Conclusion

This project involved navigation of both R and Hadoop servers. Packets explored included rsparkling and h2o amongst others. Shiny application was used for initial data and network exploration. H2O platform was used to perform logistic modeling and deep learning.

A future extension of this project can include building data pipelines that can dynamically extract weather data for the associated flights and store the data in a hadoop database. Since weather appears to be a strong indicator of departure delays, this could ensure its accountability in the flights data. Additionally data containing security alerts can be used to gauge whether flight delays have been associated with increased alerts.

If we don't want Conclusion to have a chapter number next to it, we can add the `{.unnumbered}` attribute. This has an unintended consequence of the sections being labeled as 3.6 for example though instead of 4.1. The LaTeX commands immediately following the Conclusion declaration get things back on track.

More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

The First Appendix

There were no hidden code chunks in the paper. See the second appendix for access to accessory code.

The Second Appendix, for Fun

Creation of the FullDat dataset (flights data from 2008 to 2016)

```
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)

options(rsparkling.sparklingwater.version = "1.6.8")

sc <- spark_connect(master = "yarn-client") #connecting to the cluster.
#spark_disconnect(sc) disconnect to cluster

#loading in flights dataset from 2008 to 2016
flights2008 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2008
                              /part-m-*", header=FALSE, memory=FALSE)
flights2009 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2009
                              /part-m-*", header=FALSE, memory=FALSE)
flights2010 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2010
                              /part-m-*", header=FALSE, memory=FALSE)
flights2011 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2011
                              /part-m-*", header=FALSE, memory=FALSE)
flights2012 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2012
                              /part-m-*", header=FALSE, memory=FALSE)
flights2013 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2013
```

```r
                              /part-m-*", header=FALSE, memory=FALSE)
flights2014 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2014
                              /part-m-*", header=FALSE, memory=FALSE)
flights2015 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2015
                              /part-m-*", header=FALSE, memory=FALSE)
flights2016 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights/flights/2016
                              /part-m-*", header=FALSE, memory=FALSE)

#Columns were named
flights2008 <- flights2008 %>%
  rename(year = V1) %>%
  rename(month = V2) %>%
  rename(day = V3) %>%
  rename(dep_time = V4) %>%
  rename(sched_dep_time = V5) %>%
  rename(dep_delay = V6) %>%
  rename(arr_time = V7) %>%
  rename(sched_arr_time = V8) %>%
  rename(arr_delay = V9) %>%
  rename(carrier = V10) %>%
  rename(tailnum = V11) %>%
  rename(flight = V12) %>%
  rename(origin = V13) %>%
  rename(dest = V14) %>%
  rename(air_time = V15) %>%
  rename(distance = V16) %>%
  rename(hour = V18) %>%
  rename(minute = V19) %>%
  rename(time_hour = V20)


save(flights2008, file = "Flights08.Rda") #repeat this step for 2009-2016.

load("Flights08.Rda")
load("Flights09.Rda")
load("Flights10.Rda")
load("Flights11.Rda")
load("Flights12.Rda")
load("Flights13.Rda")
load("Flights14.Rda")
load("Flights15.Rda")
load("Flights16.Rda")

FinalDat <- rbind(DatNew08, DatNew09, DatNew10, DatNew11, DatNew12,
                  DatNew13, DatNew14, DatNew15, DatNew16)
save(FinalDat, file = "FinalDataYear.Rda")
```

H2O Logistic Regression

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)
```

```r
options(rsparkling.sparklingwater.version = "1.6.8")
sc <- spark_connect(master = "yarn-client")

log <- load("FullLogData.Rda")
datlog <- FullDatLog

DatNew <- FullDatLog
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"), "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)


data3$weekend <- as.factor(data3$weekend)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month, from = c("1", "2", "3", "4", "5", "6",
                                                     "7", "8", "9", "10", "11", "12"),
                               to = c("January", "February", "March", "April", "May",
                                      "June", "July", "August", "September",
                                      "October", "November", "December"))
data3$season <- ifelse(data3$month %in% c("March", "April", "May"), "spring",
                       ifelse(data3$month %in% c("June", "July", "August"), "summer",
                              ifelse(data3$month %in% c("September",
                                                         "October", "November"),
                                     "fall", "winter")))

data3$season <- as.factor(data3$season)
data3$year <- as.factor(data3$year)
FullDatLog <- data3

FullDatLog$year <- as.factor(FullDatLog$year)
FullDatLog$dep_delayIn = ifelse(FullDatLog$dep_delay > 30, "Yes", "No")
FullDatLog$dep_delayIn <- as.factor(FullDatLog$dep_delayIn)
FullDatLog1 <- FullDatLog[c(1,2,9,10,15,16,18,21,22,23,24)]

FullDatLog <- FullDatLog1
save(FullDatLog, file = "HadoopLogMod.Rda")

FullDatLog2 <- load("HadoopLogMod.Rda")
set.seed(134)
sampled <- FullDatLog[sample(nrow(FullDatLog), 200000, replace = FALSE, prob = NULL),]

mtcars_tbl <- copy_to(sc, sampled, "LogData", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
test <- as_h2o_frame(sc, partitions$test)
```

```r
training$dep_delayIn <- as.factor(training$dep_delayIn)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)


test$dep_delayIn <- as.factor(test$dep_delayIn)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
test$year <- as.factor(test$year)

myX = setdiff(colnames(training), c("dep_delayIn", "orig_id", "hour", "month", "weekend"))

regmod <- h2o.glm(y = "dep_delayIn", x = myX, training_frame = training, family = "binomial",
        alpha = 0.1, lambda_search = FALSE, nfolds = 5)

h2o.performance(regmod)
h2o.varimp(regmod)
h2o.varimp_plot(regmod, num_of_features = 20)
mat <- h2o.confusionMatrix(regmod)
(mat$No[1]+mat$Yes[2])/(mat$No[1]+mat$No[2]+mat$Yes[1]+mat$Yes[2]) #model accuracy

pred <- h2o.predict(object = regmod, newdata = test)
mean(pred$predict==test$dep_delayIn)
plot(h2o.performance(regmod))

#Weather Logistic Regression

flights$hour <- ifelse(flights$hour == 24, 0, flights$hour)
flights_weather <- left_join(flights, weather)
flights_weather$total <- flights_weather$dep_delay + flights_weather$arr_delay
flights_weather2 <- filter(flights_weather, total > 0)

DatNew <- flights_weather2
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"), "weekend","weekday")
DatNew$month <- as.factor(DatNew$month)
DatNew$month <- plyr::mapvalues(DatNew$month, from = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10
                               to = c("January", "February", "March", "April", "May",
                                      "June", "July", "August", "September",
                                      "October", "November", "December"))
DatNew$season <- ifelse(DatNew$month %in% c("March", "April", "May"), "spring",
                         ifelse(DatNew$month %in% c("June", "July", "August"), "summer",
                                ifelse(DatNew$month %in% c("September", "October", "November"), "fall",

flights_weather2 <- DatNew
save(flights_weather2, file = "flights_weather22.Rda")

load("flights_weather22.Rda")
head(flights_weather2)
names(flights_weather2)
nrow(flights_weather2)
flights2 <- na.omit(flights_weather2)

mtcars_tbl <- copy_to(sc, flights2, "flights", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
test <- as_h2o_frame(sc, partitions$test)

training$dep_delayIn <- ifelse(training$dep_delay > 30, "Yes", "No")
test$dep_delayIn <- ifelse(test$dep_delay > 30, "Yes", "No")

training$dep_delayIn <- as.factor(training$dep_delayIn)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)
training$day <- as.factor(training$day)


test$dep_delayIn <- as.factor(test$dep_delayIn)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
test$year <- as.factor(test$year)
test$day <- as.factor(test$day)

testDat <- test[,c(1,2,3,9,10,15,16,17,20,21,22,23,24,25,26,27,28,30,31,32,33)]
trainDat <- training[,c(1,2,3,9,10,15,16,17,20,21,22,23,24,25,26,27,28,30,31,32,33)]

myX = setdiff(colnames(testDat), c("dep_delayIn"))
regmod <- h2o.glm(y = "dep_delayIn", x = myX, training_frame = trainDat,
                  family = "binomial",
                  alpha = 0.1, lambda_search = FALSE, nfolds = 5)

regmodWeather <- regmod
h2o.performance(regmodWeather)
```

```r
h2o.varimp(regmod)
h2o.varimp_plot(regmodWeather, num_of_features = 30)
h2o.confusionMatrix(regmodWeather)
(mat$No[1]+mat$Yes[2])/(mat$No[1]+mat$No[2]+mat$Yes[1]+mat$Yes[2])

pred <- h2o.predict(object = regmodWeather, newdata = testDat)
mean(pred$predict==testDat$dep_delayIn) #accuracy of test set
```

H2O Deep Learning

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)


options(rsparkling.sparklingwater.version = "1.6.8")
#spark_disconnect(sc)
sc <- spark_connect(master = "yarn-client")

yas <- load("FinalDataYear.Rda")
head(FinalDat)
nrow(FinalDat)
unique(FinalDat$year)

DatNew <- FinalDat
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)


data3$weekend <- as.factor(data3$weekend)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month, from = c("1", "2", "3",
                                                     "4", "5", "6",
                                                     "7", "8", "9",
                                                     "10", "11", "12"),
                               to = c("January", "February", "March",
                                      "April", "May", "June", "July",
                                      "August", "September",
                                      "October", "November", "December"))
data3$season <- ifelse(data3$month %in% c("March", "April", "May"), "spring",
                       ifelse(data3$month %in% c("June", "July", "August"),
                              "summer",
                              ifelse(data3$month %in% c("September", "October",
                                                        "November"),
                                    "fall", "winter")))

data3$season <- as.factor(data3$season)
data3$year <- as.factor(data3$year)
FullDatLog <- data3
FullDatLog$year <- as.factor(FullDatLog$year)
FullDatLog$dep_delay <- as.numeric(FullDatLog$dep_delay)
nrow(FullDatLog)

set.seed(12)
sampled <- FullDatLog[sample(nrow(FullDatLog),
                              200000, replace = FALSE, prob = NULL),]
mtcars_tbl <- copy_to(sc, sampled, "deep", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.5, validation = 0.25, test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
validation <- as_h2o_frame(sc, partitions$validation)
test <- as_h2o_frame(sc, partitions$test)

training$dep_delay <- as.numeric(training$dep_delay)
training$arr_delay <- as.numeric(training$arr_delay)
training$air_time <- as.numeric(training$air_time)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)

validation$dep_delay <- as.numeric(validation$dep_delay)
validation$arr_delay <- as.numeric(validation$arr_delay)
validation$air_time <- as.numeric(validation$air_time)
validation$season <- as.factor(validation$season)
validation$week <- as.factor(validation$week)
validation$weekend <- as.factor(validation$weekend)
validation$carrier <- as.factor(validation$carrier)
validation$hour <- as.factor(validation$hour)
validation$month <- as.factor(validation$month)
validation$year <- as.factor(validation$year)

test$dep_delay <- as.numeric(test$dep_delay)
test$arr_delay <- as.numeric(test$arr_delay)
test$air_time <- as.numeric(test$air_time)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
test$year <- as.factor(test$year)
```

```r
training <- training[,c(1,2,6,9,10,15,16,18,21,22,23)]
test <- test[,c(1,2,6,9,10,15,16,18,21,22,23)]
validation <- validation[,c(1,2,6,9,10,15,16,18,21,22,23)]

myX = setdiff(colnames(training), ("dep_delay"))

#First simplified deep learning model
m1 <- h2o.deeplearning(
  y="dep_delay",
  x=myX,
  activation="Tanh",
  training_frame=training,
  validation_frame=validation,
  epochs=1,
  variable_importances=T,
  nfolds = 5,
  keep_cross_validation_predictions=T
)

deepmod <- m1
h2o.performance(deepmod, train = TRUE)
h2o.performance(deepmod, valid = TRUE)
h2o.performance(deepmod, newdata = test)
h2o.varimp_plot(deepmod, num_of_features = 20)

#Grid search model iteration
hyper_params <- list(
  activation=c("Tanh", "TanhWithDropout"),
  hidden=list(c(20,20),c(40,40)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03)
)

grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id="gridDeep",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  momentum_start=0.5,
  momentum_stable=0.9,
  momentum_ramp=1e7,
  variable_importances=T,
  l1=1e-5,
  l2=1e-5,
  max_w2=10,
  hyper_params=hyper_params
)

for (model_id in grid@model_ids) { #print model MSE
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  print(sprintf("Validation set MSE: %f", mse))
}
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE

h2o.varimp_plot(optimal, num_of_features = 20)

#Random grid search model
hyper_params <- list(
  activation=c("Tanh","TanhWithDropout"),
  hidden=list(c(20,20),c(30,30,30),c(40,40,40),c(50,50),c(70,70)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6)
)

search_criteria = list(strategy = "RandomDiscrete",
                       max_runtime_secs = 600, max_models = 100,
                       seed=22, stopping_rounds=5,
                       stopping_tolerance=2e-2)

random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "Gridrandom6_model_6",
  training_frame=training,
  validation_frame=validation,
  x=myX,
  y="dep_delay",
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_validation_samples=10000,
  score_duty_cycle=0.025,
  max_w2=10,
  hyper_params = hyper_params,
  search_criteria = search_criteria
)

for (model_id in grid@model_ids) {
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  sprintf("Validation set MSE: %f", mse)
}

#Retrieve optimal model by MSE
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE

h2o.varimp_plot(optimal, num_of_features = 20)

DeepMod <- h2o.saveModel(optimal, path = "/home/ajavaid17", force = FALSE)
randomModel <- h2o.loadModel(path = "/home/ajavaid17/DeepLearning_model_R_1487567612904_2")

#Checkpoint continuation from random model
max_epochs <- 20
checkpoint <- h2o.deeplearning(
  model_id="GridModRandom_continued2",
  activation="Tanh",
  checkpoint="Gridrandom6_model_6",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  hidden=c(30,30,30),
  epochs=max_epochs,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  l1=1e-4,
  l2=1e-4,
  max_w2=10,
  rate = 0.02,
  variable_importances=T
)

spark_disconnect(sc)
h2o.shutdown(prompt=FALSE)
```

Shiny Application Code

```
library(shiny)
library(dplyr)
library(mosaic)
library(base)
library(plotly)
library(ggplot2)
```

```r
library(nycflights13)
library(lubridate)
library(igraph)
require(visNetwork)

options(shiny.sanitize.errors = TRUE)

data(flights)
data(weather)

dat <- load("airportFullDat.Rda")
datcar <- load("carrierData.Rda")
dat2 <- load("FinalDataYear.Rda")
map <- load("USMap.Rda")

nrow(FinalDat)
airportsVis <- read.csv("http://datasets.flowingdata.com/tuts/
                        maparcs/airports.csv", header = TRUE)
data2 <- stateFin2
DatNew <- FinalDat


DatNew$time_hour <- as.POSIXct(DatNew$time_hour)
DatNew$hour <- hour(DatNew$time_hour)
#DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)
carrierDat$codeCar <- as.character(carrierDat$codeCar)

data3 <- carrierDat %>% inner_join(data3,
                                   by = c("codeCar" = "carrier"))
data3 <- plyr::rename(data3, c("code" = "carrier"))

data3$weekend <- as.factor(data3$weekend)
data2$OriginAirport <- as.factor(data2$OriginAirport)
data2$DestAirport <- as.factor(data2$DestAirport)
data2$OriginState <- as.factor(data2$OriginState)
data2$DestState <- as.factor(data2$DestState)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month,
from = c("1", "2", "3", "4", "5", "6", "7", "8",
         "9", "10", "11", "12"),
                               to = c("January",
                                      "February",
                                      "March",
                                      "April",
                                      "May",
                                      "June",
                                      "July",
                                      "August",
                                      "September",
                                      "October",
                                      "November",
                                      "December"))
data3$season <- ifelse(data3$month %in% c("March",
                                          "April",
                                          "May"), "spring",
                       ifelse(data3$month %in% c("June",
                                                 "July",
                                                 "August"),
                              "summer",
                              ifelse(data3$month %in%
                                       c("September",
                                         "October",
                                         "November"),
                                     "fall", "winter")))

data3$season <- as.factor(data3$season)
namesDat3 <- data3[,c(2,4,19,22,23,24)]
namesDat3$month <- as.factor(namesDat3$month)
namesDat3$carrier <- as.factor(namesDat3$carrier)
namesDat3$week <- as.factor(namesDat3$week)
namesDat3$weekend <- as.factor(namesDat3$weekend)
namesDat4 <- namesDat3[,c(2, 4, 5, 6)]


#Flights Dataset Analysis
flights$hour <- ifelse(flights$hour == 24, 0, flights$hour)
flights_weather <- left_join(flights, weather)
flights_weather$total <- flights_weather$dep_delay +
  flights_weather$arr_delay
flights_weather2 <- filter(flights_weather, total > 0)

DatNew <- flights_weather2
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in%
      c("Saturday", "Sunday"), "weekend","weekday")
DatNew$month <- as.factor(DatNew$month)
DatNew$month <- plyr::mapvalues(DatNew$month,
  from = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"),
                                to = c("January",
                                       "February",
                                       "March",
                                       "April",
                                       "May",
                                       "June",
                                       "July",
                                       "August",
                                       "September",
                                       "October",
                                       "November",
                                       "December"))
DatNew$season <- ifelse(DatNew$month %in% c("March",
                                            "April",
                                            "May"), "spring",
                        ifelse(DatNew$month %in% c("June",
                                                   "July",
                                                   "August"),
                               "summer",
                               ifelse(DatNew$month %in%
                                        c("September",
                                          "October",
                                          "November"),
                                      "fall", "winter")))

flights_weather2 <- DatNew
flights_weather3 <- flights_weather2[,c(20, 21,
                                        22, 23, 24,
                                        25, 26, 27,
                                        28)]



ui <- navbarPage("Flights Analysis",
                 tabPanel("Graphical",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go",
                            "Graphical Flights Comparison"),

                              selectInput("response",
                                "Choose a response predictor:",
                              choices = names(namesDat3)),

                              selectInput("yearInitial",
                              "Choose year range (initial):",
                            choices = sort(unique(data3$year))),

                              selectInput("yearEnd",
                                "Choose year range (end):",
                              choices = sort(unique(data3$year))),
                              strong("Choose a
response predictor to visualize grouped changes in
mean departure delay over time from 2008 to 2016.")
                            ),
                            mainPanel(
                              plotOutput('plot', height = "900px")
                            )
                          )
                 ),
                 tabPanel("Table Summary",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go1",
                              "Airport Flights Data"),

                              actionButton("go2",
                              "State Flights Data"),


                              selectInput("origin",
                              "Choose a origin airport:",
              choices = sort(unique(data2$OriginAirport))),

    selectInput("destination", "Choose a destination airport:",
          choices = sort(unique(data2$DestAirport))),

      selectInput("originState", "Choose a origin state:",
          choices = sort(unique(data2$OriginFState))),

  selectInput("destState", "Choose a destination state:",
          choices = sort(unique(data2$DestFState))),
  strong("Choose a origin and destination airport to
see the mean departure delay in the data table.
Alternatively, origin and destination states can
also be specified.")),
      mainPanel(
    dataTableOutput("view"),
    dataTableOutput("view2")
    )
      )
       ),
  tabPanel("2013 Weather & Flights Analysis",
  sidebarLayout(
      sidebarPanel(
    actionButton("go3", "Weather and Flight Delays"),

   selectInput("response2", "Choose a response predictor:",
          choices = names(namesDat4)),

          selectInput("weatherDis", "Choose a weather phenomena:",
          choices = names(flights_weather3)),
strong("Choose a response predictor and weather
occurrence from the dropdown to visualize changes
in total delay (departure delay plus arrival delay)
by the response variable chosen grouped by
weather event for LaGuardia, John F. Kennedy and
Newark Liberty International Airport in 2013.")
                            ),
              mainPanel(
              plotOutput('plot2', height = "900px")
                            )
                          )
                 ),
                 tabPanel("Flights Network Analysis",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go5",
                              "Sample 500 Flights"),
                              strong("Click the button above to
generate a network of 500 flights visualizing departure delay greater than 90 minutes.
The network shows flights by airports (vertices). The width of the edge segment shows
the extent of departure delay between origin and destined flights. Airport
can also be selected from the dropdown list (id). A data table is also presented
which lists departure delay for the queried origin and destination airports.")
                            ),
                            mainPanel(
                              visNetworkOutput("network", height = "400px"),
                              #plotlyOutput('plot4', height = "900px"),
                              dataTableOutput("view3")
                            ))
                 ),
                 tabPanel("Maps & Flights Analysis",
                          sidebarLayout(
                            sidebarPanel(
                              strong("The map shows the average departure delay
for United States airports from 2008 to 2016.Hovering over the point will reveal
information about the average departure delay for the
                                state and airport selected.")
                            ),
                            mainPanel(
                              plotlyOutput('plot3')
                            )
                          )
                 )
)


server <- shinyServer(function(input, output) {

  df_subset <- eventReactive(input$go1, {
    a <- data2 %>% filter(OriginAirport == input$origin & DestAirport == input$destination)
    return(a)
  })

  df_subsetVis <- eventReactive(input$go5, {
    dat1 <- FinalDat[sample(nrow(FinalDat), 500, replace = FALSE, prob = NULL),]
    dat2 <- dat1[,c(1, 13, 14, 6)]
    dat3 <- suppressWarnings(inner_join(dat2, airportsVis, by = c("origin" = "iata")))
    dat4 <- suppressWarnings(inner_join(dat3, airportsVis, by = c("dest" = "iata")))
    dat5 <- dat4[,c(5, 11, 4)]
    dat5$airport.x <- sort(dat5$airport.x)
    dat5$airport.y <- sort(dat5$airport.y)
    dat5 <- plyr::rename(dat5, c("airport.x" = "OriginAirport"))
    dat5 <- plyr::rename(dat5, c("airport.y" = "DestinationAirport"))
```

```r
  dat5 <- plyr::rename(dat5, c("dep_delay" = "DepartureDelay"))
  return(dat5)
})


df_subset2 <- eventReactive(input$go2, {
  b <- data2 %>% filter(OriginFState == input$originState & DestFState
                        == input$destState) %>% arrange(desc(meanDelay))
  return(b)
})

df_weekend <- eventReactive(input$go, {
  DatCarrier <- data3 %>% filter(year >= input$yearInitial & year <= input$yearEnd) %>%
    mutate(yearF = as.factor(year)) %>%
    group_by_("yearF", input$response) %>%
    summarise(MeanDep = mean(dep_delay))
  return(DatCarrier)
})

df_weather <- eventReactive(input$go3, {
  return(flights_weather2)
})

output$plot <- renderPlot({
  dfweek <-  df_weekend()
  p = ggplot(dfweek, aes_string(x = "yearF", y = "MeanDep",
                                group = input$response, color = input$response)) +
    geom_point() + geom_line() +
    ggtitle(paste("Mean departure delay overtime by", input$response))
  print(p)
  #ggplotly(p)
})

output$plot2 <- renderPlot({
  dfweek2 <-  df_weather()
  p1 = ggplot(dfweek2, aes_string(x = input$weatherDis, y = "total",
                                  group = input$response2, color = input$response2)) +
    geom_smooth()  + ggtitle(paste("Total Delay in" , input$weatherDis, "in 2013"))
  print(p1)
  #ggplotly(p1) alternatively use interactive plotting
})

output$plot3 <- renderPlotly({
  g <- list(
    scope = 'usa',
    projection = list(type = 'albers usa'),
    showland = TRUE,
    landcolor = toRGB("gray85"),
    subunitwidth = 1,
    countrywidth = 1,
    subunitcolor = toRGB("white"),
    countrycolor = toRGB("white")
  )
```

```
    USAirSum3$MeanDepDelay <- round(USAirSum3$MeanDepDelay, 2)
    USAirSum3$OriginFState <- (tools::toTitleCase(USAirSum3$OriginFState))
    p <- plot_geo(USAirSum3, locationmode = 'USA-states', sizes = c(1, 250)) %>%
      add_markers(
        x = ~longitude, y = ~latitude, color = ~MeanDepDelay, hoverinfo = "text",alpha= 0.8,
        text = ~paste(OriginFState, "<br />", MeanDepDelay, "<br />", OriginAirport)
      ) %>%
      layout(title = 'Mean Departure Delay by Airport from 2008-2016', geo = g)
    ggplotly(p)

  })

  output$network <- renderVisNetwork({
    dfweek2 <-  df_subsetVis()
    e2 <- dfweek2
    g2=graph.data.frame(e2)
    E(g2)$width <- dfweek2$DepartureDelay/7
    visIgraph(g2) %>% visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE)
  })
  output$view <- renderDataTable(df_subset())

  output$view2 <- renderDataTable(df_subset2())

  output$view3 <- renderDataTable((df_subsetVis()))

})

options(shiny.sanitize.errors = TRUE)
shinyApp(ui = ui, server = server)
```