US Confirmed cases

train_Conf_USA <- train_Conf %>% dplyr::filter(CPR=="NA_NA_US")
#View(train_Conf_USA%>%head(10))

test_Conf_USA <- test_Conf %>% dplyr::filter(CPR=="NA_NA_US")
#View(train_Conf_USA)

train_Fat_USA <- train_Fat %>% dplyr::filter(CPR=="NA_NA_US")
test_Fat_USA <- test_Fat %>% dplyr::filter(CPR=="NA_NA_US")
# Convert to time series
tsTrConf<- ts(select(data.frame(train_Conf_USA),TargetValue),start=c(2020,3,1),frequency = 365)

plot(tsTrConf)
```



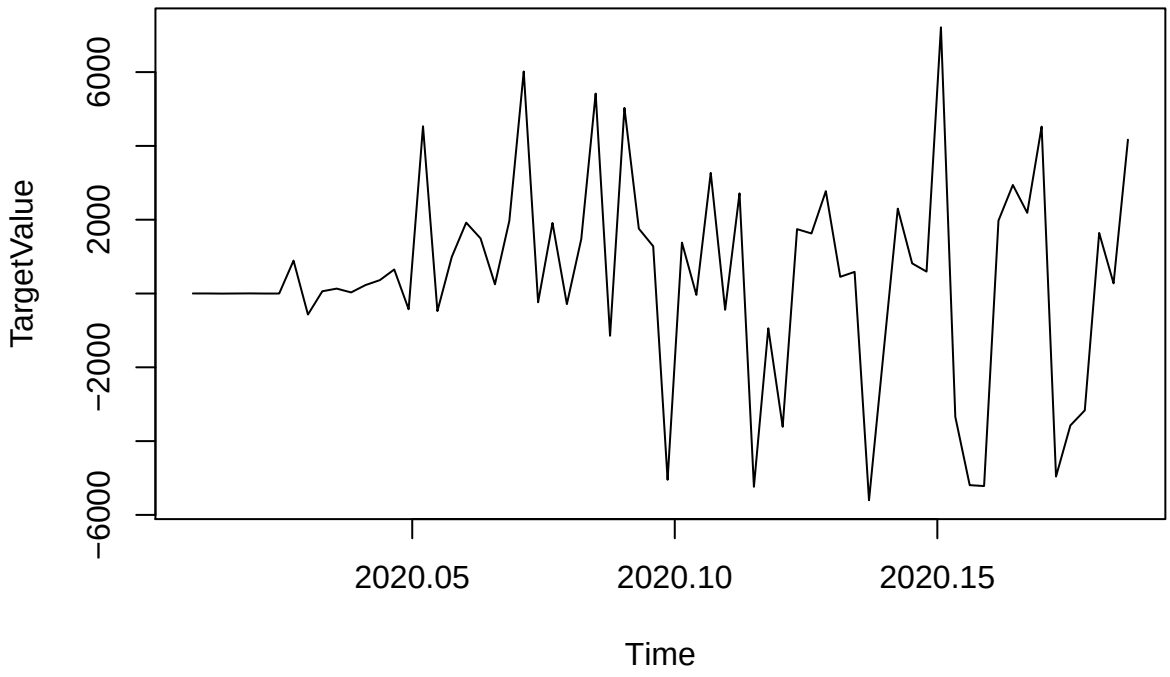
Its look like the data could be modeled as time series. Now detemine stationarity of data by urkpss Test

```
#detemine stationarity of data by urkpss Test

my_urkpssTest <- function (x, type, lags, use.lag, doplot) {
  x <- as.vector(x)
  urca <- urca::ur.kpss(x, type = type[1], lags = lags[1], use.lag = use.lag)
  output = capture.output(urca::summary(urca))[-(1:4)]
  output = output[-length(output)]
  for (i in 1:length(output)) output[i] = paste(" ", output[i])
  ans = list(name = "ur.kpss", test = urca, output = output)
  if (doplot)
    #plot(urca)
  new("fHTEST", call = match.call(), data = list(x = x),
      test = ans, title = "KPSS Unit Root Test", description = description())
}

my_urkpssTest(tsTrConf, type = c("tau"), lags = c("long"),
              use.lag = NULL, doplot = TRUE)
```

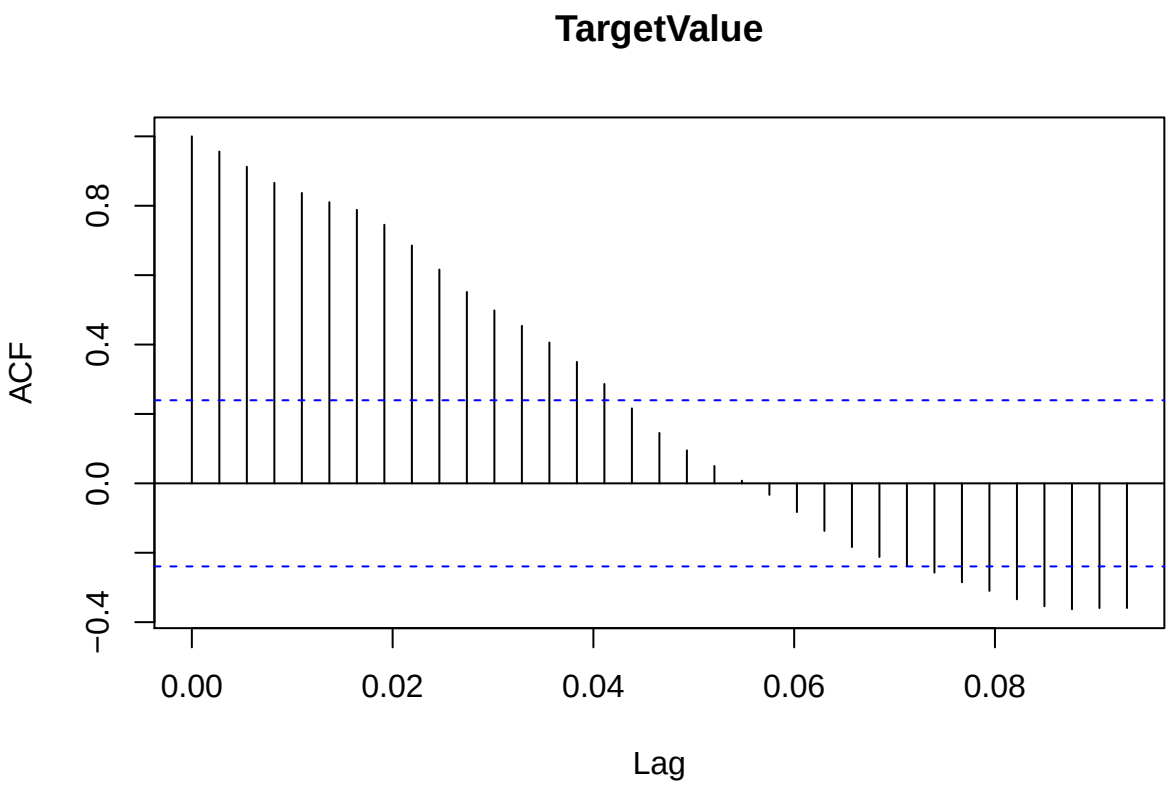
```
##
## Title:
##  KPSS Unit Root Test
##
## Test Results:
##
##   Test is of type: tau with 10 lags.
```

```
##
##   Value of test-statistic is: 0.1574
##
##   Critical value for a significance level of:
##                   10pct  5pct 2.5pct  1pct
##   critical values 0.119 0.146  0.176 0.216
##
## Description:
##  Sun May 31 20:56:20 2020 by user: AHB
```

```r
# From the results it is clear that "Value of test-statistic" > "Critical value for a significance leve
# Thus the null hypothesis of stationarity is rejected.

tsstationary<-diff(tsTrConf, differences=1)
plot(tsstationary)
```




```r
acf(tsTrConf,lag.max=34)
```




```
######
```

Feature Analysis

We Can Use Boruta library to rank features

```
## Feature Selection Analysis
train_Con_dt<-select(train_Conf_USA,TargetValue,Date,days,Wday)
train_Fat_dt<-select(train_Fat_USA,TargetValue,Date,days,Wday)
# We Can Use Boruta library to rank features
set.seed(2020)
Br_Con <- Boruta(TargetValue ~., data=train_Con_dt, doTrace=2, maxRuns=50)
```

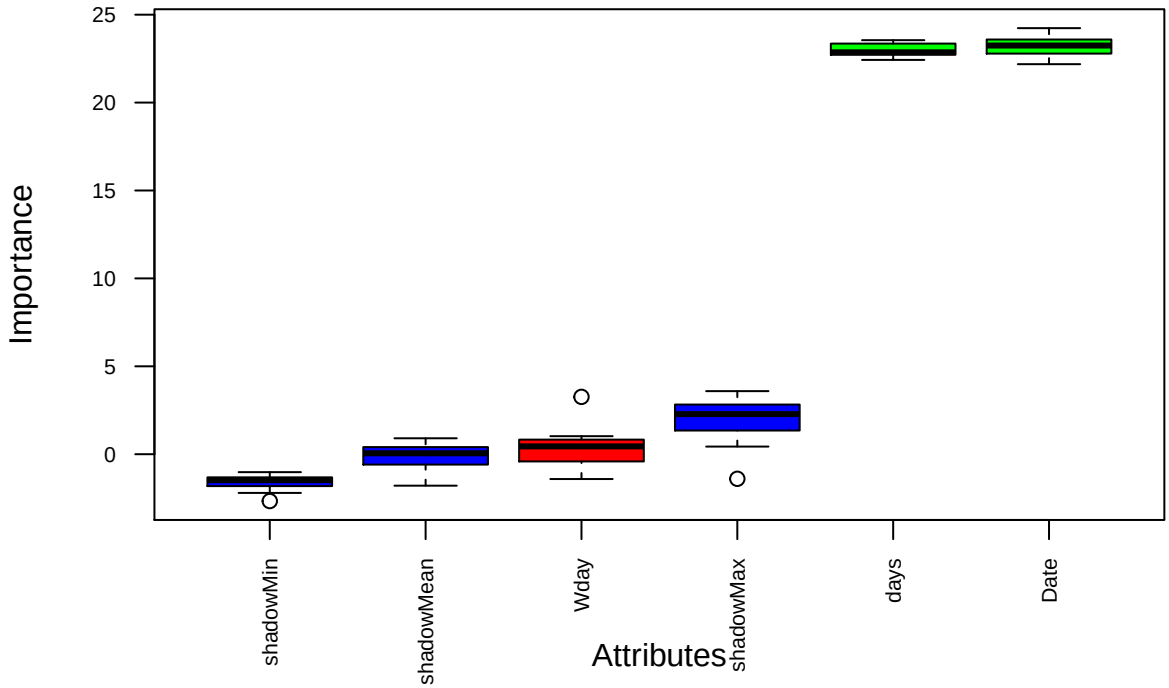
```
##  1. run of importance source...
##  2. run of importance source...
##  3. run of importance source...
##  4. run of importance source...
##  5. run of importance source...
##  6. run of importance source...
##  7. run of importance source...
##  8. run of importance source...
##  9. run of importance source...
## After 9 iterations, +0.44 secs:
```

```
##  confirmed 2 attributes: Date, days;
##  still have 1 attribute left.
##  10. run of importance source...
##  11. run of importance source...
##  12. run of importance source...
## After 12 iterations, +0.63 secs:
##  rejected 1 attribute: Wday;
##  no more attributes left.
```

```
print(Br_Con)
```

```
## Boruta performed 12 iterations in 0.6296129 secs.
##  2 attributes confirmed important: Date, days;
##  1 attributes confirmed unimportant: Wday;
```

```
plot(Br_Con,las=2,cex.axis=0.7) # las: to make lables vertical, cex.axis for scaling view
```



We can see that only the Day and days features are important for estimating TargetValue in Confirmed Cases Now lets explore Fatality cases

```
set.seed(2020)
Br_Fat <- Boruta(TargetValue ~., data=train_Fat_dt, doTrace=2, maxRuns=50)
```

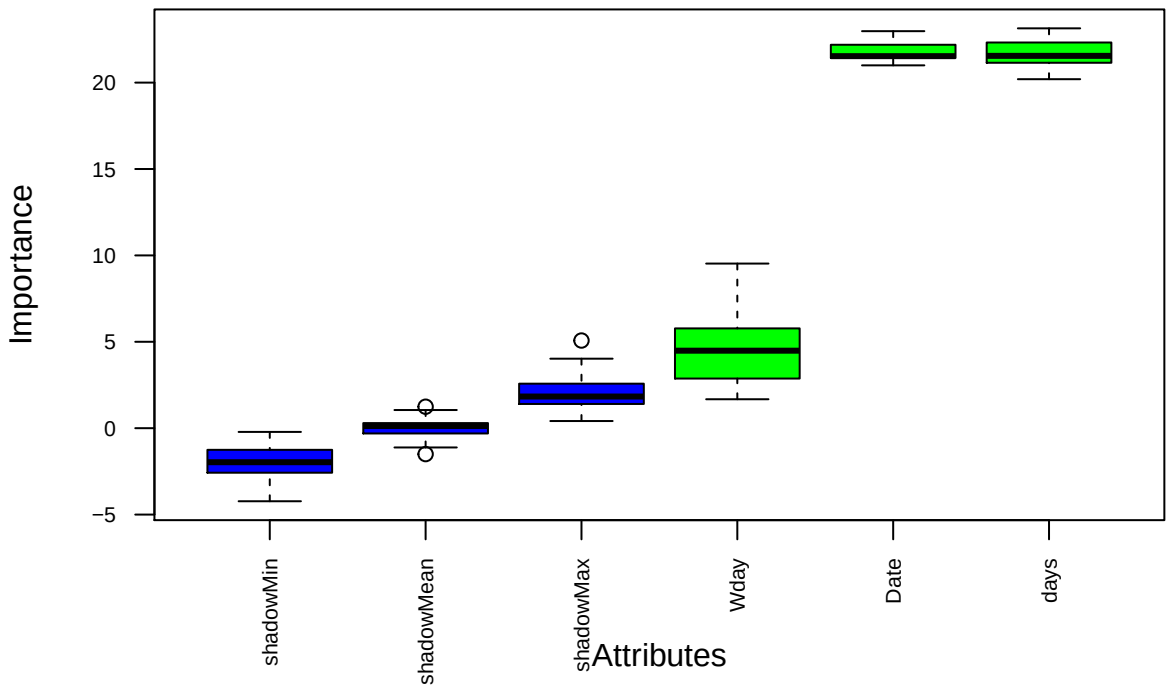
```
##  1. run of importance source...
##  2. run of importance source...
```

```
##  3. run of importance source...
##  4. run of importance source...
##  5. run of importance source...
##  6. run of importance source...
##  7. run of importance source...
##  8. run of importance source...
##  9. run of importance source...
## After 9 iterations, +0.46 secs:
##  confirmed 2 attributes: Date, days;
##  still have 1 attribute left.
##  10. run of importance source...
##  11. run of importance source...
##  12. run of importance source...
## After 12 iterations, +0.7 secs:
##  confirmed 1 attribute: Wday;
##  no more attributes left.
```

```
print(Br_Fat)
```

```
## Boruta performed 12 iterations in 0.703567 secs.
##  3 attributes confirmed important: Date, days, Wday;
##  No attributes deemed unimportant.
```

```
plot(Br_Fat,las=2,cex.axis=0.7) # las: to make lables vertical, cex.axis for scaling view
```




```
#We get approximately the same results on Fatality cases
```

Modeling

Now we now that Date is the main predictor lets see what is the best model to fit our data.

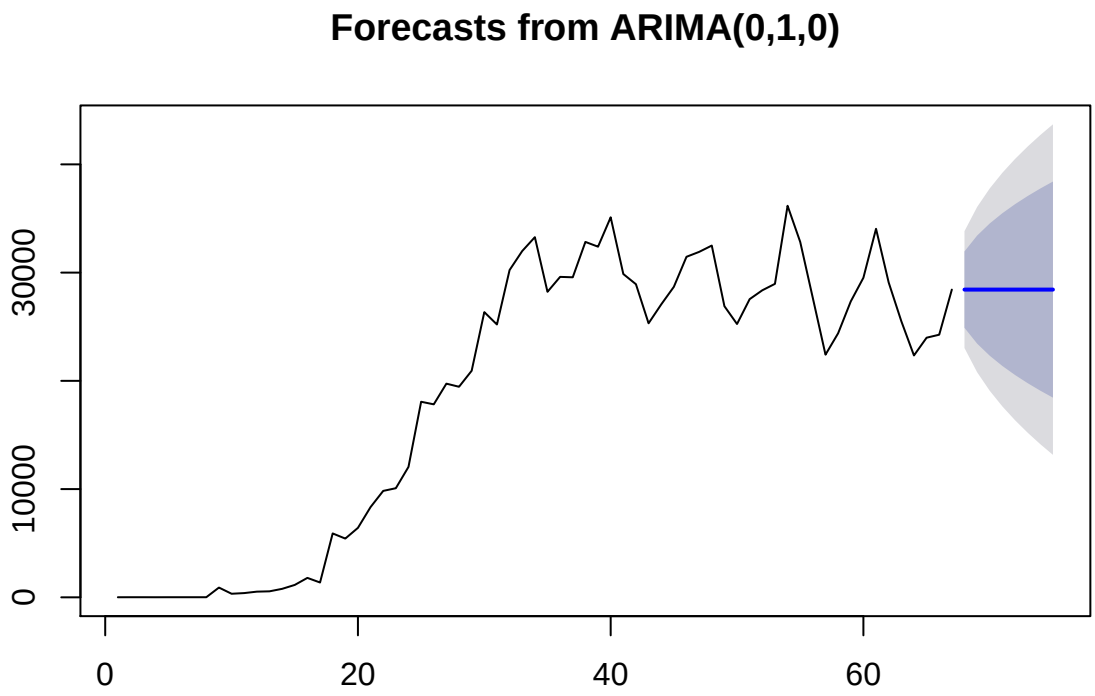
Auto ARIMA Model

first we can try Aut arima as it is a well known time series forecasting method

```
## Modeleling
Location <- "US"
# Confirmed cases fitting
fit <- auto.arima(select(data.frame(train_Con_dt) ,TargetValue))
acc<- forecast::accuracy(fit)
acc
```

```
##                      ME     RMSE      MAE      MPE     MAPE      MASE       ACF1
## Training set 424.2028 2733.268 1951.696 3.496893 16.72457 0.9850747 0.06287537
```

```
training_Con_RMSE<-acc[1,2]


Forc <- forecast(fit,h=nrow(test_Conf_USA))
plot(Forc)
```

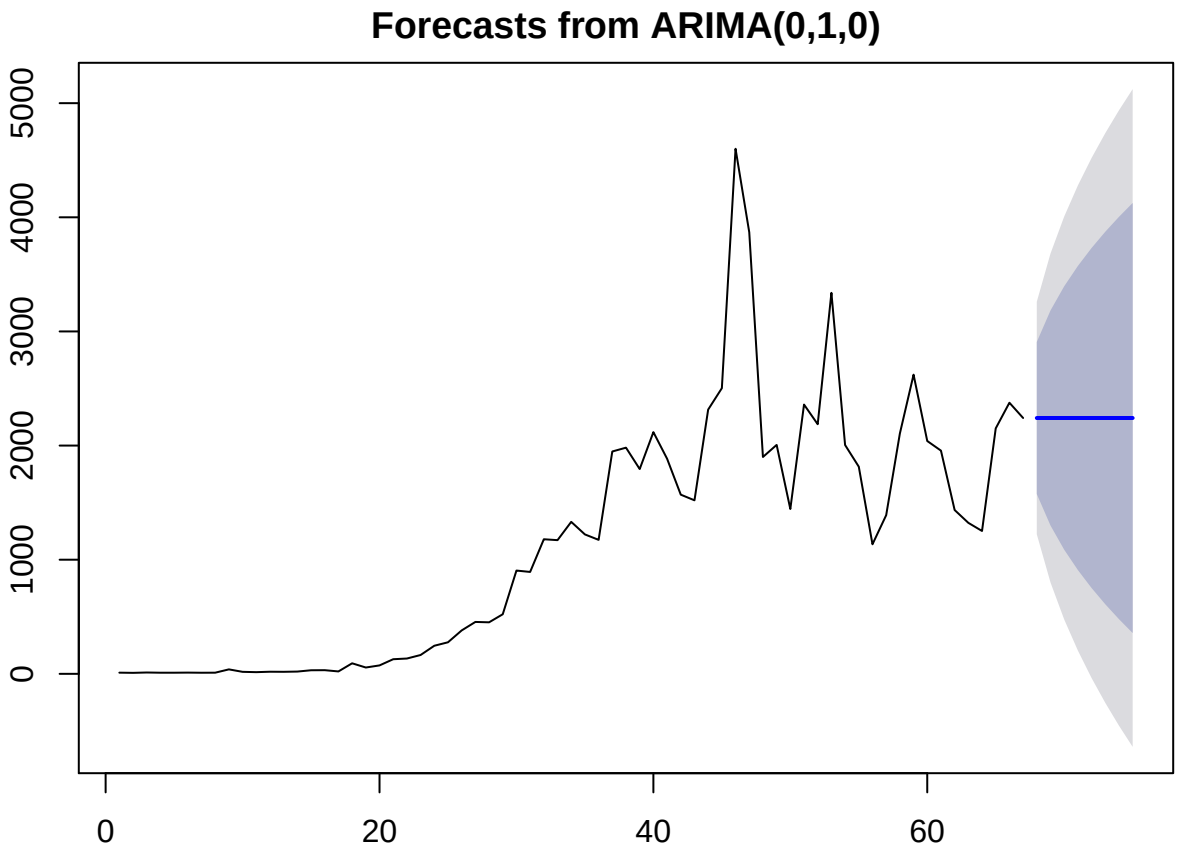


Calculate RMSE for Confirmed cases

```
#
ForcMean<- as.numeric(Forc$mean)
Conf_RMSE<-rmse(ForcMean,test_Conf_USA$TargetValue)
Conf_R2 <-R2_Score(ForcMean,test_Conf_USA$TargetValue)
```

Fit Auto ARIMA on Fatilities

```
fit <- auto.arima(select(data.frame(train_Fat_dt) ,TargetValue))
Forc <- forecast(fit,h=nrow(test_Fat_USA))
par(mar=c(2,2,2,2))
plot(Forc)
```



calculate RMSE

```
# calculate RMSE
ForcMean<- as.numeric(Forc$mean)
Fat_RMSE<-rmse(ForcMean,test_Fat_USA$TargetValue) #1800.502
Fat_R2 <-R2_Score(ForcMean,test_Fat_USA$TargetValue)
```

Show Results of Auto ARIMA

```
# Show RMSE results
rmse_results <- tibble(Method = "Auto-ARIMA", Conf_RMSE = Conf_RMSE, Fat_RMSE=Fat_RMSE,Conf_R2=Conf_R2,
rmse_results %>% knitr::kable()
```

Method	Conf_RMSE	Fat_RMSE	Conf_R2	Fat_R2	Location
Auto-ARIMA	6193.155	858.5338	-3.350081	-5.090424	US

Now lets try other regression models but first we need to initialize a parallel processing method. here we use H2O framework

```
############################################################
## Using H2O library to perform the task in parallel way
h2o.init(nthreads = 2, #Number of threads -1 means use all cores on your machine
                       max_mem_size = "7G")  #max mem size is the maximum memory to allocate to H2O
```

```
##
## H2O is not running yet, starting it now...
##
```

```
## Note:  In case of errors look at the following log files:
##     C:\Users\AHB\AppData\Local\Temp\Rtmp42k3q6\file257c7ba824d2/h2o_AHB_started_from_r.out
##     C:\Users\AHB\AppData\Local\Temp\Rtmp42k3q6\file257c13321995/h2o_AHB_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         5 seconds 179 milliseconds
##     H2O cluster timezone:       Asia/Beirut
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.1
##     H2O cluster version age:    1 month and 28 days
##     H2O cluster name:           H2O_started_from_R_AHB_fyn762
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.00 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  2
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.3 (2020-02-29)
```

Prepare H2o dataframes and parameters

```
train_Con_h2o <- as.h2o(train_Con_dt)
```

```
##   |                                                                      |
```

```
train_Fat_h2o <- as.h2o(train_Fat_dt)
```

```
##   |                                                                      |
```

```
test_Conf_h2o<-as.h2o(test_Conf_USA)
```

```
##   |                                                                      |
```

```
test_Fat_h2o <- as.h2o(test_Fat_USA)
```

```
##   |                                                                      |
```

```
x<-"days"
y<-"TargetValue"
nfolds<-5
```

Random Forest Model

first lets try to fit on Confirmed Cases

```
# fit rf model
rf_model_Con_h2o<- h2o.randomForest(x = x,
                                    y = y,
                                    training_frame = train_Con_h2o,
                                    ntrees = 50,
                                    nfolds = nfolds,
```

```
                                    keep_cross_validation_predictions = TRUE,
                                    seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(rf_model_Con_h2o,newdata = test_Conf_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(rf_model_Con_h2o, newdata = test_Conf_h2o)
print(perf)
```

```
## H2ORegressionMetrics: drf
##
## MSE:  20767995
## RMSE:  4557.192
## MAE:  3688.241
## RMSLE:  0.1979432
## Mean Residual Deviance :  20767995
```

```
Conf_RMSE <- perf@metrics[["RMSE"]]
Conf_R2<-perf@metrics[["r2"]]
```

Then on Fatalities

```
############## Fiting Fatalities############
rf_model_Fat_h2o<- h2o.randomForest(x = x,
                                    y = y,
                                    training_frame = train_Fat_h2o,
                                    ntrees = 50,
                                    nfolds = nfolds,
                                    keep_cross_validation_predictions = TRUE,
                                    seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(rf_model_Fat_h2o,newdata = test_Fat_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(rf_model_Fat_h2o, newdata = test_Fat_h2o)
print(perf)
```

```
## H2ORegressionMetrics: drf
##
## MSE:  770694.1
## RMSE:  877.8918
## MAE:  806.0219
## RMSLE:  0.5549679
## Mean Residual Deviance :  770694.1
```

```
Fat_RMSE <- perf@metrics[["RMSE"]]
Fat_R2<-perf@metrics[["r2"]]
```

Show and compare results

```
# Show RMSE  and R-score results
rmse_results <- rbind(rmse_results,tibble(Method = perf@algorithm, Conf_RMSE = Conf_RMSE,Conf_R2=Conf_R
rmse_results %>%  knitr::kable()
```

Method	Conf_RMSE	Fat_RMSE	Conf_R2	Fat_R2	Location
Auto-ARIMA	6193.155	858.5338	-3.350081	-5.090424	US
drf	4557.192	877.8918	-1.355417	-5.368171	US

Gradient Boosting Machine Model

fit Confirmed cases

```
############################################################
# Train & Cross-validate a Gradient Boosting Machine
gbm_model_Con_h2o <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train_Con_h2o,
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
                  nfolds = nfolds,
                  keep_cross_validation_predictions = TRUE,
                  seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(gbm_model_Con_h2o,newdata = test_Conf_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(gbm_model_Con_h2o, newdata = test_Conf_h2o)
print(perf)
```

```
## H2ORegressionMetrics: gbm
##
## MSE:  12071108
## RMSE:  3474.35
## MAE:  2772.226
## RMSLE:  0.1554879
## Mean Residual Deviance :  12071108
```

```
Conf_RMSE <- perf@metrics[["RMSE"]]
Conf_R2<-perf@metrics[["r2"]]
```

Fit Fatalities

```
###### Fit Fatalities
gbm_model_Fat_h2o <- h2o.gbm(x = x,
                             y = y,
                             training_frame = train_Fat_h2o,
                             ntrees = 10,
                             max_depth = 3,
                             min_rows = 2,
                             learn_rate = 0.2,
                             nfolds = nfolds,
                             keep_cross_validation_predictions = TRUE,
                             seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(gbm_model_Fat_h2o,newdata = test_Fat_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(gbm_model_Fat_h2o, newdata = test_Fat_h2o)
print(perf)
```

```
## H2ORegressionMetrics: gbm
##
## MSE:  381271.1
## RMSE:  617.4715
## MAE:  510.1453
## RMSLE:  0.4407054
## Mean Residual Deviance :  381271.1
```

```
Fat_RMSE <- perf@metrics[["RMSE"]]
Fat_R2<-perf@metrics[["r2"]]
```

Show results

```
# Show  results
rmse_results <- rbind(rmse_results,tibble(Method = perf@algorithm, Conf_RMSE = Conf_RMSE,Conf_R2=Conf_R
rmse_results %>%  knitr::kable()
```

Method	Conf_RMSE	Fat_RMSE	Conf_R2	Fat_R2	Location
Auto-ARIMA	6193.155	858.5338	-3.3500806	-5.090424	US
drf	4557.192	877.8918	-1.3554173	-5.368171	US
gbm	3474.350	617.4715	-0.3690536	-2.150406	US

Generalized Linear Regression Model

Ffit Confirmed Cases

```
### Generalized Linear regression
glm_model_Con_h2o <- h2o.glm(x = x,
                             y = y,
                             nfolds = nfolds,
                             alpha=0.5,
                             training_frame = train_Con_h2o,
                             keep_cross_validation_predictions = TRUE,
                             seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(glm_model_Con_h2o,newdata = test_Conf_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(glm_model_Con_h2o, newdata = test_Conf_h2o)
print(perf)
```

```
## H2ORegressionMetrics: glm
##
## MSE:  14406061
## RMSE:  3795.532
## MAE:  3073.619
## RMSLE:  0.1645976
```

```
## Mean Residual Deviance :  14406061
## R^2 :  -0.633874
## Null Deviance :213452481
## Null D.o.F. :7
## Residual Deviance :115248490
## Residual D.o.F. :6
## AIC :160.5683
```

```
Conf_RMSE <- perf@metrics[["RMSE"]]
Conf_R2<-perf@metrics[["r2"]]


#########################################################################
```

Fit Fatalities

```
# Fit Fatalities
glm_model_Fat_h2o <- h2o.glm(x = x,
                             y = y,
                             nfolds = nfolds,
                             alpha=0.5,
                             training_frame = train_Fat_h2o,
                             keep_cross_validation_predictions = TRUE,
                             seed = 1)
```

```
##   |                                                                      |
```

```
Prd <- h2o.predict(glm_model_Fat_h2o,newdata = test_Fat_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(glm_model_Fat_h2o, newdata = test_Fat_h2o)
print(perf)
```

```
## H2ORegressionMetrics: glm
##
## MSE:  467218.8
## RMSE:  683.5341
## MAE:  595.0582
## RMSLE:  0.4690022
## Mean Residual Deviance :  467218.8
## R^2 :  -2.860584
## Null Deviance :1774212
## Null D.o.F. :7
## Residual Deviance :3737751
## Residual D.o.F. :6
## AIC :133.1394
```

```
Fat_RMSE <- perf@metrics[["RMSE"]]
Fat_R2<-perf@metrics[["r2"]]
```

Show results

```
# Show  results
rmse_results <- rbind(rmse_results,tibble(Method = perf@algorithm, Conf_RMSE = Conf_RMSE,Conf_R2=Conf_R
rmse_results %>%  knitr::kable()
```

Method	Conf_RMSE	Fat_RMSE	Conf_R2	Fat_R2	Location
Auto-ARIMA	6193.155	858.5338	-3.3500806	-5.090424	US
drf	4557.192	877.8918	-1.3554173	-5.368171	US
gbm	3474.350	617.4715	-0.3690536	-2.150406	US
glm	3795.532	683.5341	-0.6338740	-2.860584	US

Stack Ensembler Model

Train a stacked ensemble using the above models

```
#Fit Stack ensembler

# Train a stacked ensemble using the  above models
ensemble_model_Con_h2o <- h2o.stackedEnsemble(x = x,
                                  y = y,
                                  training_frame = train_Con_h2o,
                                  base_models = list(glm_model_Con_h2o,gbm_model_Con_h2o,rf_model_Con_h2o
```

```
##   |                                                                      |
```

```
# Eval ensemble performance on a test set
Prd <- h2o.predict(ensemble_model_Con_h2o,newdata = test_Conf_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(ensemble_model_Con_h2o, newdata = test_Conf_h2o)
print(perf)
```

```
## H2ORegressionMetrics: stackedensemble
##
## MSE:  20966332
## RMSE:  4578.901
## MAE:  3709.667
## RMSLE:  0.1987544
## Mean Residual Deviance :  20966332
```

```
Conf_RMSE <- perf@metrics[["RMSE"]]
Conf_R2<-perf@metrics[["r2"]]
```

Fit fatalities

```
###### Fit fatalities
# Train a stacked ensemble using the  above models
ensemble_model_Fat_h2o <- h2o.stackedEnsemble(x = x,
                                              y = y,
                                              training_frame = train_Fat_h2o,
                                              base_models = list(glm_model_Fat_h2o,gbm_model_Fat_h2o,rf
```

```
##   |                                                                      |
```

```
# Eval ensemble performance on a test set
Prd <- h2o.predict(ensemble_model_Fat_h2o,newdata = test_Fat_h2o)
```

```
##   |                                                                      |
```

```
perf <- h2o.performance(ensemble_model_Fat_h2o, newdata = test_Fat_h2o)
print(perf)
```

```
## H2ORegressionMetrics: stackedensemble
##
## MSE:  890059
## RMSE:  943.4294
## MAE:  877.6677
## RMSLE:  0.5814848
## Mean Residual Deviance :  890059
```

```
Fat_RMSE <- perf@metrics[["RMSE"]]
Fat_R2<-perf@metrics[["r2"]]
```

Show results

```
# Show  results
rmse_results <- rbind(rmse_results,tibble(Method = perf@algorithm, Conf_RMSE = Conf_RMSE,Conf_R2=Conf_R
rmse_results %>%  knitr::kable()
```

Method	Conf_RMSE	Fat_RMSE	Conf_R2	Fat_R2	Location
Auto-ARIMA	6193.155	858.5338	-3.3500806	-5.090424	US
drf	4557.192	877.8918	-1.3554173	-5.368171	US
gbm	3474.350	617.4715	-0.3690536	-2.150406	US
glm	3795.532	683.5341	-0.6338740	-2.860584	US
stackedensemble	4578.901	943.4294	-1.3779119	-6.354471	US

Now we had fit 5 models on both Fatalities and confirmed cases, from the results we can see that Gradient Boosting Model perform the best on both Fatalities and Confirmed cases, where, auto ARIMA and Stack Ensembler performed worse.

Lets fit the best 3 mdels on different location to see the effect of the location on the models.

Prepare a function to handle the model evaluations

```
## Modeling different regions with different Models by selecting set of places from the dataset

fit_Model <- function(Model,train_Con_h2o,test_Con_h2o,train_Fat_h2o, test_Fat_h2o,x="Date",y="TargetVa
{

  nfolds<-5

  if(Model=="drf"){
    # fit rf model
    rf_model_Con_h2o<- h2o.randomForest(x = x,
                                        y = y,
                                        training_frame = train_Con_h2o,
                                        ntrees = 50,
                                        nfolds = nfolds,
                                        keep_cross_validation_predictions = TRUE,
                                        seed = 1)


    #Prd_Con <- h2o.predict(rf_model_Con_h2o,newdata = test_Con_h2o)
    perf <- h2o.performance(rf_model_Con_h2o, newdata = test_Con_h2o)
    #print(perf)
    Conf_RMSE <- perf@metrics[["RMSE"]]
    Conf_R2<-perf@metrics[["r2"]]
```

```
  ############# Fiting Fatalities############
  rf_model_Fat_h2o<- h2o.randomForest(x = x,
                                      y = y,
                                      training_frame = train_Fat_h2o,
                                      ntrees = 50,
                                      nfolds = nfolds,
                                      keep_cross_validation_predictions = TRUE,
                                      seed = 1)


  #Prd_Fat <- h2o.predict(rf_model_Fat_h2o,newdata = test_Fat_h2o)
  perf <- h2o.performance(rf_model_Fat_h2o, newdata = test_Fat_h2o)
  #print(perf)
  Fat_RMSE <- perf@metrics[["RMSE"]]
  Fat_R2<-perf@metrics[["r2"]]

}else if(Model=="gbm"){
  # Train & Cross-validate a Gradient Boosting Machine
  gbm_model_Con_h2o <- h2o.gbm(x = x,
                               y = y,
                               training_frame = train_Con_h2o,
                               ntrees = 10,
                               max_depth = 3,
                               min_rows = 2,
                               learn_rate = 0.2,
                               nfolds = nfolds,
                               keep_cross_validation_predictions = TRUE,
                               seed = 1)
  #Prd_Con <- h2o.predict(gbm_model_Con_h2o,newdata = test_Con_h2o)
  perf <- h2o.performance(gbm_model_Con_h2o, newdata = test_Con_h2o)
  #print(perf)
  Conf_RMSE <- perf@metrics[["RMSE"]]
  Conf_R2<-perf@metrics[["r2"]]
  ###### Fit Fatalities
  gbm_model_Fat_h2o <- h2o.gbm(x = x,
                               y = y,
                               training_frame = train_Fat_h2o,
                               ntrees = 10,
                               max_depth = 3,
                               min_rows = 2,
                               learn_rate = 0.2,
                               nfolds = nfolds,
                               keep_cross_validation_predictions = TRUE,
                               seed = 1)
  #Prd_Fat <- h2o.predict(gbm_model_Fat_h2o,newdata = test_Fat_h2o)
  perf <- h2o.performance(gbm_model_Fat_h2o, newdata = test_Fat_h2o)
  #print(perf)
  Fat_RMSE <- perf@metrics[["RMSE"]]
  Fat_R2<-perf@metrics[["r2"]]

}else if(Model=="glm"){

  ### Generalized Linear regression
```

```
    glm_model_Con_h2o <- h2o.glm(x = x,
                                 y = y,
                                 nfolds = nfolds,
                                 alpha=0.5,
                                 training_frame = train_Con_h2o,
                                 keep_cross_validation_predictions = TRUE,
                                 seed = 1)

    #Prd_Con <- h2o.predict(glm_model_Con_h2o,newdata = test_Con_h2o)
    perf <- h2o.performance(glm_model_Con_h2o, newdata = test_Con_h2o)
    #print(perf)
    Conf_RMSE <- perf@metrics[["RMSE"]]
    Conf_R2<-perf@metrics[["r2"]]

    # Fit Fatalities
    glm_model_Fat_h2o <- h2o.glm(x = x,
                                 y = y,
                                 nfolds = nfolds,
                                 alpha=0.5,
                                 training_frame = train_Fat_h2o,
                                 keep_cross_validation_predictions = TRUE,
                                 seed = 1)

    #Prd_Fat <- h2o.predict(glm_model_Fat_h2o,newdata = test_Fat_h2o)
    perf <- h2o.performance(glm_model_Fat_h2o, newdata = test_Fat_h2o)
    #print(perf)
    Fat_RMSE <- perf@metrics[["RMSE"]]
    Fat_R2<-perf@metrics[["r2"]]

  }

  list(Conf_RMSE=Conf_RMSE,Fat_RMSE=Fat_RMSE,Conf_R2=Conf_R2, Fat_R2=Fat_R2)

}
```

Prepare function to apply Models to places

```
 Apply_Models<-function(Md=Mdls,tr_Con=tr_C_h2o,ts_Con=ts_C_h2o,tr_Fat=tr_F_h2o,ts_Fat=ts_F_h2o,pl=plac
lapply(pl,function(p){
  tr_Con_CPR <-  tr_Con[tr_Con[,"CPR"]==p,]
  ts_Con_CPR  <-ts_Con[ts_Con[,"CPR"]==p ,]
  tr_Fat_CPR <- tr_Fat[tr_Fat[,"CPR"]==p,]
  ts_Fat_CPR <- ts_Fat[ts_Fat[,"CPR"]==p,]
 lapply(Md,function(M){
    fit_Model(M,tr_Con_CPR,ts_Con_CPR,tr_Fat_CPR,ts_Fat_CPR,x="Date",y="TargetValue")
    })
})
}
```

Now lets select 10% of the places in the Dataset and apply the best 3 models on them.

```
#Lets compare models on diferrent locations
Mdls<- c("gbm","drf","glm") #, ,
# first lets select a sample of 10% of the place where 70% are from USA
SmpSize<-0.1*(length(unique(train_Kag_clean$CPR)))
```

```
set.seed(2020)
Sample_US <- sample(train_Conf$CPR[str_detect(train_Conf$CPR,"_US\\b")],round(SmpSize*0.7))
set.seed(2020)
Sample_NotUS<- sample(train_Conf$CPR[!str_detect(train_Conf$CPR,"_US\\b")],round(SmpSize*0.3))
set.seed(2020)
Sample_CPR <-sample(c(as.character(Sample_US) ,as.character(Sample_NotUS)))
Sample_CPR %>%  head() %>% knitr::kable()
```

x
Suffolk_Massachusetts_US
Franklin_Tennessee_US
Issaquena_Mississippi_US
Latimer_Oklahoma_US
NA_NA_Bolivia
NA_NA_Algeria

```
places <-Sample_CPR
# lets use H2O library to do it in parallel
```

Lets prepare the training and test datasets

```
tr_C_h2o<-as.h2o(select(train_Conf,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
ts_C_h2o<-as.h2o(select(test_Conf,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
tr_F_h2o<-as.h2o(select(train_Fat,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
ts_F_h2o<-as.h2o(select(test_Fat,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
x<-"Date"
y<-"TargetValue"
```

Now we can apply places to the 3 selected models and show results

```
#### this will take about an hour on modest pc
tic("Total  time")
#results <- Apply_Models() # I had already save the results
load("Var/ML_Res2020-05-29_19-35.rda")# if you want to generate your own results please comment this li
toc()
```

```
## Total  time: 0.02 sec elapsed
```

```
####### Show results
ShowResults_tbl <- function(M=Mdls,p=places, r=results){
  RMSE_results<- tibble(Model=character(),Place=character(),Conf_RMSE=numeric(),Fat_RMSE=numeric(),Conf
  for(j in 1:length(p)){

    for (i in 1:length(M)){
      RMSE_results <- bind_rows(RMSE_results,tibble( Model = M[i],Place=p[j], Conf_RMSE=r[[j]][[i]][["C
```

```
    }
    #RMSE_results <- bind_rows(RMSE_results,tibble( Model = "---------",Place="---------",Conf_RMSE=000
  }

  RMSE_results
}
#Res_clean<-ShowResults_tbl() # if you want to generate your own results uncomment this tag

##### Save result to file
#current_DT<-format(Sys.time(), "%Y-%m-%d_%H-%M")
#Resfile<- paste0("Var/ML_Res",current_DT,".rda")
#save(Res_clean,results , file = Resfile)
#########################################################################

#########################################################################
```

Lets see which model has the Highest average R2 score

```
# Lets see which model has the Highest average R2 score
Res_clean %>%  select(Place,Model,Conf_R2) %>% group_by (Model)%>%
  summarise(Conf_R2=sum(Conf_R2)/n())%>% arrange(desc(Conf_R2))%>% knitr::kable()
```

Model	Conf_R2
gbm	-2.059407
drf	-2.711928
glm	-11.954231

```
Res_clean %>% select(Place,Model,Fat_R2) %>% group_by (Model)%>%
  summarise(Fat_R2=sum(Fat_R2)/n())%>% arrange(desc(Fat_R2))%>% knitr::kable()
```

Model	Fa
gbm	-0.80
drf	-1.03
glm	-13.14

It is ob vious that Gradient Boosting Machine Model has the best Average R-Score for both Confirmed cases and Fatality

Final Training and Prediction

Now lets train Gradient Boosting Machine on the whole training set ## Prediction Function

```
## Prediction Function
Predic_gbm <- function(train_Con_h2o,test_Con_h2o,train_Fat_h2o, test_Fat_h2o,x="Date",y="TargetValue"
{


  nfolds<-5

  # Train & Cross-validate a Gradient Boosting Machine
  gbm_model_Con_h2o <- h2o.gbm(x = x,
                               y = y,
                               training_frame = train_Con_h2o,
```

```
                                ntrees = 10,
                                max_depth = 3,
                                min_rows = 2,
                                learn_rate = 0.2,
                                nfolds = nfolds,
                                keep_cross_validation_predictions = TRUE,
                                seed = 1)

  Prd_Con <-h2o.predict(gbm_model_Con_h2o,newdata = test_Con_h2o)
  #Mean Absolute Error
  mae_Con <-gbm_model_Con_h2o@model[["training_metrics"]]@metrics[["mae"]]
  RMSE_Con <-gbm_model_Con_h2o@model[["training_metrics"]]@metrics[["RMSE"]]
  ###### Fit Fatalities
  gbm_model_Fat_h2o <- h2o.gbm(x = x,
                                y = y,
                                training_frame = train_Fat_h2o,
                                ntrees = 10,
                                max_depth = 3,
                                min_rows = 2,
                                learn_rate = 0.2,
                                nfolds = nfolds,
                                keep_cross_validation_predictions = TRUE,
                                seed = 1)

  Prd_Fat <- h2o.predict(gbm_model_Fat_h2o,newdata = test_Fat_h2o)
  ##Mean Absolute Error
  mae_Fat<-gbm_model_Fat_h2o@model[["training_metrics"]]@metrics[["mae"]]
  RMSE_Fat <-gbm_model_Fat_h2o@model[["training_metrics"]]@metrics[["RMSE"]]
  list(data.frame(Predicted_Confirmed=as.vector(Prd_Con),Predicted_Fatilities=as.vector(Prd_Fat)),mae_C

}
```

Function to apply locations to the prediction function

```
Apply_CPR<-function(tr_Con=tr_C_h2o,ts_Con=ts_C_h2o,tr_Fat=tr_F_h2o,ts_Fat=ts_F_h2o,pl=places,x="Date",
  lapply(pl,function(p){
    tr_Con_CPR <-  tr_Con[tr_Con[,"CPR"]==p,]
    ts_Con_CPR  <-ts_Con[ts_Con[,"CPR"]==p ,]

    tr_Fat_CPR <- tr_Fat[tr_Fat[,"CPR"]==p,]
    ts_Fat_CPR <- ts_Fat[ts_Fat[,"CPR"]==p,]

      Predic_gbm(tr_Con_CPR,ts_Con_CPR,tr_Fat_CPR,ts_Fat_CPR,x="Date",y="TargetValue")
    })
}
```

Prepare all training and testing dataset

```
train_Kag_Con<- train_Kag_clean %>% dplyr::filter(Target=="ConfirmedCases")
test_Kag_Con<- test_Kag_clean %>% dplyr::filter(Target=="ConfirmedCases")

train_Kag_Fat <-train_Kag_clean %>% dplyr::filter(Target=="Fatalities")
test_Kag_Fat <-test_Kag_clean %>% dplyr::filter(Target=="Fatalities")

tr_C_h2o<-as.h2o(select(train_Kag_Con,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
ts_C_h2o<-as.h2o(select(test_Kag_Con,CPR,Date))
```

```
##   |                                                                      |
```

```
tr_F_h2o<-as.h2o(select(train_Kag_Fat,CPR,Date, TargetValue))
```

```
##   |                                                                      |
```

```
ts_F_h2o<-as.h2o(select(test_Kag_Fat,CPR,Date))
```

```
##   |                                                                      |
```

```
x<-"Date"
y<-"TargetValue"
```

Start fitting and prediction on Gradient Boosting Model

```
All_CPR<-   as.vector(unique(test_Kag_clean$CPR))
places<-All_CPR
#### this will take about an hour on modest pc
tic("Total  time")
#gmb_Res <- Apply_CPR() # I had already save the results
load("Var/gmb_Res2020-05-30_19-17.rda")# if you want to generate your own results please comment this l
toc() #Total  time: 5543.53 sec elapsed
```

```
## Total  time: 0.11 sec elapsed
```

```
##### Save result to file
#current_DT<-format(Sys.time(), "%Y-%m-%d_%H-%M")
#Resfile<- paste0("Var/gmb_Res",current_DT,".rda")
#save(gmb_Res , file = Resfile)
```

Process Prediction Results

```
#### Process Prediction Results

predicted_results<- tibble(ForecastId=numeric(),CPR=character(),ConfirmedCases=numeric(),Fatalities=num
Predlength<-length(gmb_Res[[1]][[1]][["Predicted_Confirmed"]])
for(i in 1:length(gmb_Res)){
  CPR_i<- rep(All_CPR[i],Predlength)
    predicted_results <- bind_rows(predicted_results,tibble(ForecastId=i,
                                                            CPR=CPR_i,
                                                            ConfirmedCases=gmb_Res[[i]][[1]][["Predicte
                                                            Fatalities=gmb_Res[[i]][[1]][["Predicted_Fa
                                    mae_Con=rep(gmb_Res[[i]][["mae_Con"]],Predlength),
                                    mae_Fat=rep(gmb_Res[[i]][["mae_Fat"]],Predlength)))

}

predicted <- predicted_results %>% pivot_longer(c("ConfirmedCases","Fatalities"), names_to = "Target", v
  arrange(ForecastId) %>% mutate(ForecastId=seq(1:nrow(.))) %>% left_join(select(test_Kag,Date,Forecastl
predicted <-as.data.frame(predicted)
```

Prepare final results for evaluation

```
# adjust columns to be according to the follwing structure "ForecastId", "q_0.05","q_0.5","q_0.95"
### use mid quantile only for predicted data
Predicted_C <- predicted %>% dplyr::filter(Target=="ConfirmedCases") %>%
  select(ForecastId,mae_Con,TargetValue,Date)%>%
  mutate(q_0.05=qnorm(0.05,TargetValue,mae_Con),
         q_0.5=TargetValue,
         q_0.95=qnorm(0.95,TargetValue,mae_Con))%>%
  select(-mae_Con,-TargetValue)

Predicted_F <- predicted %>% dplyr::filter(Target=="Fatalities") %>%
  select(ForecastId,mae_Fat,TargetValue,Date)%>%
  mutate(q_0.05=qnorm(0.05,TargetValue,mae_Fat),
         q_0.5=TargetValue,
         q_0.95=qnorm(0.95,
                      TargetValue,mae_Fat))%>%
  select(-mae_Fat,-TargetValue)
```

Evaluate Using Pinball evaluation Method

Evaluating our final model after trained on the hole training set and using the common dates between test and train sets for validation.

##Preparing functions

```
#Pinball evaluation performance preparation

pinball <- function(true_array, predicted_array, tau, weight){
  array <- predicted_array * (predicted_array > 0)
  abs_diff <- abs(true_array - array)
  result <- abs_diff * (1 -tau) * (array > true_array) + abs_diff * (tau) * (array <= true_array)
  result <- (mean(result)) * weight
  return (mean(result))
}
Avg_loss<- function(true_array, mean_array, min_array, max_array, weights){
  result <- (pinball(true_array, max_array, 0.95, weights) +
               pinball(true_array, min_array, 0.05, weights) +
               pinball(true_array, mean_array, 0.5, weights))
  return (result / 3)
}
# Dates intersection between training and testing
minDate<-min(test_Kag$Date)
maxDate<-max(train_Kag$Date)
# Actual Values from original Trian Kaggle Dataset
TargetVal_C_trained <- train_Kag %>% dplyr::filter(Target=="ConfirmedCases" & Date>= minDate)%>% .$Targ
TargetVal_F_trained <- train_Kag %>% dplyr::filter (Target=="Fatalities" & Date>= minDate) %>% .$Target

weights_C <-train_Kag %>% dplyr::filter(Target=="ConfirmedCases" & Date>= minDate)%>% .$Weight
weights_F <-train_Kag %>% dplyr::filter(Target=="Fatalities" & Date>= minDate)%>% .$Weight

# prepare masks for intersection period between training and testing
Prd_Dt_Msk_C<- Predicted_C[,"Date"]<= maxDate & Predicted_C[,"Date"]>= minDate
Prd_Dt_Msk_F<- Predicted_F[,"Date"]<= maxDate & Predicted_F[,"Date"]>= minDate
```

Pinball results for Confirmed Cases

```
# Pinball results for Confirmed Cases
Pinball_Results_Con <- Avg_loss(TargetVal_C_trained,
                                Predicted_C[Prd_Dt_Msk_C,]$q_0.5,
                                Predicted_C[Prd_Dt_Msk_C,]$q_0.05,
                                Predicted_C[Prd_Dt_Msk_C,]$q_0.95,
                                weights_C)
print(paste("Pinball Result for Confirmed Cases:",Pinball_Results_Con))
```

```
## [1] "Pinball Result for Confirmed Cases: 0.222660356402174"
```

Pinball Results for Fatalities

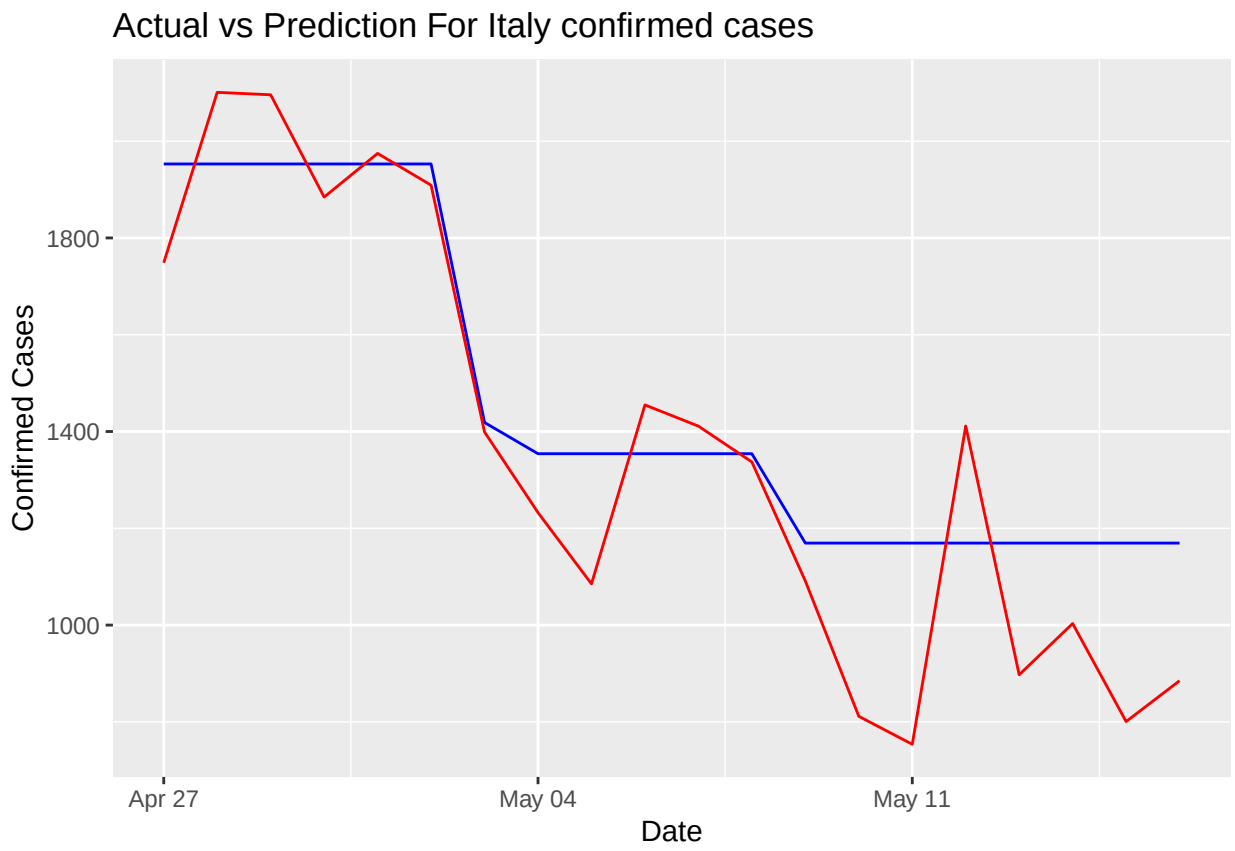
```
# Pinball Results for Fatalities
Pinball_Results_Fat <- Avg_loss(TargetVal_F_trained,
                                Predicted_F[Prd_Dt_Msk_F,]$q_0.5,
                                Predicted_F[Prd_Dt_Msk_F,]$q_0.05,
                                Predicted_F[Prd_Dt_Msk_F,]$q_0.95,
                                weights_F)
print(paste("Pinball Result for Fatality  Cases:",Pinball_Results_Fat))
```

```
## [1] "Pinball Result for Fatality  Cases: 0.420488149142684"
```

Show italy case predictions

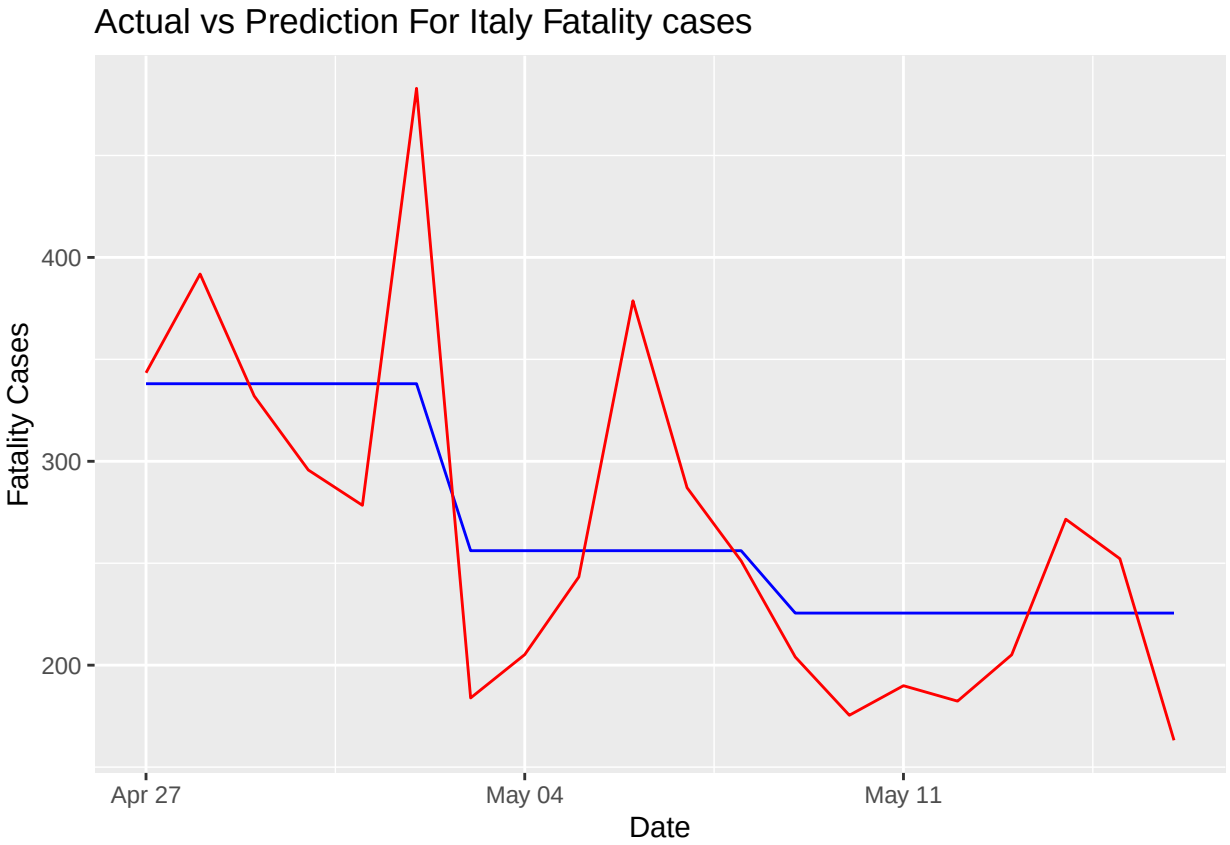
Confirmed Cases Prediction Italy Case

```
######################### Plot Prediction Results
# Italy Results
Italy_C_IdDate<-test_Kag %>% dplyr::filter(Country_Region=="Italy" & Target=="ConfirmedCases" & Date <=m
Italy_F_IdDate<-test_Kag %>% dplyr::filter(Country_Region=="Italy" & Target=="Fatalities" & Date <=maxDa

Italy_C_test<-train_Kag %>% dplyr::filter(Country_Region=="Italy" & Target=="ConfirmedCases" & Date >=mi
Italy_F_test<-train_Kag %>% dplyr::filter(Country_Region=="Italy" & Target=="Fatalities"& Date >=minDate

Prd_It_C<-Predicted_C%>% select(-Date)%>%inner_join(Italy_C_IdDate, by="ForecastId")
Prd_It_F<-Predicted_F%>% select(-Date)%>%inner_join(Italy_F_IdDate, by="ForecastId")

ggplot(aes(x=Date), data=Prd_It_C) +
  geom_line(aes(y=q_0.5), col="blue",data=Prd_It_C) +
  geom_line(aes(y=TargetValue), data =Italy_C_test, col="red" ) +
  labs(colour="TargetValue",
       x="Date",
       y="Confirmed Cases") +
  scale_color_manual(name="", values = c("red","blue")) +
  scale_fill_manual(name="", values=c("red","blue"))+
  ggtitle("Actual vs Prediction For Italy confirmed cases")
```



Fatalities Prediction Italy case

```
ggplot(aes(x=Date), data=Prd_It_F) +
  geom_line(aes(y=q_0.5), col="blue",data=Prd_It_F) +
  geom_line(aes(y=TargetValue), data =Italy_F_test, col="red" ) +
  labs(colour="TargetValue",
       x="Date",
       y="Fatality Cases") +
  scale_color_manual(name="", values = c("red","blue")) +
  scale_fill_manual(name="", values=c("red","blue"))+
  ggtitle("Actual vs Prediction For Italy Fatality cases")
```



Now we can re-assign training Targeet Values to their original known values

```
#################### Set training Target Values to their original known values
Predicted_C[Prd_Dt_Msk_C,]$q_0.5 <-TargetVal_C_trained
Predicted_C[Prd_Dt_Msk_C,]$q_0.05 <-TargetVal_C_trained
Predicted_C[Prd_Dt_Msk_C,]$q_0.95 <-TargetVal_C_trained

Predicted_F[Prd_Dt_Msk_F,]$q_0.5 <-TargetVal_F_trained
Predicted_F[Prd_Dt_Msk_F,]$q_0.05 <-TargetVal_F_trained
Predicted_F[Prd_Dt_Msk_F,]$q_0.95 <-TargetVal_F_trained
```

Submit Results

```
# Submission function
Submit  <- function(test_c,test_f){
#test_c <-Predicted_C
#test_f <- Predicted_F
  # expected test_x to have "ForecastId", "q_0.05","q_0.5","q_0.95" columns
  test_full <- rbind.data.frame(test_c,test_f) %>% select(ForecastId,q_0.05,q_0.5,q_0.95) %>%
    "colnames<-"(c("Id", "0.05","0.5","0.95")) %>% arrange(Id)

  # prepare and join submission dataset
  sub_temp <- sub_Kag %>% separate(ForecastId_Quantile,c("Id","q"),"_")  %>%
    pivot_wider(names_from=q,values_from=TargetValue ) %>% select(Id) %>% mutate(Id=as.numeric(Id)) %>%
  left_join(test_full, by="Id") %>%
```

```
    pivot_longer("0.05":"0.95",names_to = "Quantile",values_to = "TargetValue") %>%
    mutate(Id=as.character(Id)) %>%unite("_", Id:Quantile) %>% rename("ForecastId_Quantile"="_")

  current_DT<-format(Sys.time(), "%Y-%m-%d_%H-%M")
  Subfile<- paste0("Output/submission",current_DT,".csv")
  write.csv(sub_temp,Subfile,row.names = FALSE)
}

#####################################################
Submit(select(Predicted_C,-Date),select(Predicted_F,-Date))
```

Shutdown H2o Server

```
# shutdown H2o Server
h2o.shutdown(prompt = TRUE)
```

```
## Are you sure you want to shutdown the H2O instance running at http://localhost:54321/ (Y/N)?
```

Conclusion

In conclusion, it is clear now that COVID confirmed cases and fatality cases can be modelled as time series regression where the date is the primary predictor. This is not the optimal way to model upgrowing cases however, due to lake of information about specific locations we reside on this Gradient Boosting Machine model. For better estimation, we will consider adding more predictors to the model using other datasets that may include lockdown period and social connection related to each location.