7/27/2018

Project Report

BATCH 39 - PHD

Kunal Ray
INSOFE

Problem Description

Air travel is becoming increasingly complex with multiple variables impacting the same. The flight delay is one of such variables that impacts carrier, Airport and passenger and may result in significant commercial loss or reputation loss to all the stakeholders and thus huge cost on the economy. Thus, prediction of delay is crucial not only from view point of customer from time management perspective and carrier for retention of customer faith but also from Airport point of view for managing the traffic more efficiently to optimize the number of arriving flights by appropriate adjustment of schedules. The contribution of weather conditions has been identified to be the very important contributor of these delays.

It is of the utmost interest of one of the participants of ecosystem to predict flight delays based on the flights details and predicted weather conditions by a good predictive model, which you are going to build, to take the necessary corrective and preventive actions to improve business
as well as service.

Data Provided

The historical data containing scheduled departure and arrival times, date, origin, destination and weather data is available, and the data scientists can predict if delay can happen or not using the flight data and aviation weather data for a specific flight.

Every single flight is observed as per their scheduled departure and arrival timestamps, to record the details of trips made, traffic conditions, etc. Flight details like Origin, destination, date of flight, scheduled departure and scheduled arrival time stamps etc. Weather stations data details like station id along with its linked AirportID, ground height etc. Hourly aviation weather conditions data also provided for 2 years etc. Origin, Destination details in flight data can be mapped to the AirportID in other datasets.

Train.csv: Flight Trip Details		
Attribute Name	**Data Type**	**Description**
FlightNumber	ID	ID
Year	Date	Travel Year
Month	Date	Travel Month
DayofMonth	Date	Travel Date – day of Travel Month
DayofWeek	Categorical	Day of week (1 to 7)
ScheduledDepTime	Time Stamp	Scheduled flight departure time at Origin
ScheduledArrTime	Time Stamp	Scheduled flight arrival time at Destination
ScheduledTravelTime	Integer	Scheduled travel time in minutes
Origin	ID	Flight starting location code
Destination	ID	Flight destination location code
Distance	Integer	Distance in miles b/w origin & destination
ActualArrivalTimeStamp	Date Time	Flight Actual Arrival TimeStamp

Weather Data (*hourly.txt)		
Attribute Name	**Data Type**	**Description**
WeatherStationID	ID	ID
YearMonthDay	Date	Date
Time	Time	Reported as Local Standard Time
SkyConditions	Categorical	SkyConditions
Visibility	Numeric	Visibility measured in Statuet Miles (SM)
DBT	Numeric	Dry Bulb Temperature – reported in deg C
DewPointTemp	Numeric	In deg C
RelativeHumidityPercent	Numeric	In %age
WindSpeed	Numeric	Reported in knots
WindDirection	Numeric	Direction from which wind blows. Reported to nearest degree, from true north
WindGustValue	Numeric	Maximum 5-second peak wind speed measured
StationPressure	Numeric	The pressure felt at that station or spot, but not adjusted to an equivalent at sea level

Precipitation Data (*hpd.txt)		
Attribute Name	**Data Type**	**Description**
WeatherStationID	ID	ID
YearMonthDay	Date	
Time	Time Stamp	Reported as Local Standard Time
HourlyPrecip	Numeric	Precipitation to measure rain fall amount (in mm)

AllStationsData_PHD.txt		
Attribute Name	**Data Type**	**Description**
WeatherStationID	ID	ID
AirportID	ID	ID
GroundHeight	Numeric	GroundHeight
StationHeight	Numeric	StationHeight
BarometerHeight	Numeric	BarometerHeight
Latitude	Numeric	Measure for precise location
Longitude	Numeric	Measure for precise location
TimeZone	Categorical	TimeZone

Approach & Data Consolidation

In order to solve the afore – mentioned problem, disparate data sources need to be aggregated separately and then combined. Pre – processing has to be done multiple times and new features related to the industry need to be generated from existing ones.

A few additional openly available variables from verified sources could be added to improve the accuracy further.

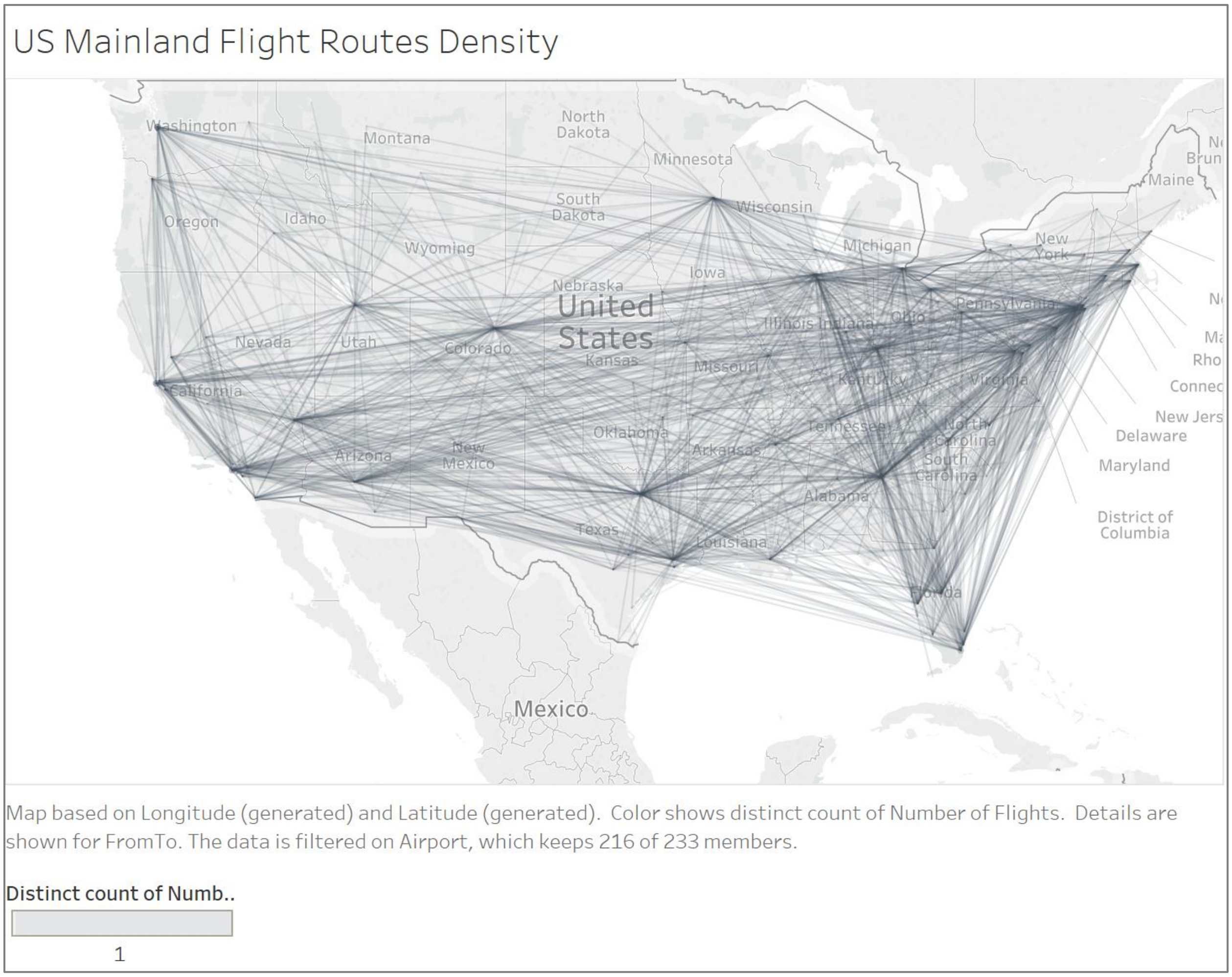


Fig1: Flight routes across all data rows from Train.csv

Step 1: Finding list of Weather Stations which are closest to each other.

To start with, load the file "*AllStationData_PHD.txt*" into R. There are various information points which together uniquely identify any weather station. Calculate the distance between each Weather Station & Every Other Weather Station based on Latitude, Longitude, Ground Height, Station Height, Barometer Height & Time Zone.

After individual distances have been calculated, order them in ascending order and pick top 5 and add these Weather Station IDs as the 5 closest Weather Station IDs. These 5 closest Weather Stations shall be used in the next step for imputing missing values.

For this exercise, R packages 'sp' and 'rgeos' was used. First closest, 2nd closest, 3rd closest etc. stations were identified separately and then merged with the Weather Station data frame.

Results are saved in '*closestation.rds*' and code chunk can be found in '*StationData.r*'.

```
library(sp)
library(rgeos)

stationdata$WeatherStationID<-as.factor(stationdata$WeatherStationID)
stationdata$GroundHeight<-as.numeric(stationdata$GroundHeight)
stationdata$StationHeight<-as.numeric(stationdata$StationHeight)
stationdata$BarometerHeight<-as.numeric(stationdata$BarometerHeight)
stationdata$TimeZone<-as.numeric(stationdata$TimeZone)

sp.stndata <- stationdata
coordinates(sp.stndata) <- ~GroundHeight+StationHeight+BarometerHeight+Latitude+Longitude+TimeZone

class(sp.stndata)

d <- gDistance(sp.stndata, byid=T)

# Finding closest match

min.d <- apply(d, 1, function(x) order(x, decreasing=F)[2])

newdata1 <- cbind(stationdata, stationdata[min.d,], apply(d, 1, function(x) sort(x,decreasing=F)[2]))

colnames(newdata1) <- c(colnames(stationdata), 'ClosestWS', 'n.AirportID', 'n.GroundHeight',
                        'n.StationHeight', 'n.BarometerHeight', 'n.Latitude', 'n.Longitude',
                        'n.TimeZone','distance')
```

Fig 2: Distance calculation b/w Weather Stations & Finding Closest Matches

Step 2: Loading & Imputing Precipitation Data

HPD data is loaded from files '*2004*.txt*' & '*2005*.txt*' into R environment. Using R's 'lubridate' package, the dates are converted to required formats and changed to factor. Minute level time data is converted to two – hourly time slots thus giving 12 slots per day.

After that, precipitation is aggregated by Weather Station, Year Month Day, Time Slot to give an average value and remove any minute level NULL values.

At this point, the existing data frame is merged with the file '*closestation.rds*' which is derived in the previous step. If the precipitation information is missing for any Weather Station, Date & Time Slot combination, it is attempted to be filled by the mean values from the 5 closest Weather Stations.

The same step is followed for all months separately for Train as well as for Test Data months. Post this, the consolidated precipitation data is saved externally in the file '*hpd.rds*' and '*hpdTest.rds*' for Train & Test respectively. The relevant code files are as follows:

Month	**Train**	**Test**
January	hpd01.R	N/A
March	hpd03.R	hpd03Test.R
May	hpd05.R	N/A
July	hpd07.R	hpd07Test.R
September	hpd09.R	hpd09Test.R
November	hpd11.R	hpd11Test.R
Consolidated	hpd.R	hpdTest.R

```r
# Merging with Closest Weather Stations

rm(hpd200401) # Free up memory
temp<-hpd0401
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "OrigPrecip"
hpd0401<-sqldf('select a.*, b.AvgPrecip from hpd0401 a left join (select Key0, AvgPrecip from temp) b
           on a.Key1=b.Key0')
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "ClosestPrecip"
gc()
hpd0401<-sqldf('select a.*, b.AvgPrecip from hpd0401 a left join (select Key0, AvgPrecip from temp) b
           on a.Key2=b.Key0')
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "Closest2ndPrecip"
gc()
hpd0401<-sqldf('select a.*, b.AvgPrecip from hpd0401 a left join (select Key0, AvgPrecip from temp) b
           on a.Key3=b.Key0')
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "Closest3rdPrecip"
gc()
hpd0401<-sqldf('select a.*, b.AvgPrecip from hpd0401 a left join (select Key0, AvgPrecip from temp) b
           on a.Key4=b.Key0')
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "Closest4thPrecip"
gc()
hpd0401<-sqldf('select a.*, b.AvgPrecip from hpd0401 a left join (select Key0, AvgPrecip from temp) b
           on a.Key5=b.Key0')
names(hpd0401)[names(hpd0401) == "AvgPrecip"] = "Closest5thPrecip"
gc()

# Check for null values & impute using closest neighbours

rm(temp) # Remove temp file
rm(closestation) # Remove unrequired file

colSums(is.na(hpd0401)) # NA values in OrigPrecip column

hpd0401$OrigPrecip<-ifelse(is.na(hpd0401$OrigPrecip),rowMeans(hpd0401[,c("ClosestPrecip", "Closest2ndPrecip",
                                                              "Closest3rdPrecip","Closest4thPrecip",
                                                              "Closest5thPrecip")], na.rm=TRUE),
                       hpd0401$OrigPrecip)
```

Fig 3: Aggregation & Imputation of Hourly Precipitation Data

Step 3: Loading & Imputing Weather Data

HLY data for 2004 & 2005 is loaded via files '*2004*hourly.txt*' and '*2005*hourly.txt*' respectively. Following pre – processing steps are carried out on each hourly file:

1. Visibility column has a unit '*SM*' attached to it, making it factor. Units are first removed and then it is converted to numeric.
2. The column '*Sky Condition*' has several levels such as BKN*, CLR, OVC*, VV*, SCT* etc. Search on the web shows that these levels are codes for heights of cloud layers[1]. For example: BKN022 indicates a broken (over half the sky) cloud layer with its base at 2200 ft. The lowest BKN or OVC layer specifies the cloud ceiling. If the code is VV*, it means that clouds cannot be seen because of fog or heavy precipitation, so vertical visibility is given instead. Clear Sky means that there are no clouds below 12000 ft.
   To consolidate, if any OVC, BKN or VV cases are observed below 12000 ft. that height is considered as the cloud ceiling else 12000 ft. is taken as the cloud ceiling.

[1] https://en.wikipedia.org/wiki/METAR

```
# Splitting Sky Condition & Deriving the lowest ceiling height with Broken or Overcast
hly200401$SkyConditions<-as.character(hly200401$SkyConditions)
library(data.table)
setDT(hly200401)[, paste0("SC", 1:3) := tstrsplit(SkyConditions, " ")]
library(stringr)
hly200401$SC4<-ifelse(str_sub(hly200401$SC1,1,3)%in%c('BKN','OVC') | str_sub(hly200401$SC1,1,2) == 'VV',
                      as.integer(str_sub(hly200401$SC1,-2-1))*100,12000)
hly200401$SC5<-ifelse(str_sub(hly200401$SC2,1,3)%in%c('BKN','OVC') | str_sub(hly200401$SC2,1,2) == 'VV',
                      as.integer(str_sub(hly200401$SC2,-2-1))*100,12000)
hly200401$SC6<-ifelse(str_sub(hly200401$SC3,1,3)%in%c('BKN','OVC') | str_sub(hly200401$SC3,1,2) == 'VV',
                      as.integer(str_sub(hly200401$SC3,-2-1))*100,12000)

hly200401$SC4<-ifelse(is.na(hly200401$SC4),12000,hly200401$SC4)
hly200401$SC5<-ifelse(is.na(hly200401$SC5),12000,hly200401$SC5)
hly200401$SC6<-ifelse(is.na(hly200401$SC6),12000,hly200401$SC6)
```

Fig 4: Deriving cloud ceiling based on Sky Condition

3. The next step is deriving Time Slots after the Year Month Day information has been converted to date format using '*lubridate*'.
4. Post this, all weather data is aggregated by Weather Station, Year Month Day, Time Slot as average values to ensure removal of any minute level NULL values.
5. This is followed by merging with the file '*closestation.rds*' derived from Step #1 and Weather Data across same time slot, date and closest location(s) are merged with the original data frame. In case any weather information is missing for any of the station – date – time slot combinations, it can be imputed using the means of the closest 5 stations for the same station – date & time slot combinations.

Same steps are followed for each of the Train & Test months one by one and then consolidated. Following are the files which are used for this.

Month	Train	Test
January	hly01.R	N/A
March	hly03.R	hly03Test.R
May	hly05.R	N/A
July	hly07.R	hly07Test.R
September	hly09.R	hly09Test.R
November	hly11.R	hly11Test.R
Consolidated	hly.R	hlyTest.R

Step 4: Pre – processing Train & Test & Merging

Train & Test Files are loaded. Scheduled date is derived using Year, Month & Date. Scheduled arrival time is loaded and converted to timestamp. Similarly, actual arrival timestamp is obtained.

```
flight$ActualArrivalTimeStamp<-dmy_hm(flight$ActualArrivalTimeStamp)

# Deriving Target Variable

flight$Delay<-difftime(flight$ActualArrivalTimeStamp, flight$ScheduledArrivalTimeStamp,
                       units = "mins") # Calculating time difference

flight$FlightDelayStatus<-ifelse(flight$Delay>15,1,0) # Deriving target variable
flight$FlightDelayStatus<-as.factor(flight$FlightDelayStatus)
```

Fig 5: Derivation of Target (Dependent) Variable

The target variable is derived by finding the difference between the scheduled and actual arrival time stamps and selecting those cases where time difference is at least 15 minutes.

In the next step, Scheduled Departure & Scheduled Arrival Times are converted to two hourly time slots. Bureau of Transport Statistics (BTS) which provides the carrier, tail number and flight number information. This data is available openly for download[2] and is added to the train data frame.

Day of Week & Month are converted to factors and finally the precipitation and weather data from Step 2 & Step 3 are merged with Train & Test Data to derive a consolidated data structure stored in '*flight.rds*' & '*flighttest.rds*' respectively.

```
# Merging with flight data first by origin & then by destination

flight<-sqldf('select a.*, b.OrigSkyCond, b.OrigVis, b.OrigDBT, b.OrigDewPtTemp, b.OrigRelHumPerc,
              b.OrigWindSp, b.OrigWindDir, b.OrigWindGustVal, b.OrigStnPres from flight a left join
              (select distinct AirportID, YearMonthDay, TimeSlot, OrigSkyCond, OrigVis, OrigDBT,
              OrigDewPtTemp, OrigRelHumPerc, OrigWindSp, OrigWindDir, OrigWindGustVal, OrigStnPres
              from hly) b on a.Origin = b.AirportID and a.SchedDate = b.YearMonthDay
              and a.SchDepSlot = b.TimeSlot')

flight<-sqldf('select a.*, b.DestSkyCond, b.DestVis, b.DestDBT, b.DestDewPtTemp, b.DestRelHumPerc,
              b.DestWindSp, b.DestWindDir, b.DestWindGustVal, b.DestStnPres from flight a left join
              (select distinct AirportID, YearMonthDay, TimeSlot, OrigSkyCond as DestSkyCond, OrigVis as DestVis, OrigDBT as DestDBT,
              OrigDewPtTemp as DestDewPtTemp, OrigRelHumPerc as DestRelHumPerc, OrigWindSp as DestWindSp, OrigWindDir as DestWindDir,
              OrigWindGustVal as DestWindGustVal, OrigStnPres as DestStnPres
              from hly) b on a.Destination = b.AirportID and a.SchedDate = b.YearMonthDay
              and a.SchArrSlot = b.TimeSlot')
```

Fig 6: Merger of train data with weather information for origin & destination

Visualization

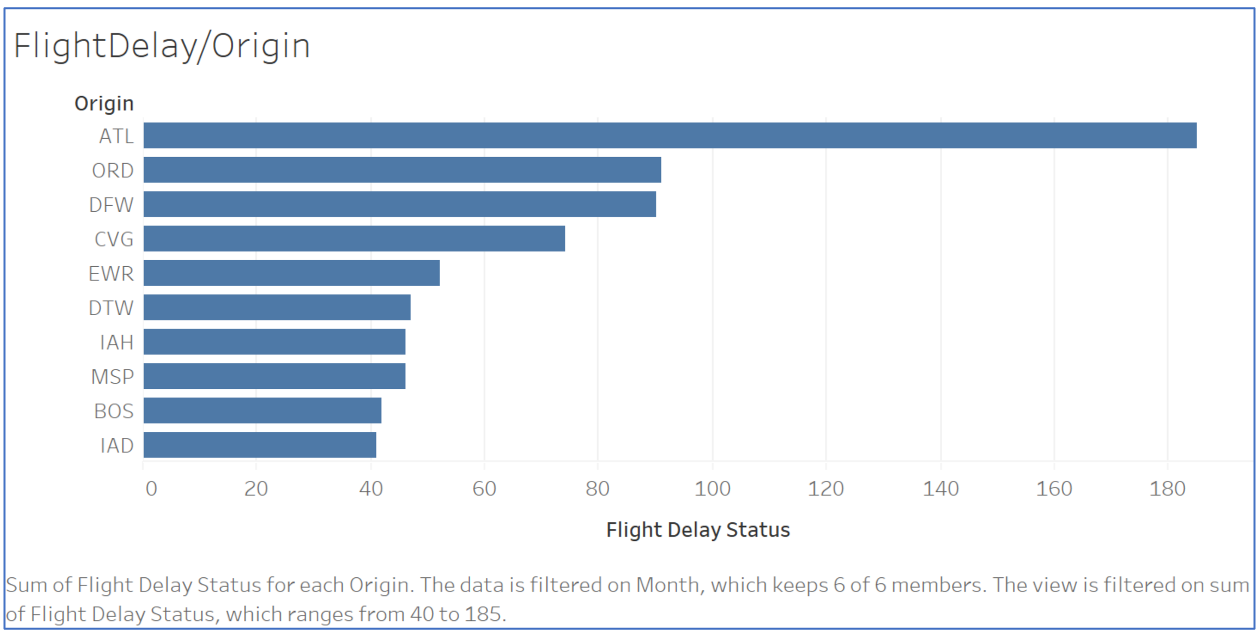


Fig 7: Instances of flight delay are observed more in some origin airports compared to others

[2] BTS Data can be found here: https://transtats.bts.gov/DL_SelectFields.asp **OR** https://drive.google.com/drive/folders/1_guRqrP__cYQk0IgXpdSruteMCANzfBO

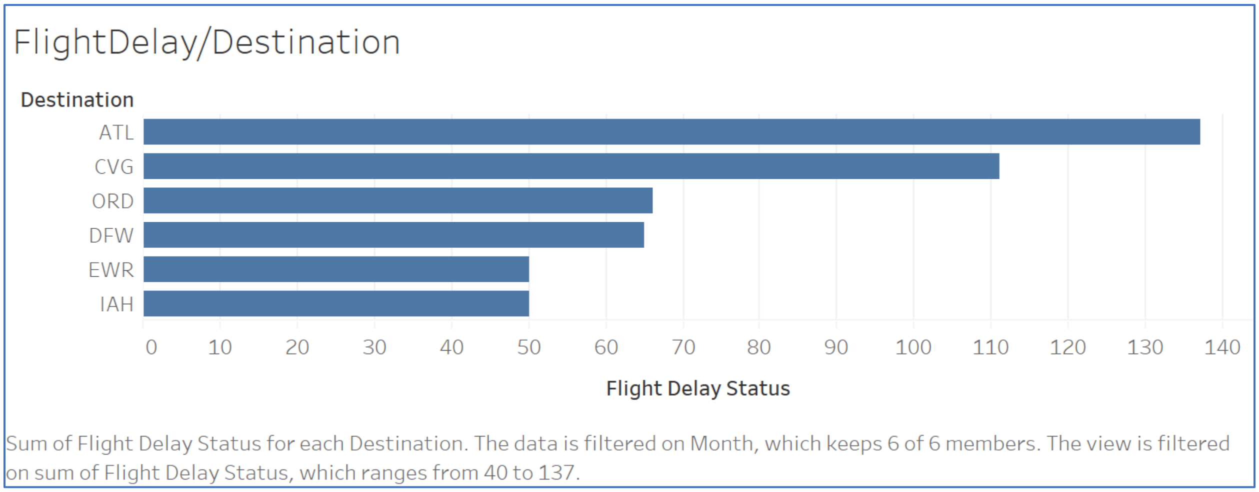


Fig: 8 Instances of flight delay are observed more in some destination airports compared to others

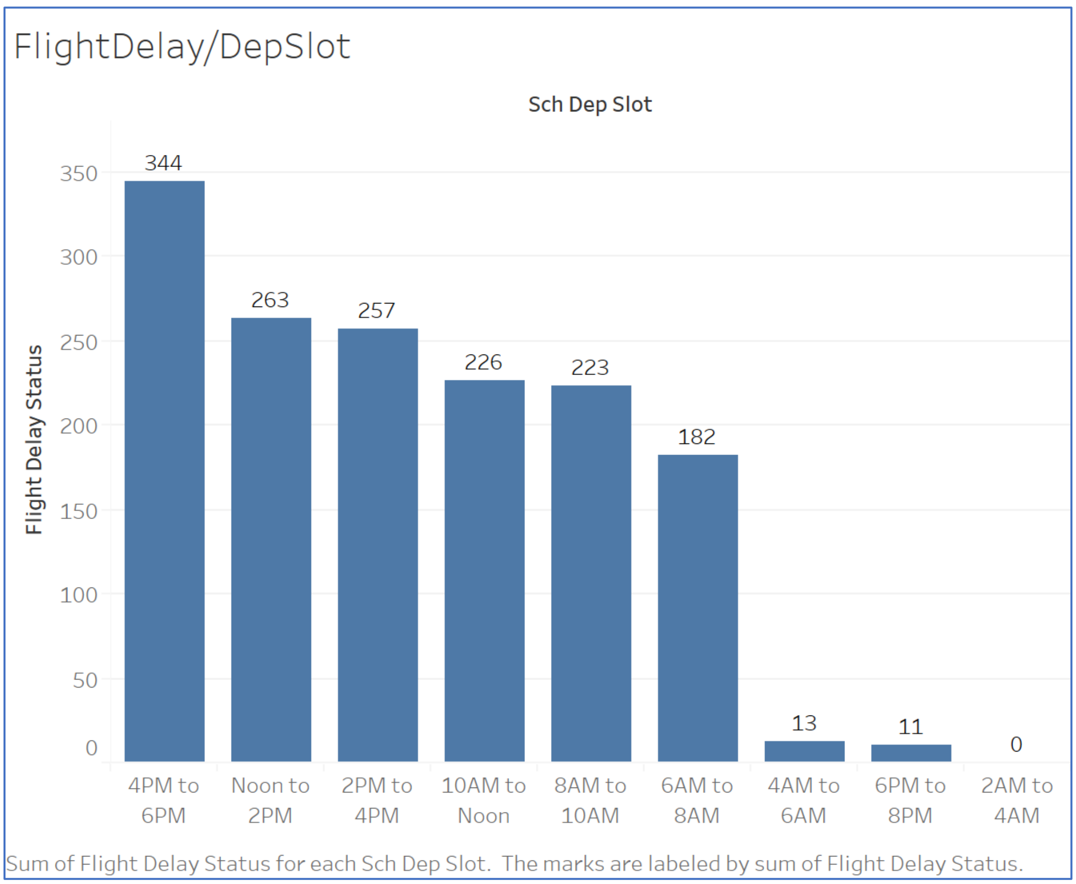


Fig 9: Flight delay is observed less when departure slot is b/w 4AM-6AM, 6PM-8PM or 2AM-4AM

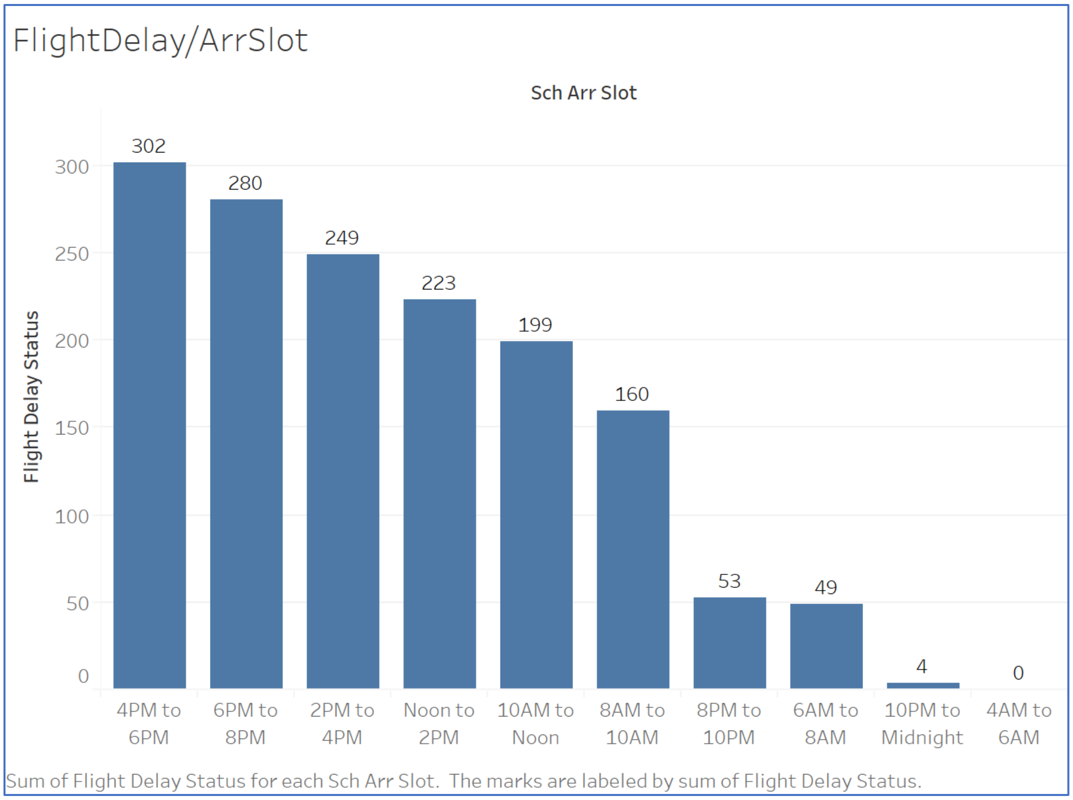


Fig 10: Flight delay is observed less when arrival slot is b/w 8PM-10PM, 6AM-8AM, 10PM-Midnight or 4AM-6AM

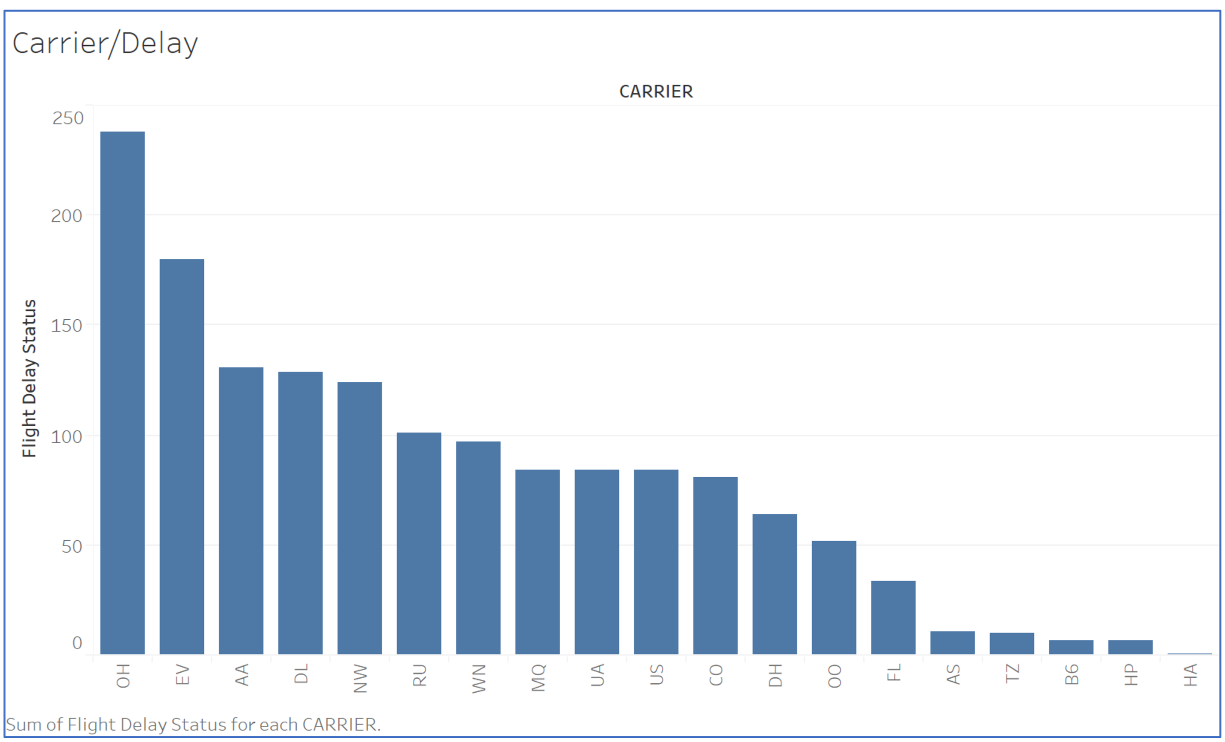
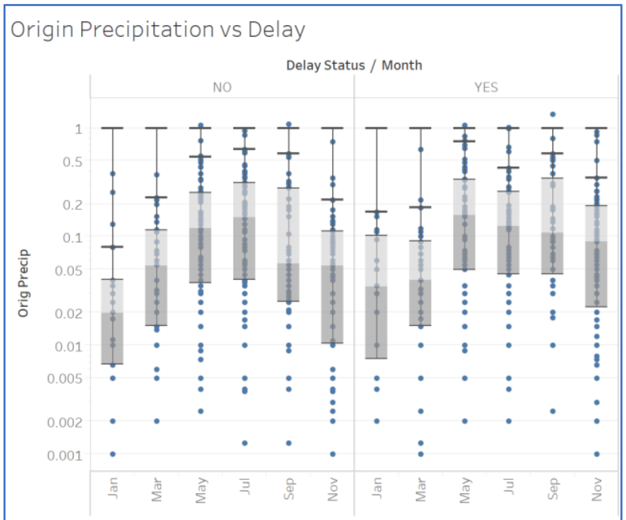
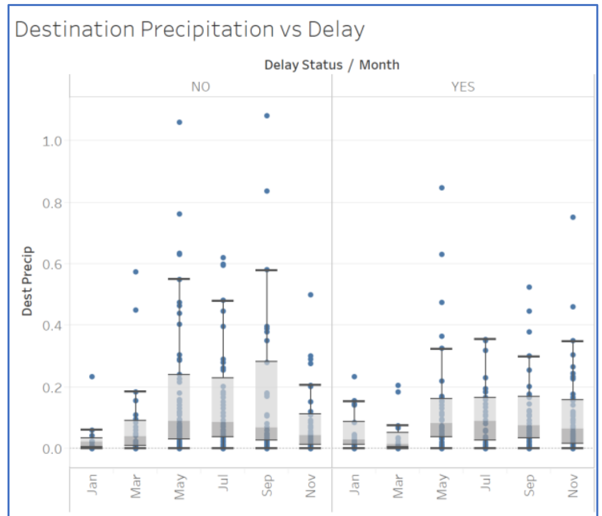


Fig 11: Some carriers are found to be more prone to delays than others

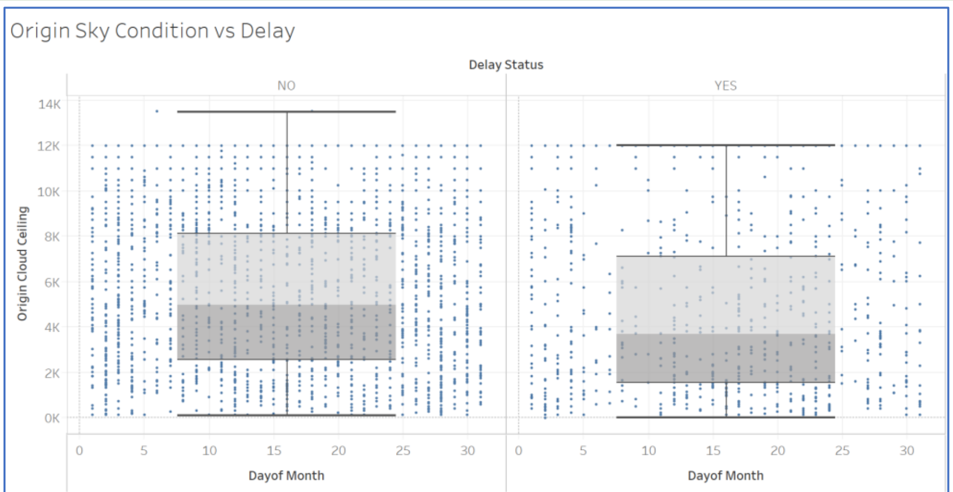
Visualization of Weather Data



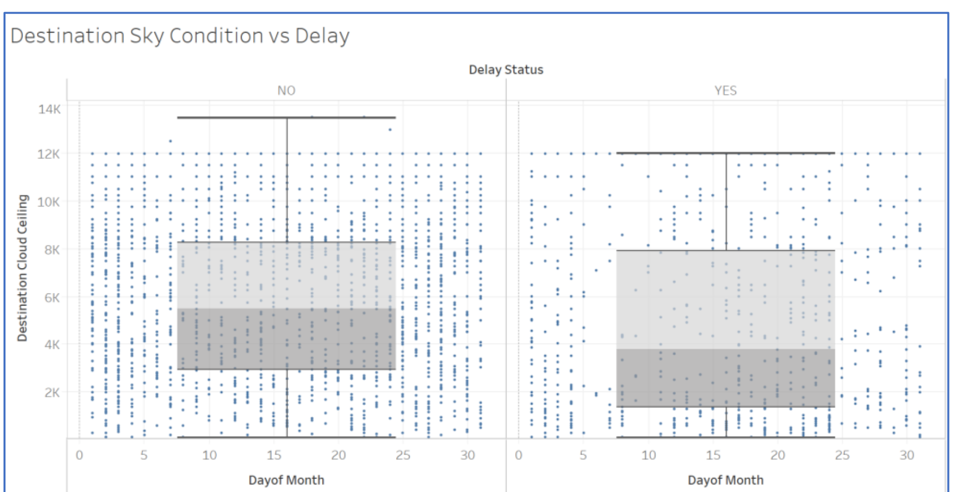
Orig Precip for each Month broken down by Delay Status. The view is filtered on Month, which keeps 6 of 6 members.



Dest Precip for each Month broken down by Delay Status. The view is filtered on Month, which keeps 6 of 6 members.

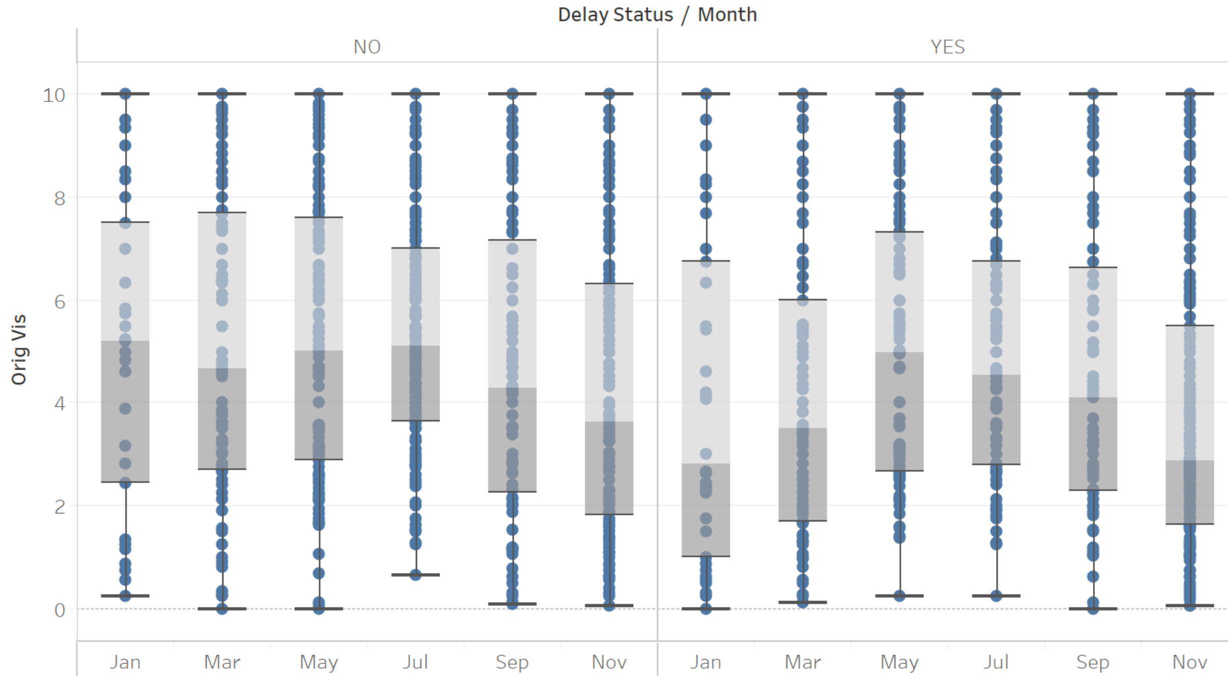


Dayof Month vs. Origin Cloud Ceiling broken down by Delay Status.



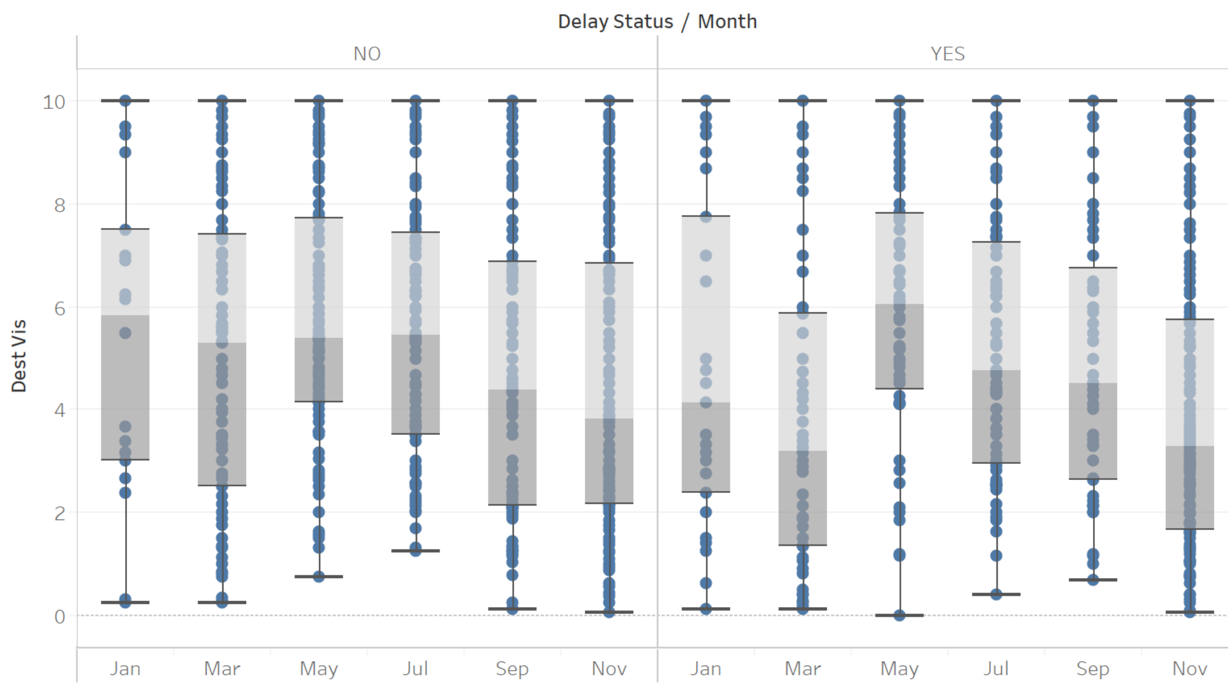
Dayof Month vs. Destination Cloud Ceiling broken down by Delay Status.

Visibility at Origin vs Delay



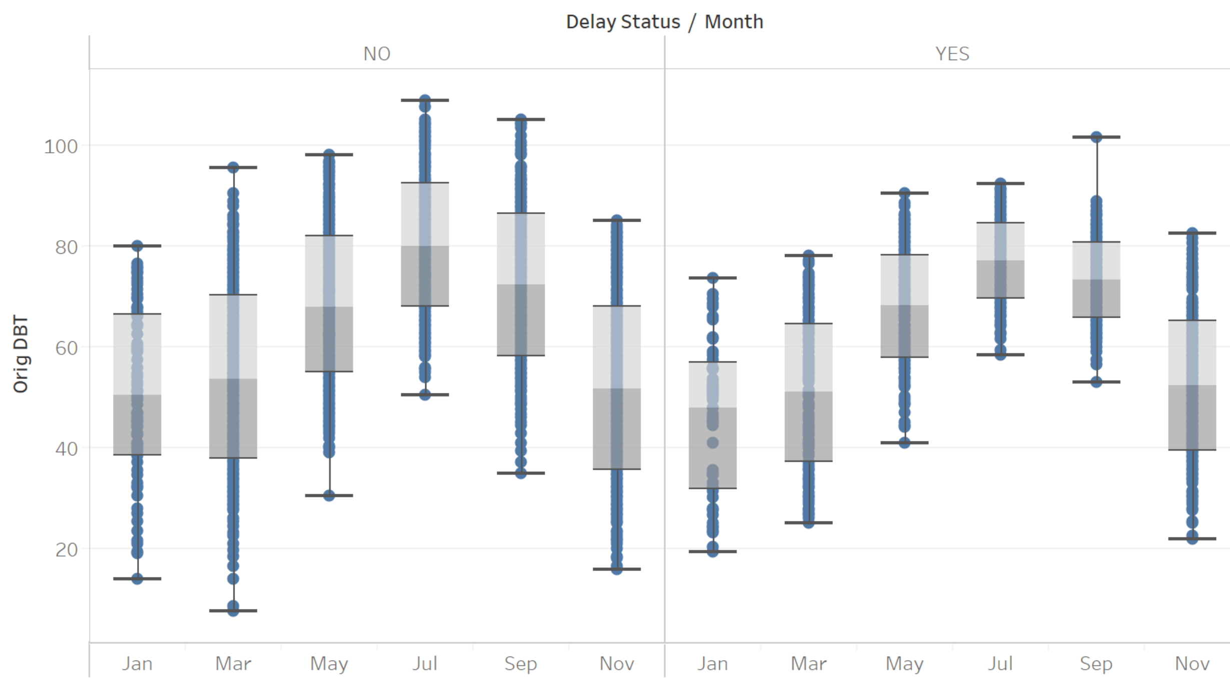
Orig Vis for each Month broken down by Delay Status.

Visibility at Destination vs Delay



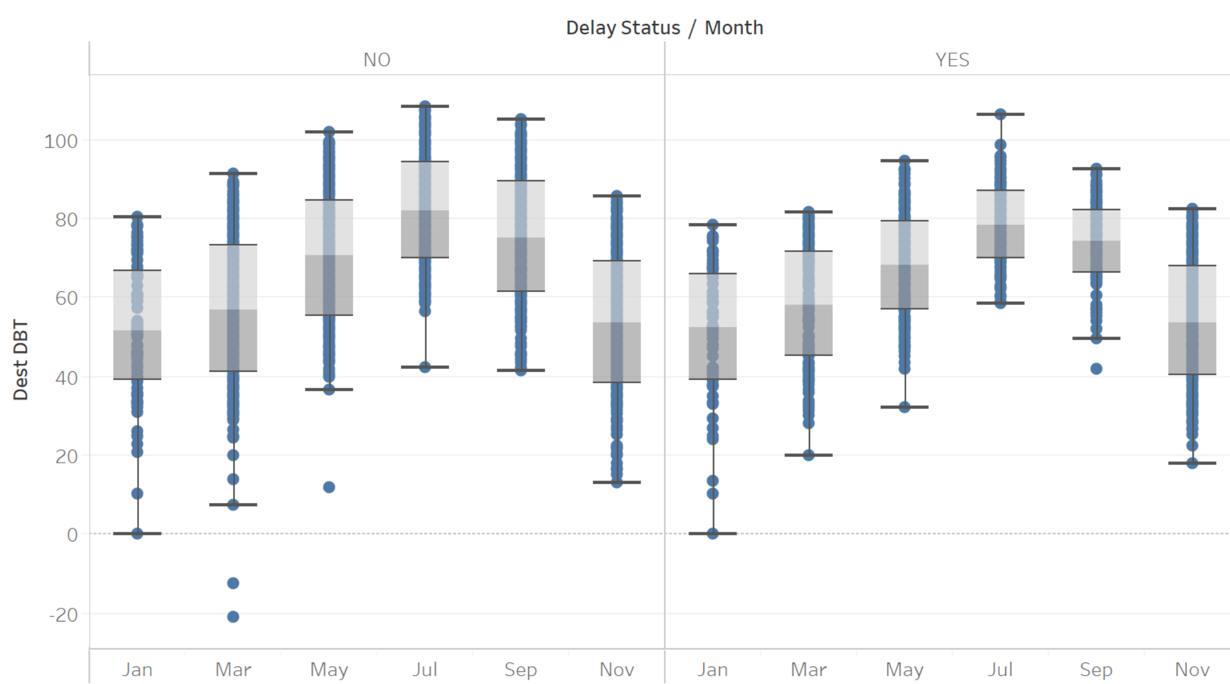
Dest Vis for each Month broken down by Delay Status.

Origin Dry Bulb Temperature vs Delay

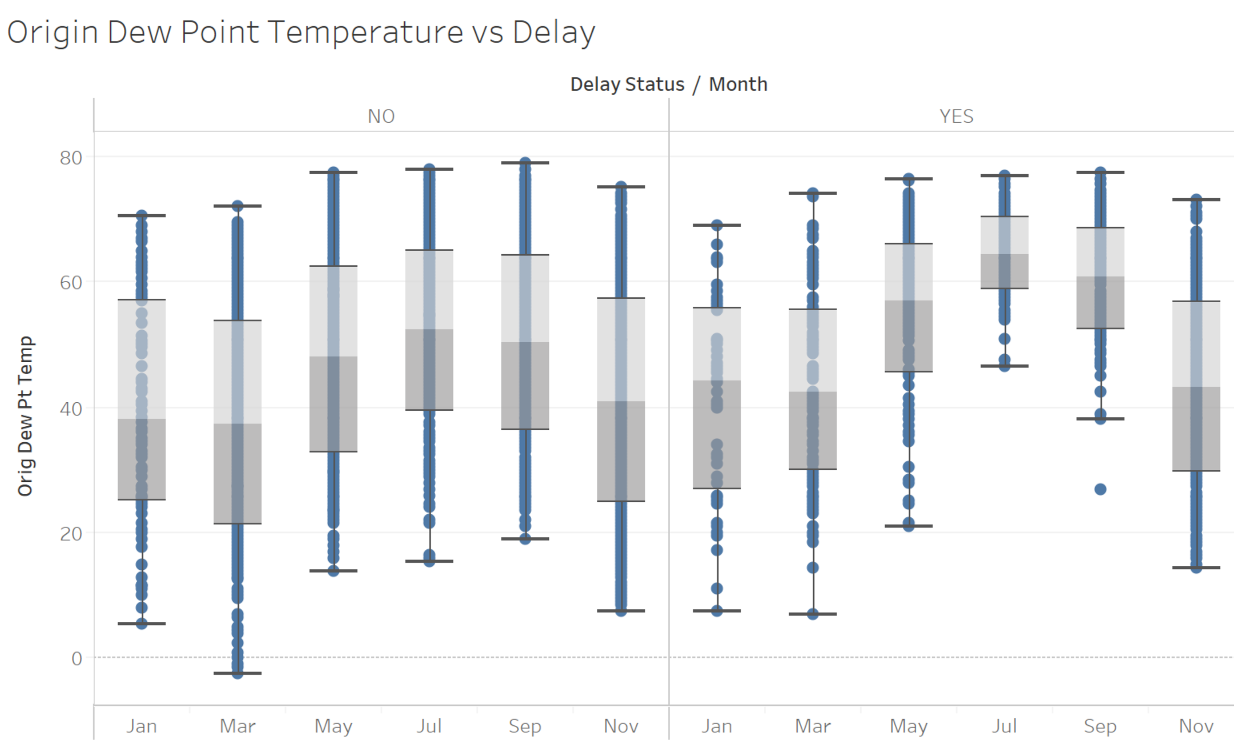


Orig DBT for each Month broken down by Delay Status.

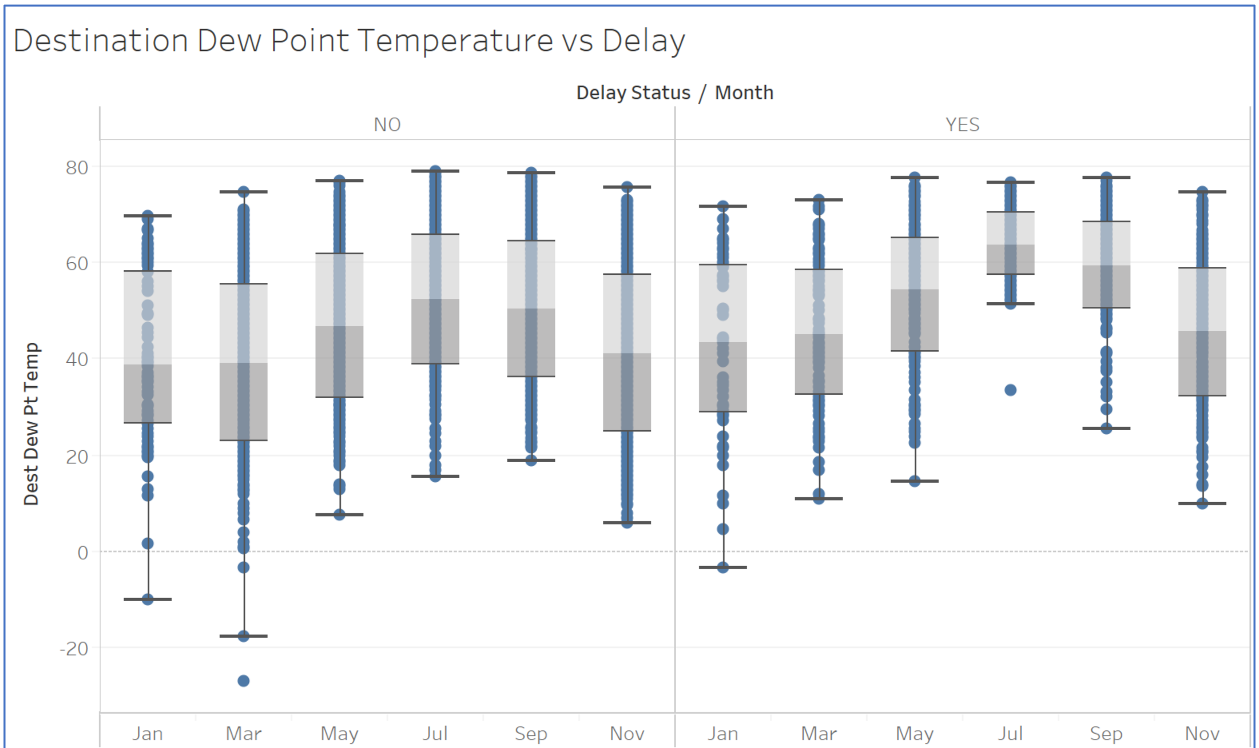
Destination Dry Bulb Temperature vs Delay



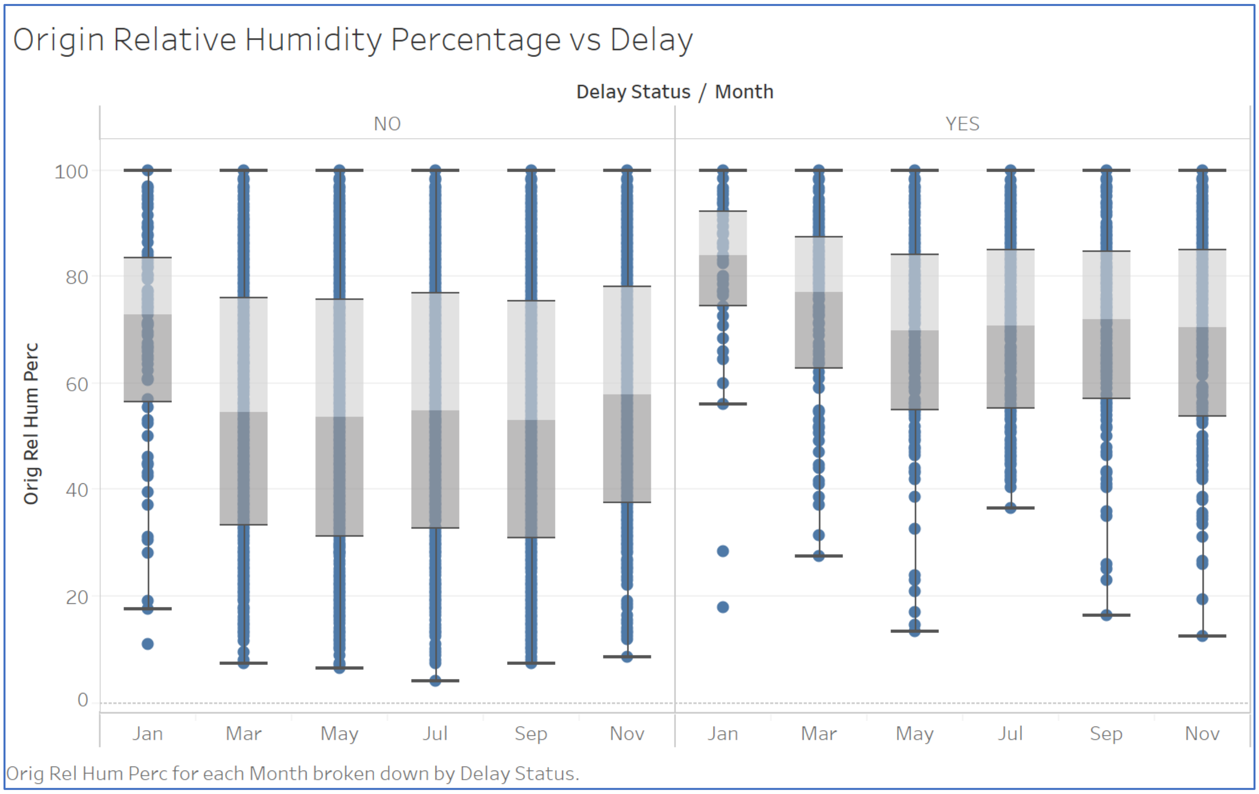
Dest DBT for each Month broken down by Delay Status.

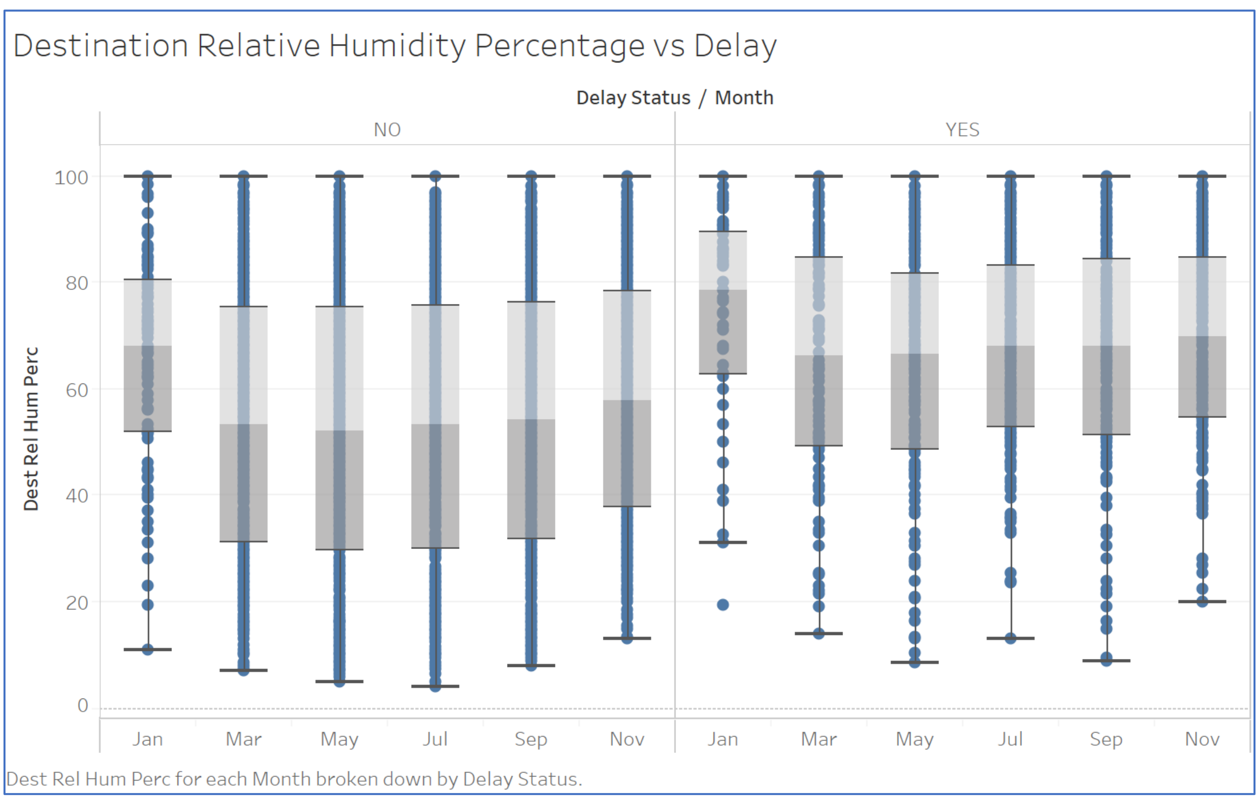


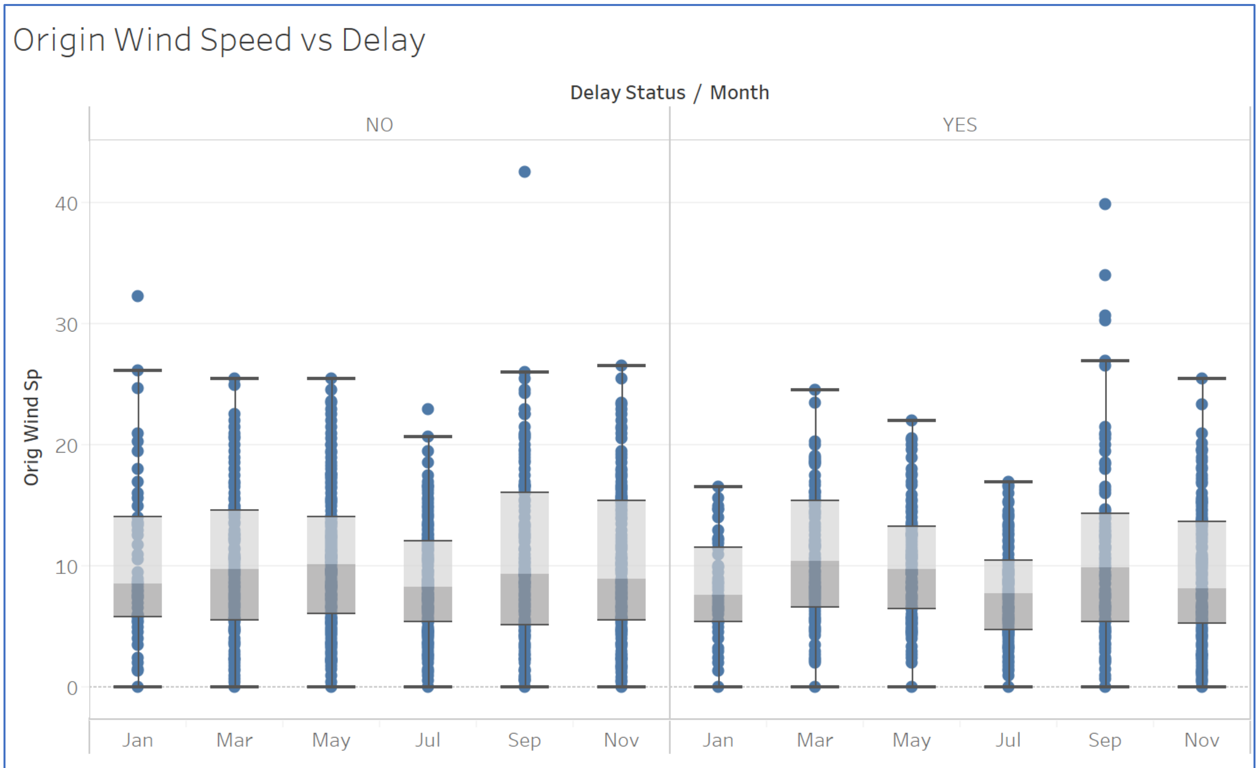
Orig Dew Pt Temp for each Month broken down by Delay Status.



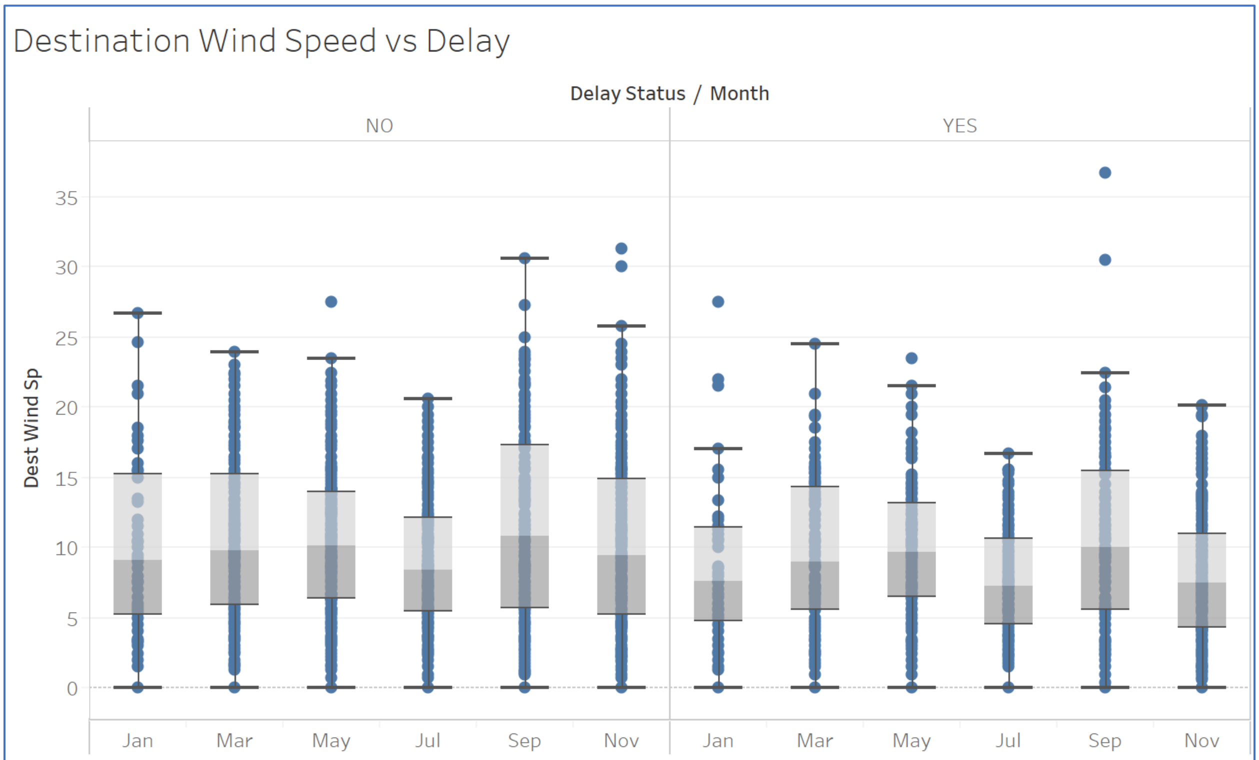
Dest Dew Pt Temp for each Month broken down by Delay Status.


Origin Relative Humidity Percentage vs Delay
Delay Status / Month
NO
YES
Orig Rel Hum Perc
100
80
60
40
20
0
Jan
Mar
May
Jul
Sep
Nov
Jan
Mar
May
Jul
Sep
Nov
Orig Rel Hum Perc for each Month broken down by Delay Status.

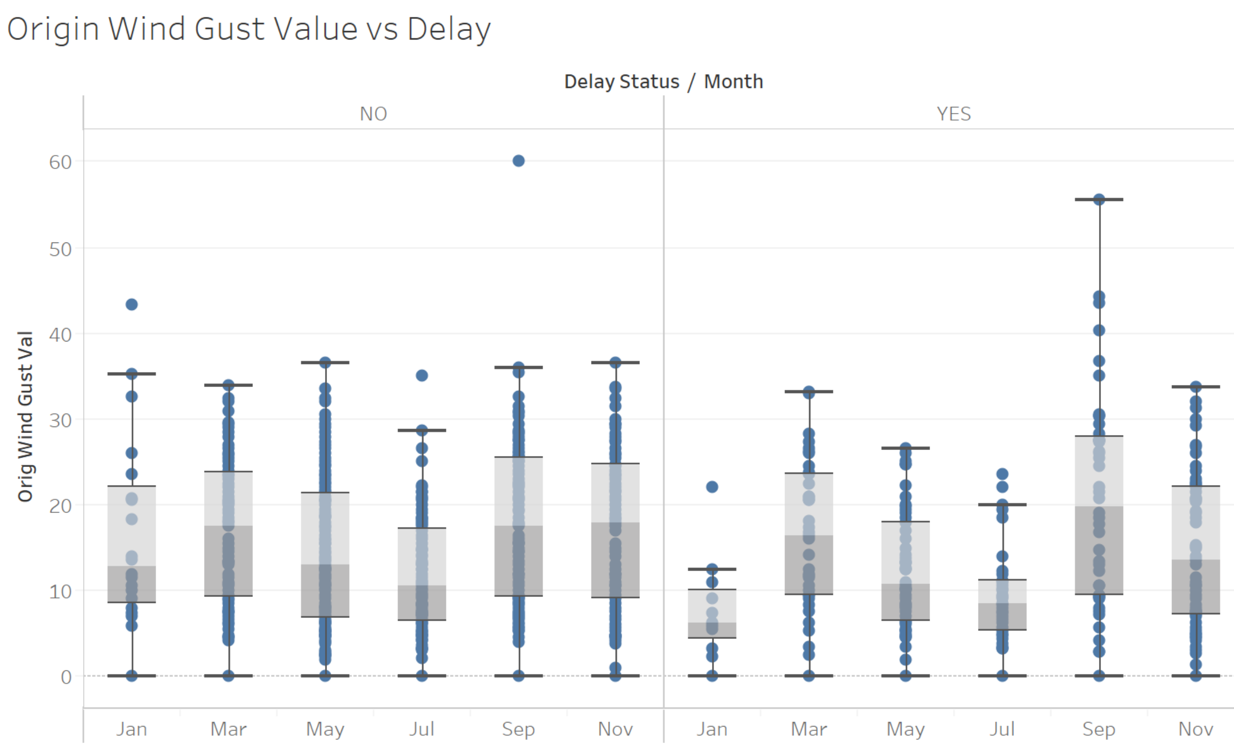

Destination Relative Humidity Percentage vs Delay
Delay Status / Month
NO
YES
Dest Rel Hum Perc
100
80
60
40
20
0
Jan
Mar
May
Jul
Sep
Nov
Jan
Mar
May
Jul
Sep
Nov
Dest Rel Hum Perc for each Month broken down by Delay Status.



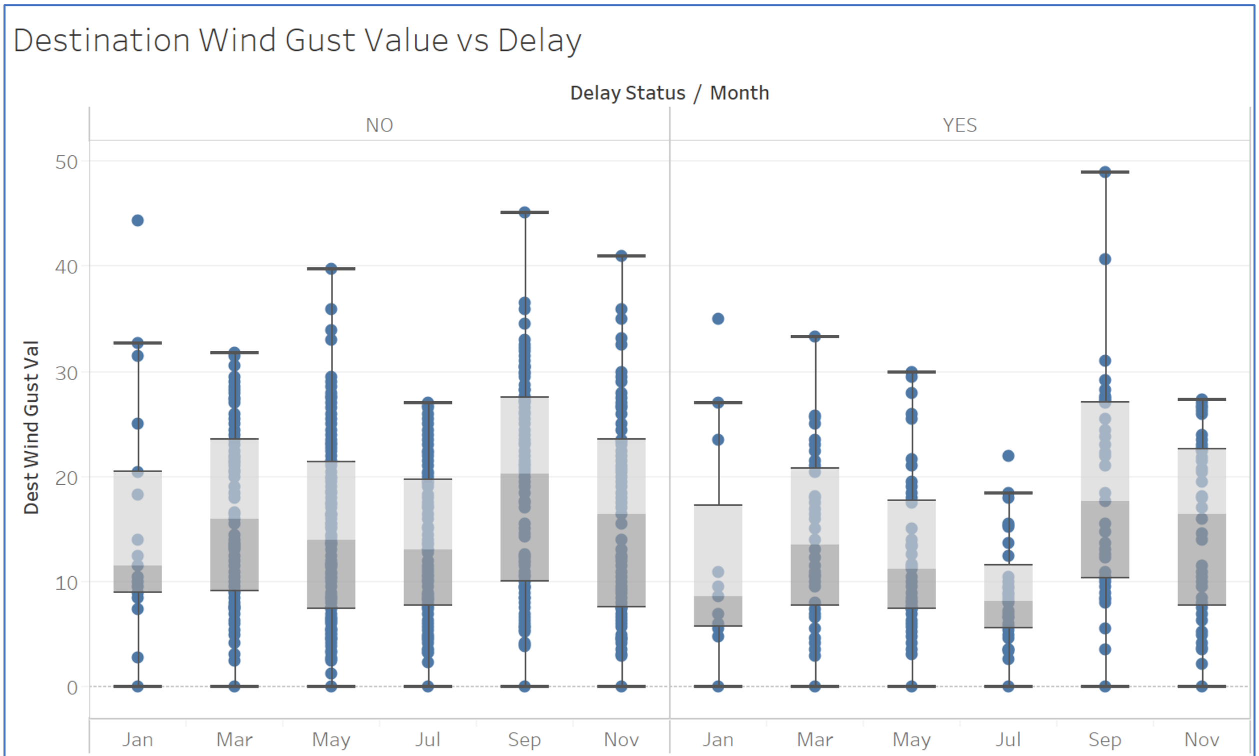
Orig Wind Sp for each Month broken down by Delay Status.



Dest Wind Sp for each Month broken down by Delay Status.

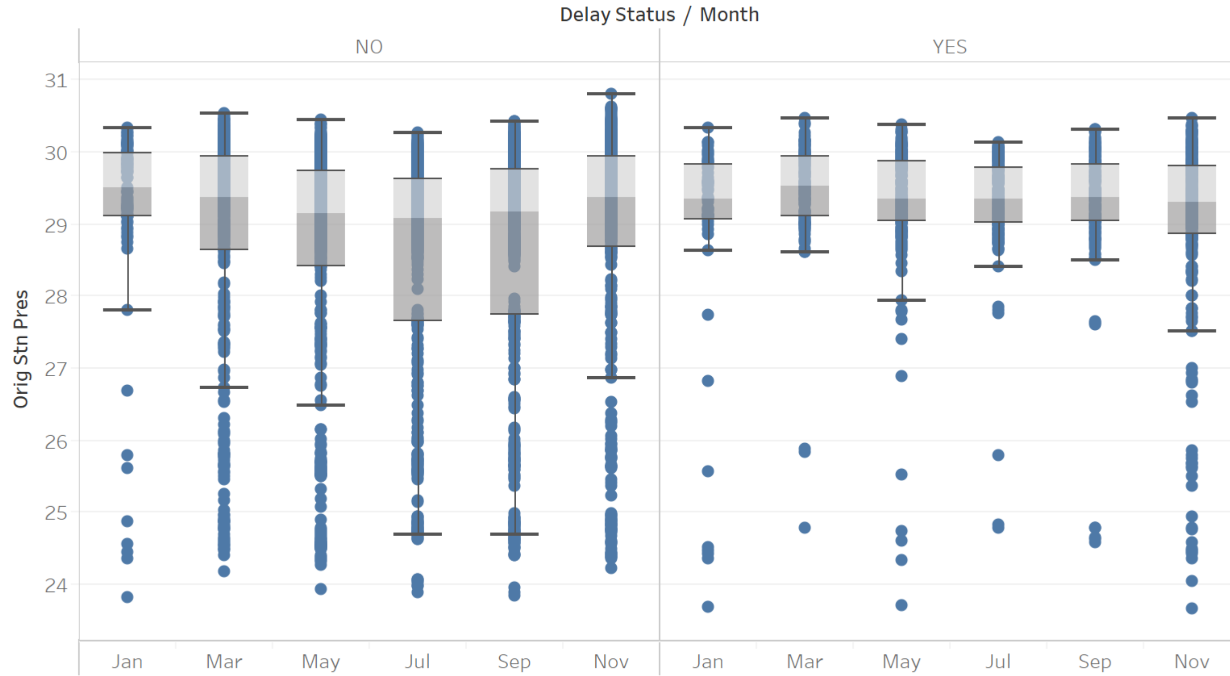


Orig Wind Gust Val for each Month broken down by Delay Status.



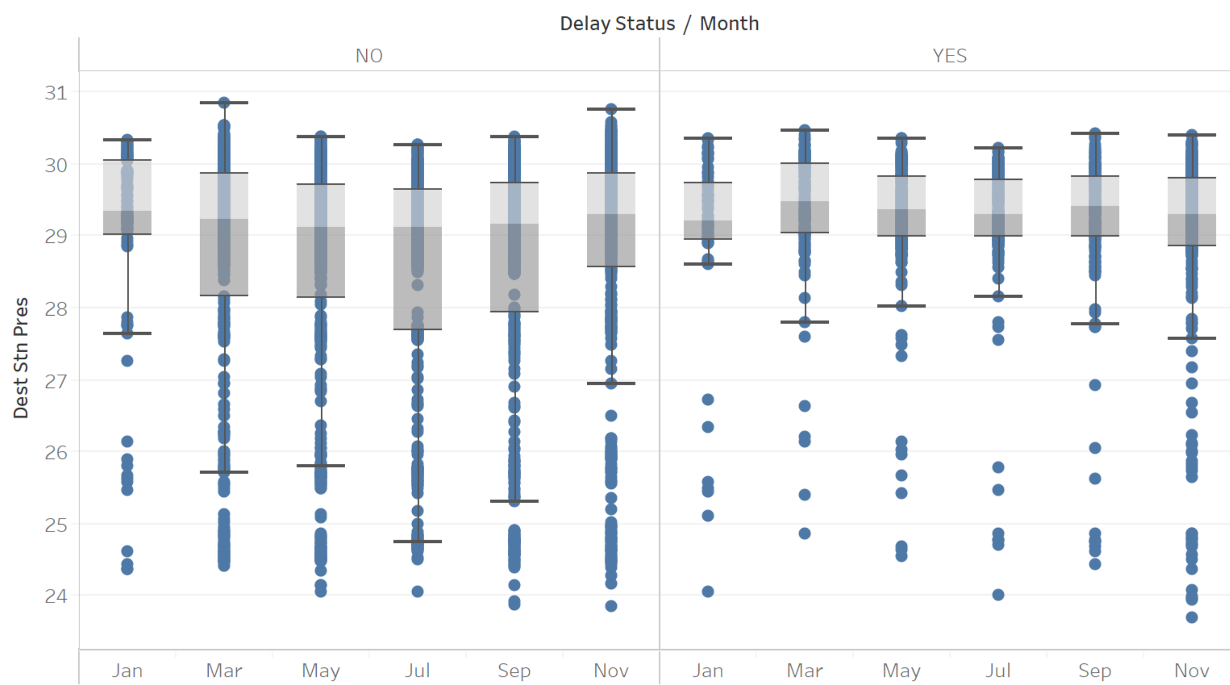
Dest Wind Gust Val for each Month broken down by Delay Status.

Origin Station Pressure vs Delay



Orig Stn Pres for each Month broken down by Delay Status.

Destination Station Pressure vs Delay



Dest Stn Pres for each Month broken down by Delay Status.

Further Pre – processing based on Visualization & Feature Engineering

Read in '*flight.rds*' and '*flighttest.rds*' as derived in the previous step. Following features were generated

1. US Holiday Lists for 2004[3] & 2005[4] were downloaded and a new column (number of days from nearest holiday was generated). Code snapshot is given below.

```r
# Deriving column days from nearest holiday for 2004

x<-1:nrow(flight) # Calculating number of rows
y<-1:nrow(hols04) # Calculating number of rows

flight$nrsthol<-365
flight$nrsthol<-as.integer(flight$nrsthol)
flight$SchedDate<- as.POSIXct(flight$SchedDate, format="%Y-%m-%d",tz = 'GMT')

for (i in seq_along(x)) {
  for (j in seq_along(y)) {
    temp1<-flight[i,c('nrsthol')]
    temp2<-difftime(flight[i,c('SchedDate')], hols04[j,c('Date')],units = "days")
    temp2<-abs(as.integer(temp2))
    temp1<-ifelse(temp2<temp1,temp2,temp1)
    flight[i,c('nrsthol')]<-temp1
  }
}
```

Fig 12: Derivation of 'nrsthol' column for 2004

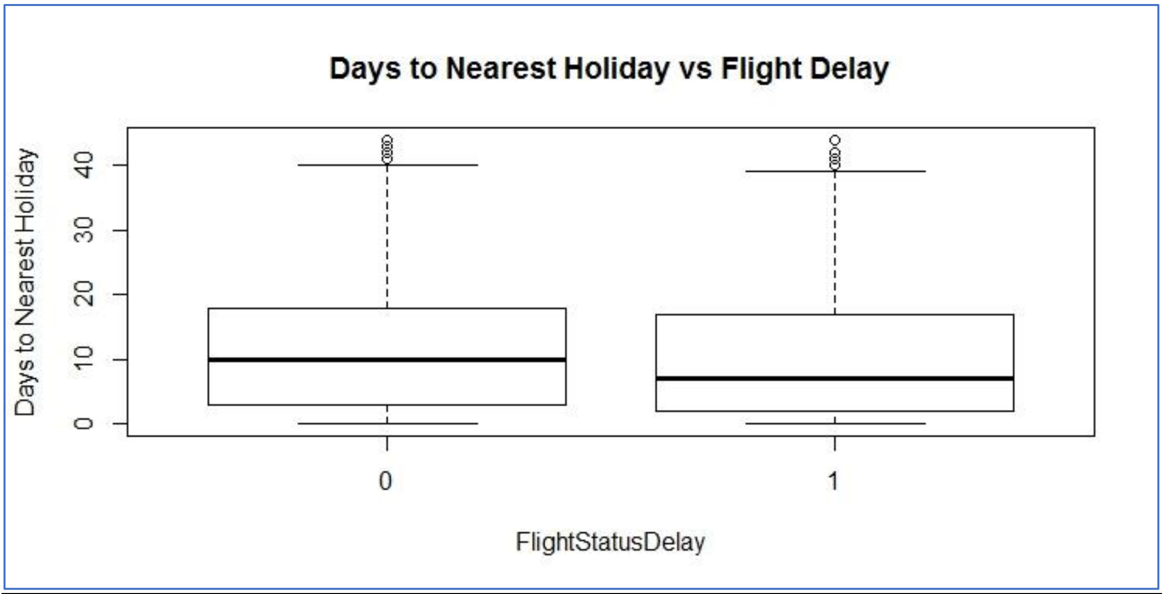


Fig 13: Distribution of Days from Nearest Holiday vs Delay Status

2. Impute any remaining missing values using Central Imputation from DMwR.

[3] US 2004 Holiday List: https://www.timeanddate.com/holidays/us/2004
[4] US 2005 Holiday List: https://www.timeanddate.com/holidays/us/2005

3. Reduce the number of levels in Origin & Destination based on Visualization keeping the most important ones and tagging rest as 'Others'.
4. Similarly, reduce the number of levels in Scheduled Departure & Scheduled Arrival time slots.
5. **Dew Point Temperature – Dry Bulb Temperature Ratio:** The dew point in relation to the temperature gives the pilots information about the humidity and can affect visibility. If the dew point is close to the temperature, humidity is high, which can cause hazy conditions or even fog.[5] Keeping this in mind, the ratio of Dry Point Temperature & Dew Point Temperature was created as an additional feature.

```
# Creating DewPtTemp DBT Ratio

flight$OrigDPTDBTRat<-flight$OrigDewPtTemp/flight$OrigDBT
flight$DestPTDBTRat<-flight$DestDewPtTemp/flight$DestDBT

flighttest$OrigDPTDBTRat<-flighttest$OrigDewPtTemp/flighttest$OrigDBT
flighttest$DestPTDBTRat<-flighttest$DestDewPtTemp/flighttest$DestDBT

# Calculating MaxDewPtTempDBTRatio

flight$DEwPTTempDBTRatMax<-apply(flight[,c('OrigDPTDBTRat','DestPTDBTRat')],1,max)
flighttest$DEwPTTempDBTRatMax<-apply(flighttest[,c('OrigDPTDBTRat','DestPTDBTRat')],1,max)

flight$OrigDPTDBTRat<-NULL
flight$DestPTDBTRat<-NULL
flighttest$OrigDPTDBTRat<-NULL
flighttest$DestPTDBTRat<-NULL
```

Fig 14: Creation of Feature DewPtTemp/DBT

6. **Density Altitude:** Air density is perhaps the single most important factor affecting aircraft performance. It has a direct bearing on.
   a) Lift generated by the wing
   b) Efficiency of a propeller or rotor
   c) The power output of any engine etc.[6]

   Therefore, with the existing available features, the density altitude was derived as an additional feature using the following formula:

$$\mathrm{DA} = (145442.16\ \mathrm{ft}) \times \left(1 - \left[\frac{(17.326\ ^\circ\mathrm{F}/\mathrm{inHg}) \times P}{459.67\ ^\circ\mathrm{F} + T}\right]^{0.235}\right)$$

[5] https://aviation.stackexchange.com/questions/25231/why-do-pilots-need-the-ceiling-time-and-dew-point-in-the-atis
[6] https://en.wikipedia.org/wiki/Density_altitude

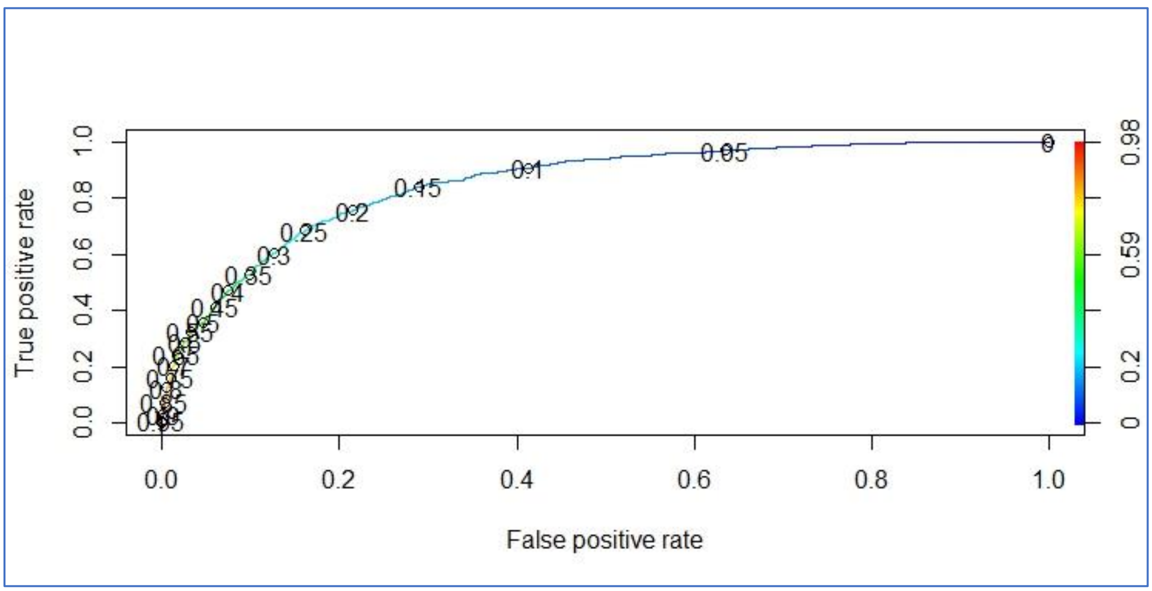
```
# Calculating Density Altitude

flight$OrigDBTF<-flight$OrigDBT*1.8+32
flight$DestDBTF<-flight$DestDBT*1.8+32

flighttest$OrigDBTF<-flighttest$OrigDBT*1.8+32
flighttest$DestDBTF<-flighttest$DestDBT*1.8+32

flight$OrigDA<-145442.16*(1-((17.326*flight$OrigStnPres)/(459.67+flight$OrigDBTF))^0.235)
flight$DestDA<-145442.16*(1-((17.326*flight$DestStnPres)/(459.67+flight$DestDBTF))^0.235)

flighttest$OrigDA<-145442.16*(1-((17.326*flighttest$OrigStnPres)/(459.67+flighttest$OrigDBTF))^0.235)
flighttest$DestDA<-145442.16*(1-((17.326*flighttest$DestStnPres)/(459.67+flighttest$DestDBTF))^0.235)

flight$MaxDA<-apply(flight[,c('OrigDA','DestDA')],1,max)
flighttest$MaxDA<-apply(flighttest[,c('OrigDA','DestDA')],1,max)
```

Fig 15: Calculating Density Altitude

7. This was followed by splitting the data into Train & Validation and then Standardization of numerical features.

Model Building

1. **Logistic Regression:** The first model built on the train data was Logistic Regression using Step AIC. After removing multi – collinearities, the final model emerged with the following ROC Curve.



Following are the performance metrics of the model:

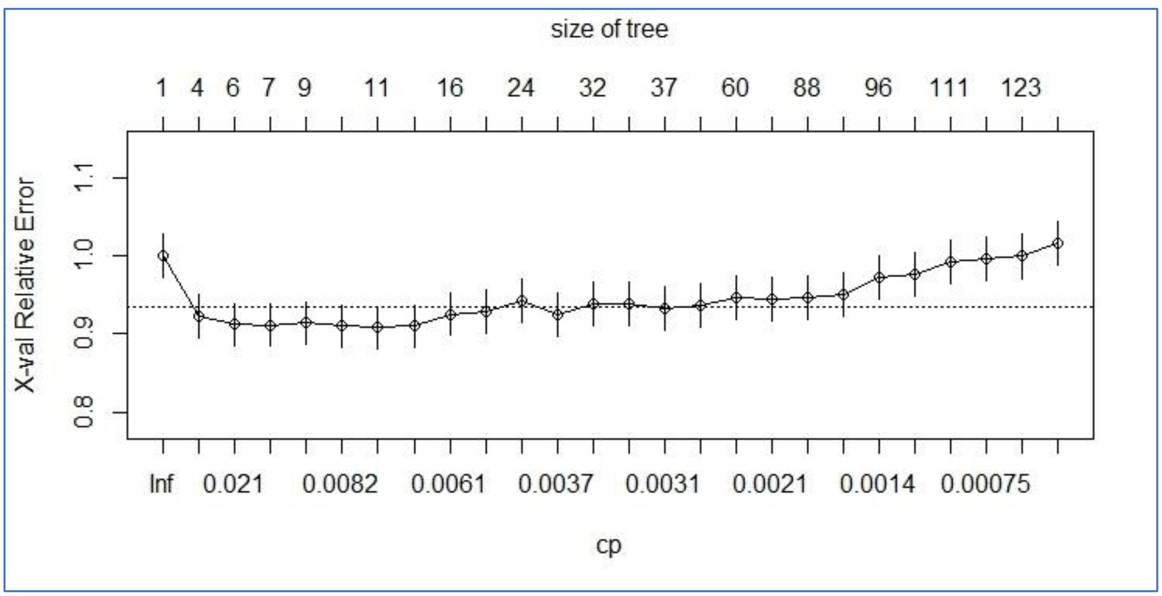
Metric	Train	Validation
Accuracy	0.8449	0.8485
Sensitivity/Recall	0.326	0.346
Precision	0.6977	0.708
F1 Score	0.4448	0.465

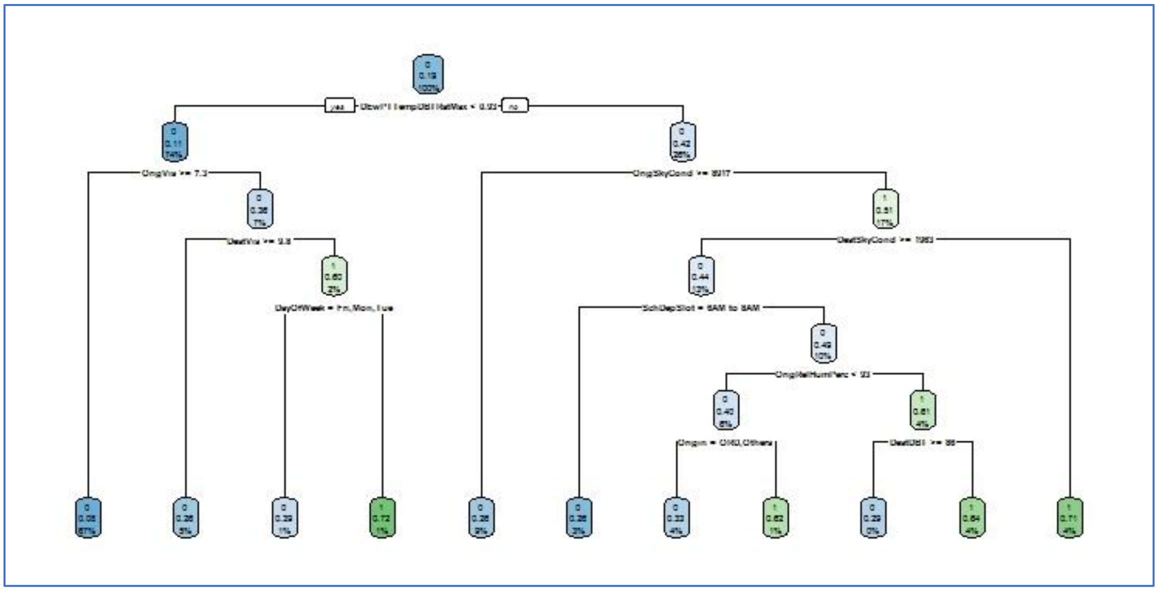
2. **Decision Trees – CART:** Following are the important features and performance metrics on Validation Data.

```
DEwPTTempDBTRatMax      OrigRelHumPerc      DestRelHumPerc            OrigVis
       211.7488803         149.7313904          99.0939060         69.6231826
       DestSkyCond          OrigPrecip         OrigSkyCond         SchDepSlot
        66.8187516          45.5390075          42.6171986         13.1025865
           DestVis              Origin             OrigDBT         SchArrSlot
        11.1245245           9.5427039           8.2176088          5.8049434
        DestPrecip       OrigDewPtTemp         OrigWindDir              MaxDA
         2.6631225           2.0348365           1.7246125          0.9063118
          Distance          OrigWindSp        DestDewPtTemp
         0.3976127           0.1658555           0.1325376
```

Validation Performance of basic CART Model	
Accuracy	0.8356
Sensitivity	0.3393
Specificity	0.9522
F1 Score	0.3533

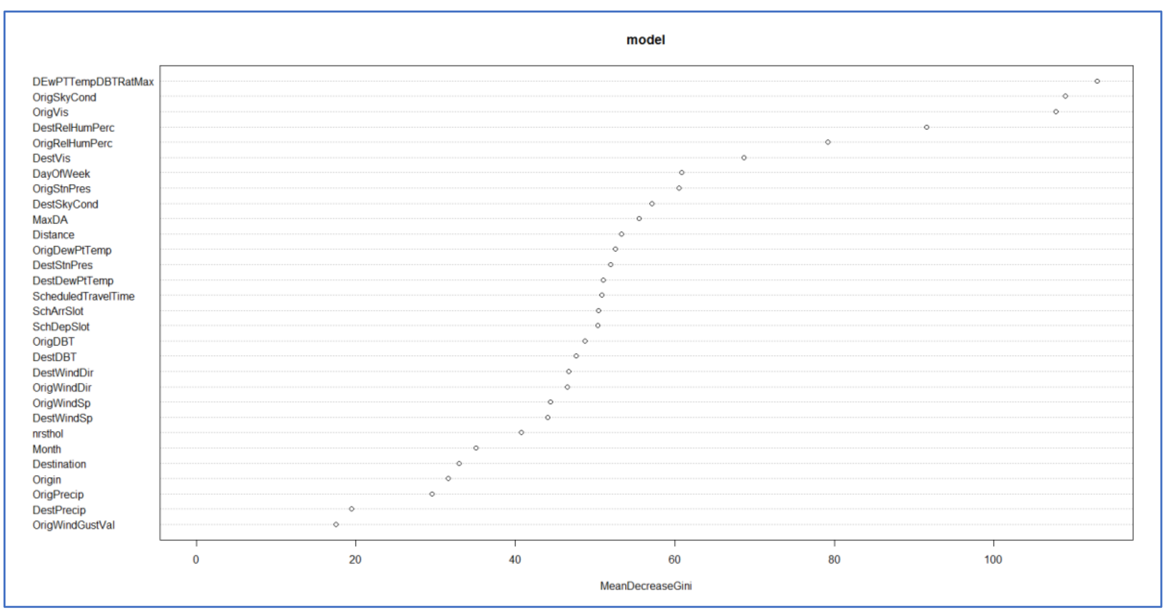
<u>Model tuning by changing cp</u>





Validation Performance of tuned CART Model	
Accuracy	0.8378
Sensitivity	0.3710
Specificity	0.9474
F1 Score	0.3768

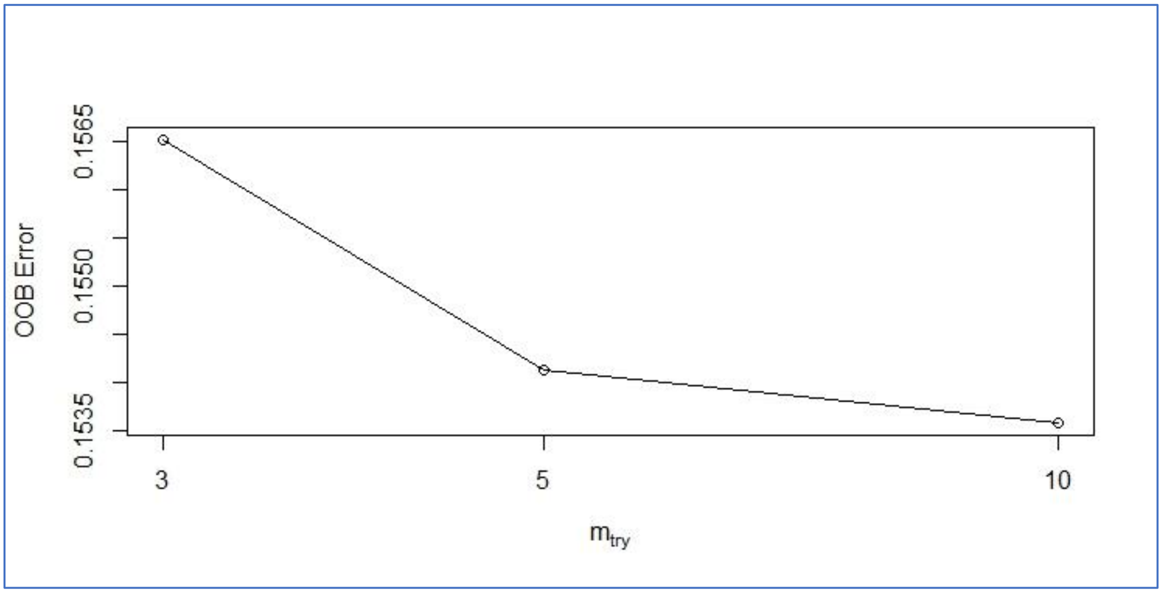
3. **Random Forest:** Following is a summary of important variables and performance on Validation Data.



Validation Performance of basic RF Model	
Accuracy	0.858

Sensitivity	0.4118
Specificity	0.9628
F1 Score	0.4483

Model tuning using mtry

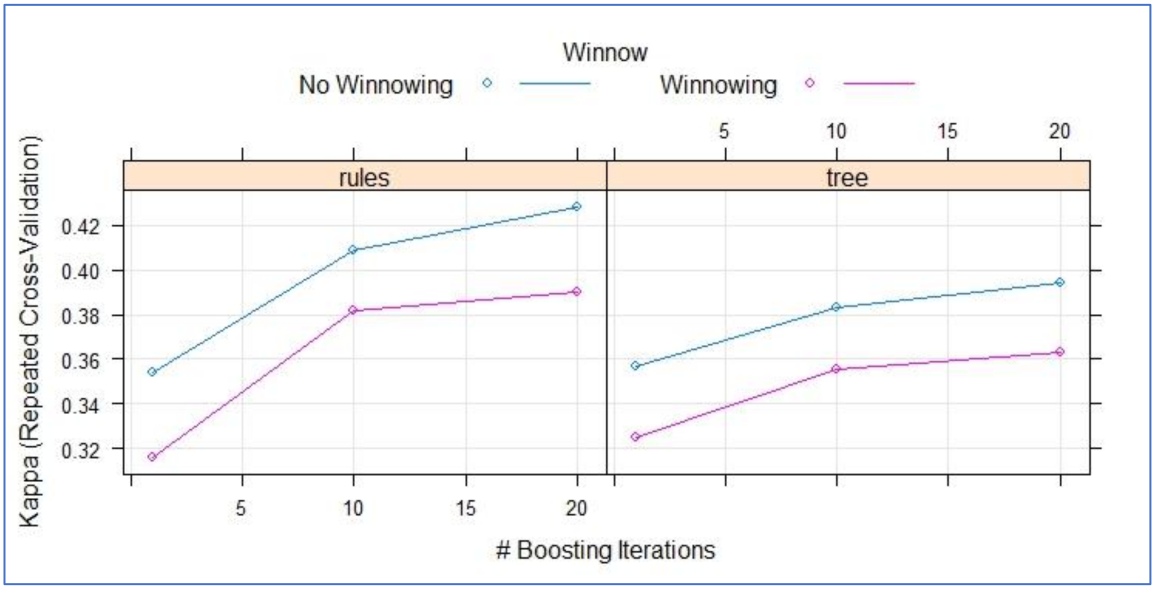


Validation Performance of tuned RF Model	
Accuracy	0.8597
Sensitivity	0.4186
Specificity	0.9633
F1 Score	0.4561

4. **C5.0 with Cross Validation:** Following are the performance metrics on Validation Data.

```r
# Trying C5.0 CV using CARET

library(C50)

set.seed(123)

tuned<-train(train_data[,1:31],train_data$FlightDelayStatus,
             method = "C5.0",
             trControl = trainControl(method = "repeatedcv",repeats = 7),
             control = C5.0Control(earlyStopping = FALSE),
             metric = "Kappa") # Building a tuned model

# Predicting on Validation Data

preds_tuned<-predict(tuned,validation_data) # Predicting on Val Data

confusionMatrix(preds_tuned,validation_data$FlightDelayStatus, positive = "1")
```

Accuracy	0.8386
Sensitivity	0.4977
Specificity	0.9187
F1 Score	0.4429



Feature Selection using CARET '*rfe*'

```
#Feature selection using rfe in caret

control <- rfeControl(functions = rfFuncs,
                      method = "repeatedcv",
                      repeats = 3,
                      verbose = FALSE)

outcomeName<-'FlightDelayStatus'

predictors<-names(train_dummy_data)[!names(train_dummy_data) %in% outcomeName]

Delay_Pred_Profile <- rfe(train_dummy_data[,predictors], train_dummy_data[,outcomeName],
                         rfeControl = control, metric = "Kappa")

Delay_Pred_Profile$optVariables
```

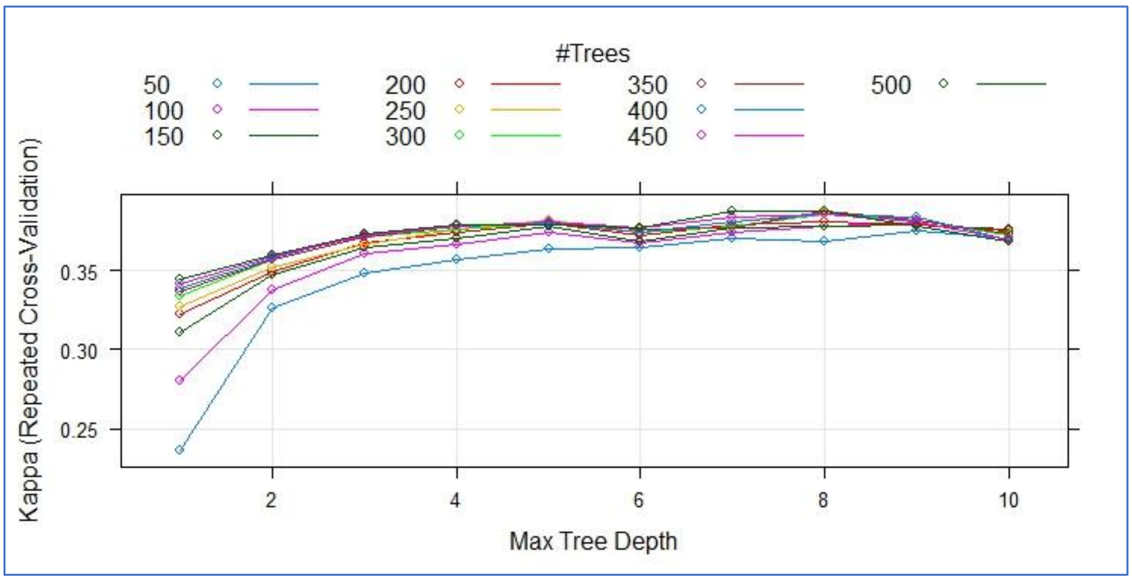
Large Subset: Visibility at Origin, Origin Sky Conditions, Visibility at Destination, Dew Point Temperature DBT Ratio, Destination Relative Humidity Percentage, Origin Relative Humidity Percentage, Destination Sky Conditions, Origin Dew Point Temperature, Origin Station Pressure, Origin Precipitation, Origin DBT, Destination Dew Point Temperature, Density Altitude, Destination DBT, Destination Precipitation, Distance.

Small Subset: Visibility at Origin, Origin Sky Conditions, Visibility at Destination, Dew Point Temperature DBT Ratio, Destination Relative Humidity Percentage

5. **ADA Boost with Large Subset Feature Selection:** Following are the performance metrics on Validation Data.

```
predictors<-c('OrigVis','OrigSkyCond','DestRelHumPerc','DestVis','OrigRelHumPerc','DestSkyCond',
              'OrigDewPtTemp','OrigPrecip','OrigStnPres','OrigDBT','DestDewPtTemp','DestPrecip',
              'DestDBT','Distance','nrsthol','Origin.Others')

# Modeling using ADABoost

model_adaTL<-train(train_dummy_data[,predictors],train_dummy_data[,outcomeName],method='ada',
                   trControl=fitControl,tuneLength=10,metric = "Kappa")

plot(model_adaTL)

print(model_adaTL)

preds_adaTL<-predict(model_adaTL,val_dummy_data) # Predicting on Val Data

confusionMatrix(preds_adaTL,val_dummy_data$FlightDelayStatus, positive = "1")
```

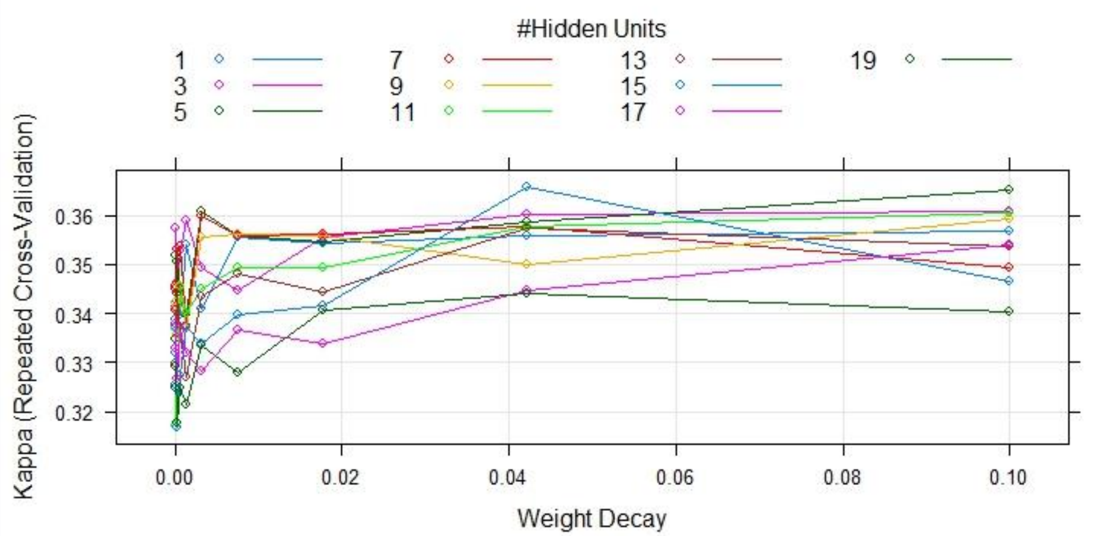
Accuracy	0.8438
Sensitivity	0.3914
Specificity	0.9501
F1 Score	0.4024



6. **Neural Net with Large Subset Feature Selection:** Following are the performance metrics on Validation Data.

```
# Modeling using NeuralNet

model_NNTL<-train(train_dummy_data[,predictors],train_dummy_data[,outcomeName],method='nnet',
                  trControl=fitControl,tuneLength=10,metric = "Kappa")

plot(model_NNTL)

print(model_NNTL)

preds_NNTL<-predict(model_NNTL,val_dummy_data) # Predicting on Val Data

confusionMatrix(preds_NNTL,val_dummy_data$FlightDelayStatus, positive = "1")
```

Accuracy	0.8231
Sensitivity	0.3937
Specificity	0.9240
F1 Score	0.3562

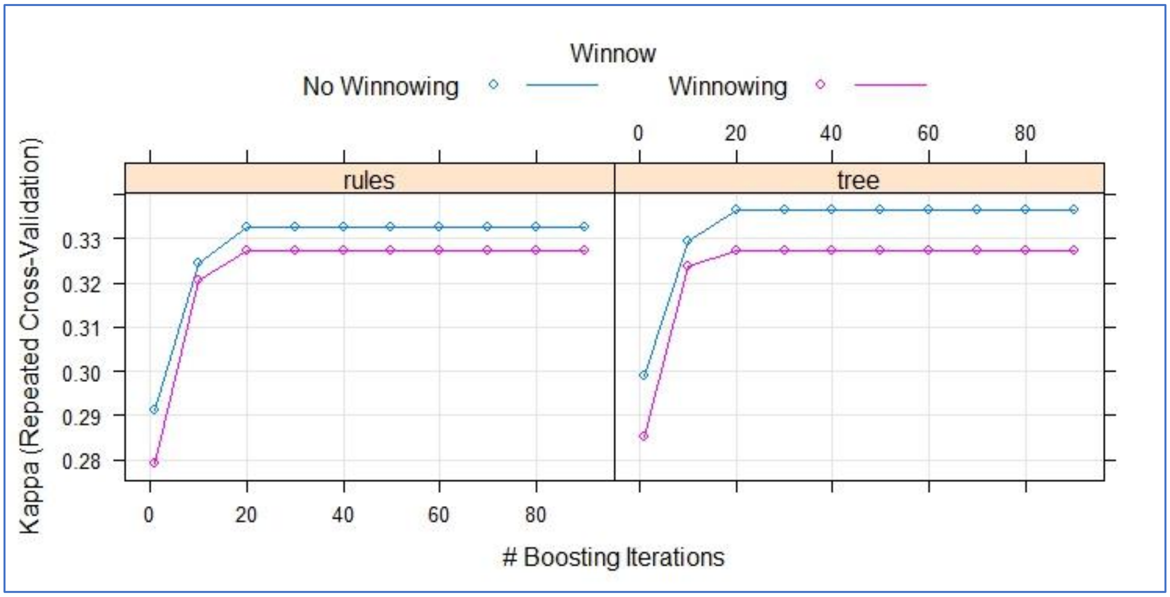


7. **C5.0 with Small Subset Feature Selection:** Following are the performance metrics on Validation Data.

```
# Trying C5

model_C5TL<-train(train_dummy_data[,predictors],train_dummy_data[,outcomeName],method='C5.0',
                  trControl=fitControl,tuneLength=10,metric = "Kappa")

preds_c5<-predict(model_C5TL,val_dummy_data) # Predicting on Val Data

confusionMatrix(preds_c5,val_dummy_data$FlightDelayStatus, positive = "1")

plot(model_C5TL)
```

Accuracy	0.8382
Sensitivity	0.3213
Specificity	0.9596

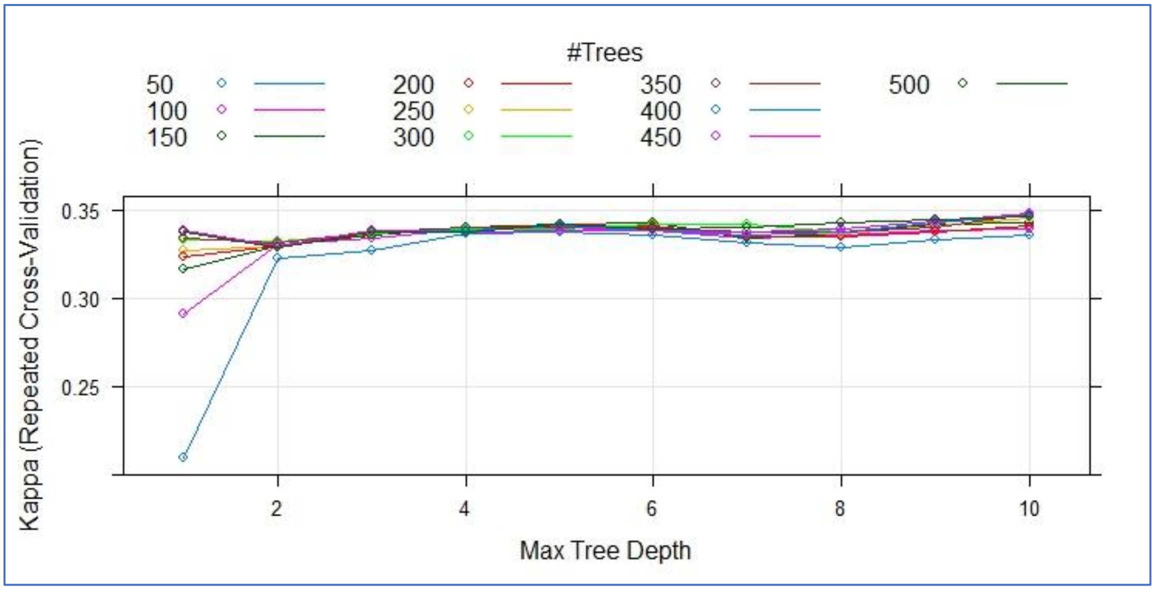
F1 Score	0.3484



8. **ADA Boost with Small Subset Feature Selection:** Following are the performance metrics on Validation Data.

```
# Trying ADABoost

model_adaTL<-train(train_dummy_data[,predictors],train_dummy_data[,outcomeName],method='ada',
                   trControl=fitControl,tuneLength=10,metric = "Kappa")

plot(model_adaTL)

preds_adaTL<-predict(model_adaTL,val_dummy_data) # Predicting on Val Data

confusionMatrix(preds_adaTL,val_dummy_data$FlightDelayStatus, positive = "1")
```

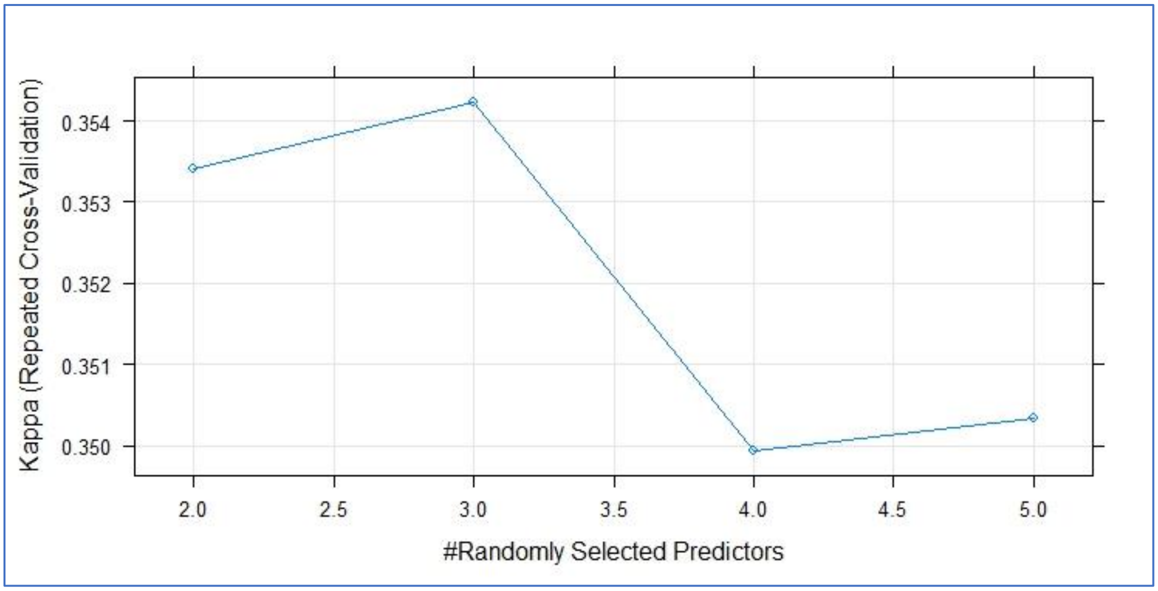
Accuracy	0.8391
Sensitivity/Recall	0.3801
Specificity	0.9469
Kappa	0.3849



9. **Random Forest with Small Subset Feature Selection:** Following are the performance metrics on Validation Data.

```
# Trying RF

model_rfTL<-train(train_dummy_data[,predictors],train_dummy_data[,outcomeName],method='rf',
                  trControl=fitControl,tuneLength=10,metric = "Kappa")

plot(model_rfTL)

preds_rf<-predict(model_rfTL,val_dummy_data) # Predicting on Val Data

confusionMatrix(preds_rf,val_dummy_data$FlightDelayStatus, positive = "1")
```

Accuracy	0.8305
Sensitivity	0.3823
Specificity	0.9357
Kappa	0.3662


Kappa (Repeated Cross-Validation)
0.354
0.353
0.352
0.351
0.350
2.0
2.5
3.0
3.5
4.0
4.5
5.0
#Randomly Selected Predictors