Assignment #4

PREDICTING DELAYED FLIGHTS

20190552 손지영

사용 데이터

- **FlightRecords.csv** : 2004년 1월동안 Washington, DC 지역으로부터 New York City로 운행한 2201개의 항공기 운행 기록
- dayweek: 운행 요일 (1: 월, 2: 화, ..., 7: 일)
- deptime: 출발시각 (예: 1455 = 14시55분, 839: 8시39분)
- origin: 출발공항코드(DCA: Reagan Nation, IAD: Dulles, BWI: Baltimore-Washington Int'l)
- dest: 도착공항코드(JFK: Kennedy, LGA: LaGuardia, EWR: Newark)
- carrier: 항공사코드(CO: Continental, DH: Atlantic Coast, DL: Delta, MQ: American Eagle, OH: Comair, RU: Continental Express, UA: United, US: USAirways)
- weather: 날씨 (0: OK, 1: Bad)
- delay: 연착여부("delayed" or "ontime"

```
library(ggplot2)
library(gridExtra)
library(dplyr)
library(ggcorrplot)
library(rsample)
library(caret)
library(vip)
library(tidyr)
library(ROCR)
library(glmnet)
library(psych)
library(e1071)
```

1번

문제에서 주어진 조건에 맞춰 data preprocessing을 진행하자.

```
# 데이터 불러오기
fly = read.csv('FlightRecords.csv')
str(fly)
```

```
## 'data.frame':    2201 obs. of  13 variables:
##  $ schedtime   : int  1455 1640 1245 1715 1039 840 1240 1645 1715 2120 ...
##  $ carrier     : chr  "OH" "DH" "DH" "DH" ...
##  $ deptime     : int  1455 1640 1245 1709 1035 839 1243 1644 1710 2129 ...
```

```
##  $ dest        : chr  "JFK" "JFK" "LGA" "LGA" ...
##  $ distance    : int  184 213 229 229 229 228 228 228 228 228 ...
##  $ date        : chr  "1/1/2004" "1/1/2004" "1/1/2004" "1/1/2004" ...
##  $ flightnumber: int  5935 6155 7208 7215 7792 7800 7806 7810 7812 7814 ...
##  $ origin      : chr  "BWI" "DCA" "IAD" "IAD" ...
##  $ weather     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ dayweek     : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ daymonth    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ tailnu      : chr  "N940CA" "N405FJ" "N695BR" "N662BR" ...
##  $ delay       : chr  "ontime" "ontime" "ontime" "ontime" ...
```

```r
# 문제에서 주어진 열만 전체 데이터셋에서 추출하기
df_fly = fly[, c(13, 2, 3, 4, 8, 9, 10)]

# 조건에 맞게 출발 시간 추출 & 변환
df_fly = subset(df_fly, (600<=deptime) & (deptime<2200))
df_fly$deptime = as.integer(df_fly$deptime/100)
df_fly$deptime = as.factor(df_fly$deptime)

# delay변수 순서 변경하기
df_fly$delay = factor(df_fly$delay, levels=c('ontime','delayed'))

# 나머지 변수 factor로 변환하기
df_fly$dayweek = factor(df_fly$dayweek, levels=c('1','2','3','4','5','6','7'),
                        labels=c('월','화','수','목','금','토','일'))
df_fly$weather = factor(df_fly$weather, levels=c(0, 1), labels=c('OK','Bad'))
df_fly$carrier = factor(df_fly$carrier)
df_fly$dest = factor(df_fly$dest)
df_fly$origin = factor(df_fly$origin)
str(df_fly)
```

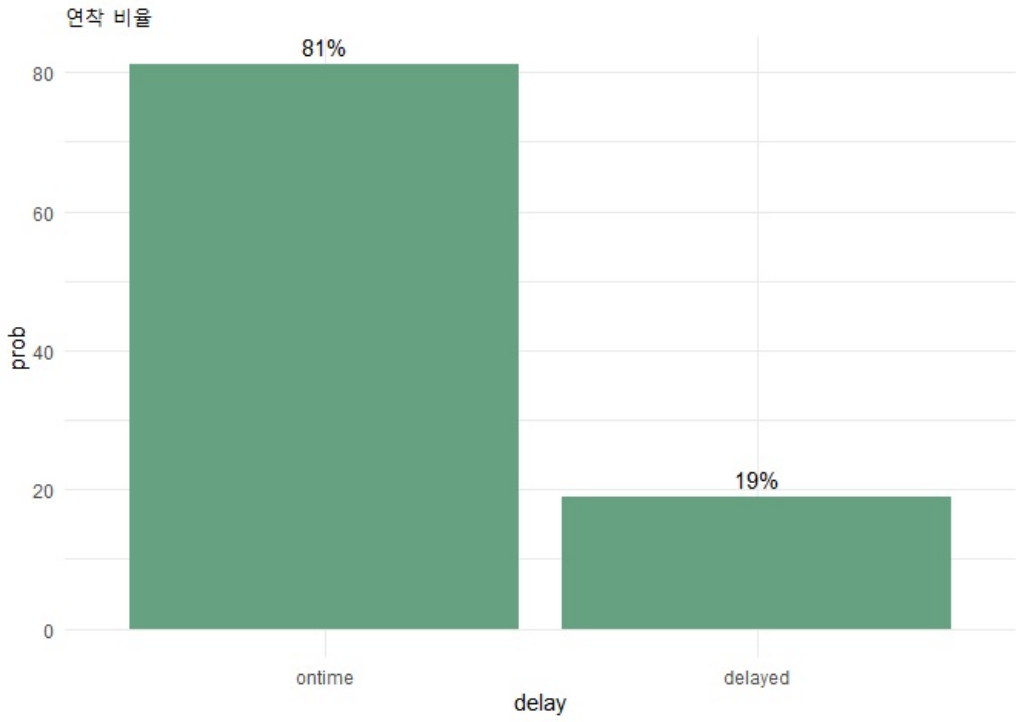
```
## 'data.frame':    2159 obs. of  7 variables:
##  $ delay  : Factor w/ 2 levels "ontime","delayed": 1 1 1 1 1 1 1 1 1 1 ...
##  $ carrier: Factor w/ 8 levels "CO","DH","DL",..: 5 2 2 2 2 2 2 2 2 2 ...
##  $ deptime: Factor w/ 16 levels "6","7","8","9",..: 9 11 7 12 5 3 7 11 12 16 ...
##  $ dest   : Factor w/ 3 levels "EWR","JFK","LGA": 2 2 3 3 3 2 2 2 2 2 ...
##  $ origin : Factor w/ 3 levels "BWI","DCA","IAD": 1 2 3 3 3 3 3 3 3 3 ...
##  $ weather: Factor w/ 2 levels "OK","Bad": 1 1 1 1 1 1 1 1 1 1 ...
##  $ dayweek: Factor w/ 7 levels "월","화","수",..: 4 4 4 4 4 4 4 4 4 4 ...
```

2번

요일 별 연착비율, 출발 시간대 별 연착 비율, 출발 공항 별 연착비율, 도착 공항 별 연착 비율, 항공사 별 연착 비율, 날씨 별 연착 비율을 각각 그래프로 시각화해보자. 어떤 특성을 관찰할 수 있는가?

[전체 연착 비율 비교]

```r
# 전체 연착 비율 살펴보기 위해 변환
df_fly_cnt = count(df_fly, delay)
df_fly_cnt$prob = df_fly_cnt$n / sum(df_fly_cnt$n) *100
# 실제 비율을 %로 알아보고 싶어 위와 같이 처리하였다.

# 전체 연착 비율 살펴보기
ggplot(df_fly_cnt, aes(x=delay, y=prob)) +
  geom_bar(fill='#66a182', stat='identity') +
  labs(title='연착 비율') +
  theme_minimal() +
  geom_text(aes(label=paste0(round(prob), "%")), vjust=-0.5)
```



먼저 delay변수 내의 두 범주간의 비율을 살펴보았다. ontime과 delayed의 비율이 약 4:1로 불균형한 분포를 갖고 있는 타겟변수임을 확인하였다.

[변수별 연착 비율 비교]

```
# 연착 비율 선그래프 함수
line_graph = function(df, var, x_label, total_title){
  df_total = df %>% count({{var}})
  df_cnt = df %>% group_by({{var}}, delay) %>% summarize(count = n(), .groups = 'drop')

  p = ggplot() +
    geom_line(df_cnt, mapping=aes(x={{var}}, y=count, group=delay, color=delay),linewidth=1) +
    geom_point(df_cnt, mapping=aes(x={{var}}, y=count, group=delay, color=delay),size=2) +
    labs(x=x_label, title = total_title) +
    scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
    geom_line(df_total, mapping=aes(x={{var}}, y=n, group=1), size=1, color='Lime Green') +
    geom_point(df_total, mapping=aes(x={{var}}, y=n, group=1),size=2, color='Lime Green') +
    theme_light() +
    theme(legend.position = "none")
  return (p)
}

# 누적 막대그래프
bar_graph = function(df, var, x_label, total_title) {
  df_total = df %>% count({{var}})
  df_cnt = df %>% group_by({{var}}, delay) %>% summarize(count = n(), .groups = 'drop')

  p = ggplot(df_cnt, aes(x={{var}}, y=count, fill=delay)) +
    geom_bar(stat='identity', position='fill', width=0.7) +
    scale_color_manual(values = c("Dodger Blue", "Indian Red1")) +
    scale_fill_manual(values = c("Dodger Blue", "Indian Red1")) +
    theme_minimal() +
    #geom_text(aes(label = percent(count/sum(count))),
    #          position=position_fill(vjust=0.5)) +
    # 해당 코드를 통해 각 그룹이 전체에서 차지하는 %를 확인할 수 있지만,
          # 그림이 너무 복잡해져 생략하였음
    labs(x=x_label, title = total_title) +
    theme(legend.position = "none")
    theme_minimal()
```

```
  return (p)
}
```

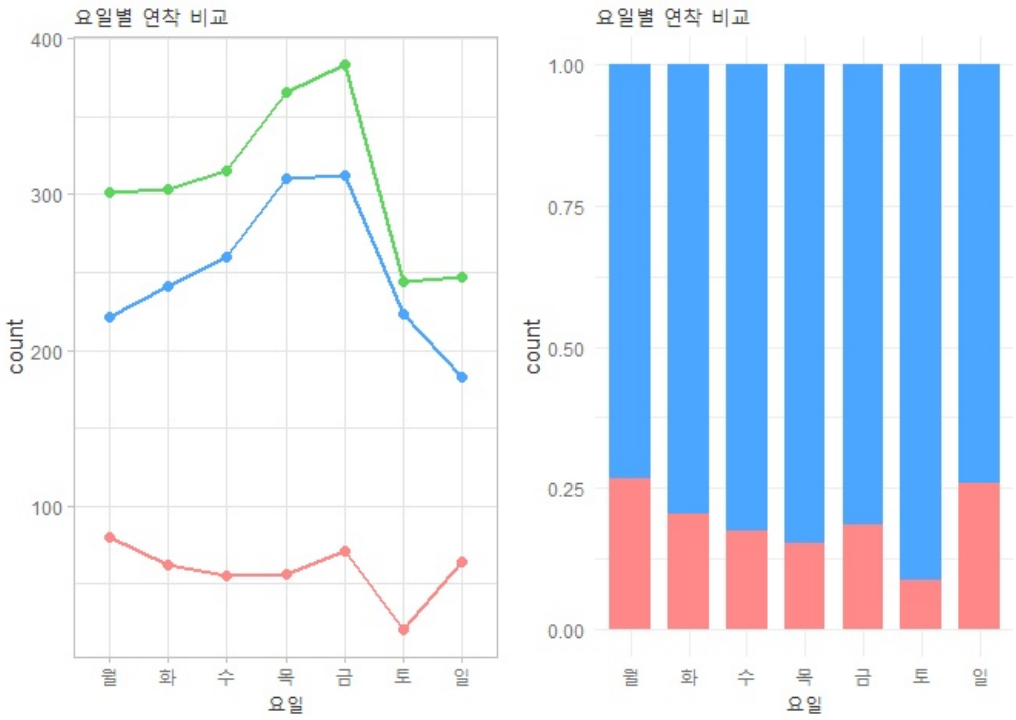
해당 문제를 해결함에 있어 연착 비율을 효과적으로 비교하기 위해 두가지 그래프를 사용하였다. 먼저 선그래프를 통해 전체 운행, 연착, 제시간 도착 수를 확인하였다. 다음으로 누적 막대그래프를 통해 모든 범주에 대한 y축을 동일하게 맞춤으로써 각 범주에서의 연착 비율을 비교하였다. 두 그래프는 계속 반복해서 그려야 하기 때문에 함수로 만들어 사용하였다.

(1) 요일별

```
# 요일별 연착 비율
p1_1 = line_graph(df_fly, dayweek, '요일', '요일별 연착 비교')
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
p1_2 = bar_graph(df_fly, dayweek, '요일', '요일별 연착 비교')
grid.arrange(p1_1, p1_2, ncol = 2)
```



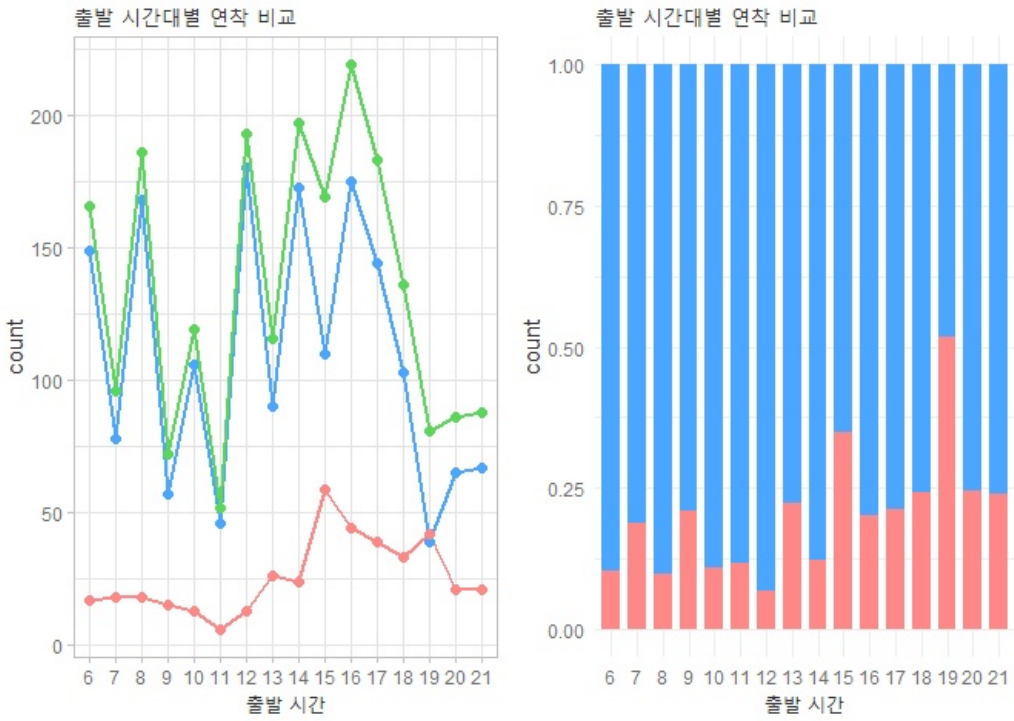
선 색상 설명

- 초록 : total
- 파랑 : ontime
- 빨강 : delayed

먼저 선그래프에서 초록선을 통해 비행기의 전체 운행 수를 알 수 있다. 따라서 그래프를 살펴보면 비행기는 주로 목요일과 금요일에 많이 운행되는 것을 알 수 있다. 또한 파란선을 통해 ontime의 비율을 살펴보면 역시 목요일과 금요일에 높다. 하지만 오른쪽 막대그래프를 살펴보면 토요일에서의 ontime 비율이 가장 높은 것을 확인할 수 있다. 이는 왼쪽 그래프에서 전체 운행수가 많기 때문에 당연하게 목요일과 금요일의 ontime이 높게 나타나는 것이며 실제 운행 수 대비 ontime의 비율이 높은 것을 보기 위해서는 오른쪽 그림과 함께 종합적으로 판단해야된다는 것을 알 수 있다. 따라서 왼쪽 그림을 통해 목요일과 금요일의 운행수가 높게 나타나며, 오른쪽 그림을 통해 월요일과 일요일에 연착이 되는 비율이 높다는 것을 알 수 있다.

(2) 출발 시간대별

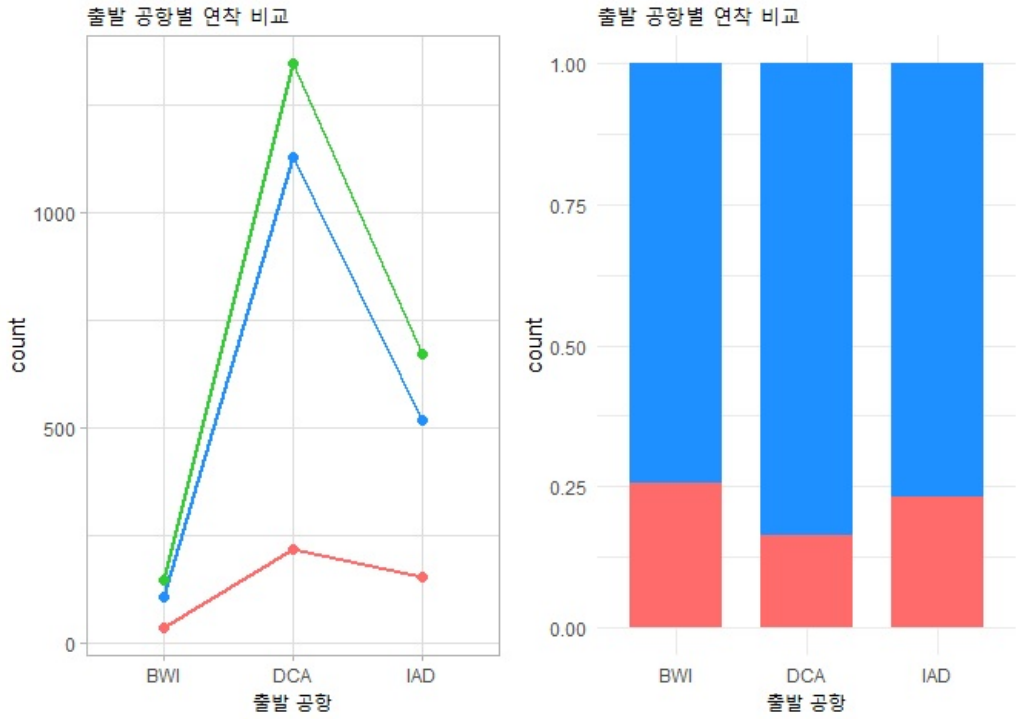
```
# 출발 시간대 별 연착 비율
p2_1 = line_graph(df_fly, deptime, '출발 시간', '출발 시간대별 연착 비교')
p2_2 = bar_graph(df_fly, deptime, '출발 시간', '출발 시간대별 연착 비교')
grid.arrange(p2_1, p2_2, ncol = 2)
```



먼저 왼쪽 그래프를 통해 전체 운행수를 비교해보면 8, 12, 14, 16시에 상대적으로 많이 운행하고 있음을 확인할 수 있다. 또한 9시부터 11시 사이와 19시 이후부터는 운행수가 많이 줄어들고 있음을 확인할 수 있다. 다음으로 왼쪽 그림을 보면, 15시와19시에서 연착 비율이 높음을 확인할 수 있다.

(3) 출발 공항별

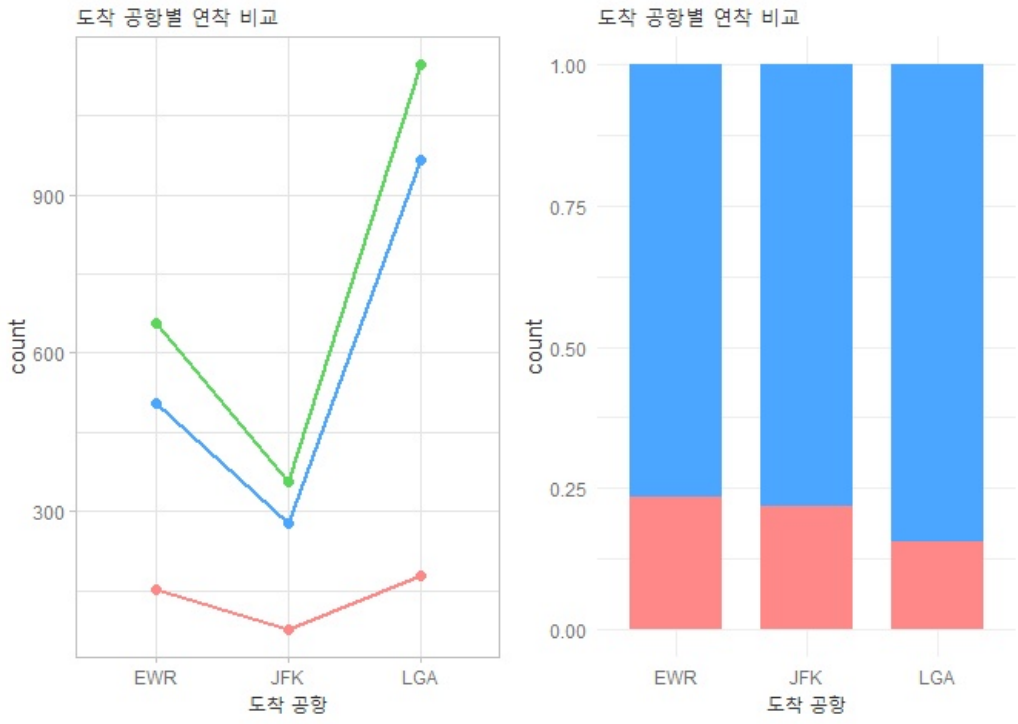
```
# 출발 공항별 연착 비율
p3_1 = line_graph(df_fly, origin, '출발 공항', '출발 공항별 연착 비교')
p3_2 = bar_graph(df_fly, origin, '출발 공항', '출발 공항별 연착 비교')
grid.arrange(p3_1, p3_2, ncol = 2)
```



먼저 왼쪽 그래프를 통해 항공사 DCA의 전체 운행수가 높은 것을 알 수 있으며, 연착수 역시 가장 높다는 것을 알 수 있다. 하지만 오른쪽 그래프에서는 오히려 DCA 항공사에서 연착 비율이 가장 낮다는 것을 알 수 있다. 따라서 DCA에서는 많은 고객들이 이용하지만 상대적으로 연착하는 비율은 낮다. 하지만 '출발 공항' 변수는 각 범주간의 연착 비율의 차이가 유의미하게 많이 나지 않는다고 판단된다.

(4) 도착 공항별

```
# 도착 공항별 연착 비율
p4_1 = line_graph(df_fly, dest, '도착 공항', '도착 공항별 연착 비교')
p4_2 = bar_graph(df_fly, dest, '도착 공항', '도착 공항별 연착 비교')
grid.arrange(p4_1, p4_2, ncol = 2)
```



먼저 왼쪽 그래프를 통해 LGA가 가장 많이 이용되는 공항임을 확인할 수 있다. 하지만 오른쪽 그래프를 통해 연착 비율은 가장 낮음을 확인할 수 있다. 하지만 '도착 공항' 변수 역시 각 범주간의 연착 비율의 차이가 유의미하게 많이 나지 않는다고 판단된다.

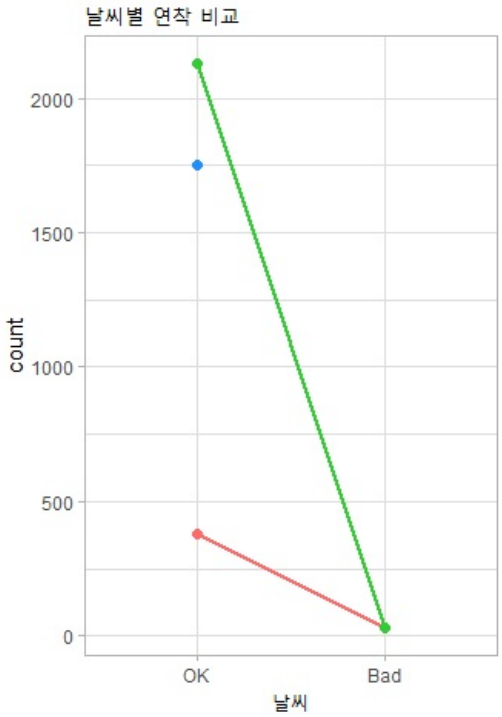
(5) 항공사별

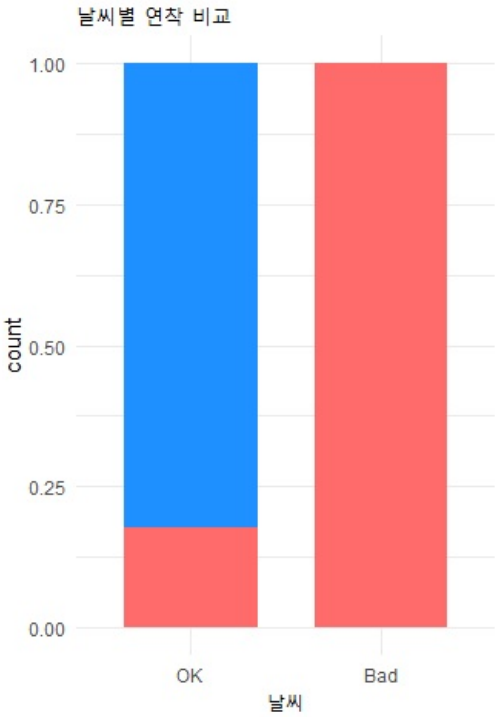
```
# 항공사별 연착 비율
p5_1 = line_graph(df_fly, carrier, '항공사', '항공사별 연착 비교')
p5_2 = bar_graph(df_fly, carrier, '항공사', '항공사별 연착 비교')
grid.arrange(p5_1, p5_2, ncol = 2)
```

먼저 전체 수를 비교해보았을때, DH이 가장 크며 DL, RU, US가 그 다음으로 비등하게 크다는 것을 알 수 있다. 하지만 오른쪽 그림을 통해 많이 이용되는 DH와 RU가 연착 비율이 상대적으로 높으며, DL과 US의 연착 비율이 낮음을 알 수 있다.

(6) 날씨별

```
# 날씨별 연착 비율
p6_1 = line_graph(df_fly, weather, '날씨', '날씨별 연착 비교')
p6_2 = bar_graph(df_fly, weather, '날씨', '날씨별 연착 비교')
grid.arrange(p6_1, p6_2, ncol = 2)
```





이때까지의 변수들과 다르게 날씨 변수와 같은 경우 LEVEL이 'Bad'이면 모두 연착 비율을 갖음을 확인할 수 있다. 이로써 날씨가 좋지 않은 경우는 전체 날에서 많지 않지만 날씨가 안 좋을 경우 연착에 큰 영향을 미친다는 것을 알 수 있다.

[전체 고찰]

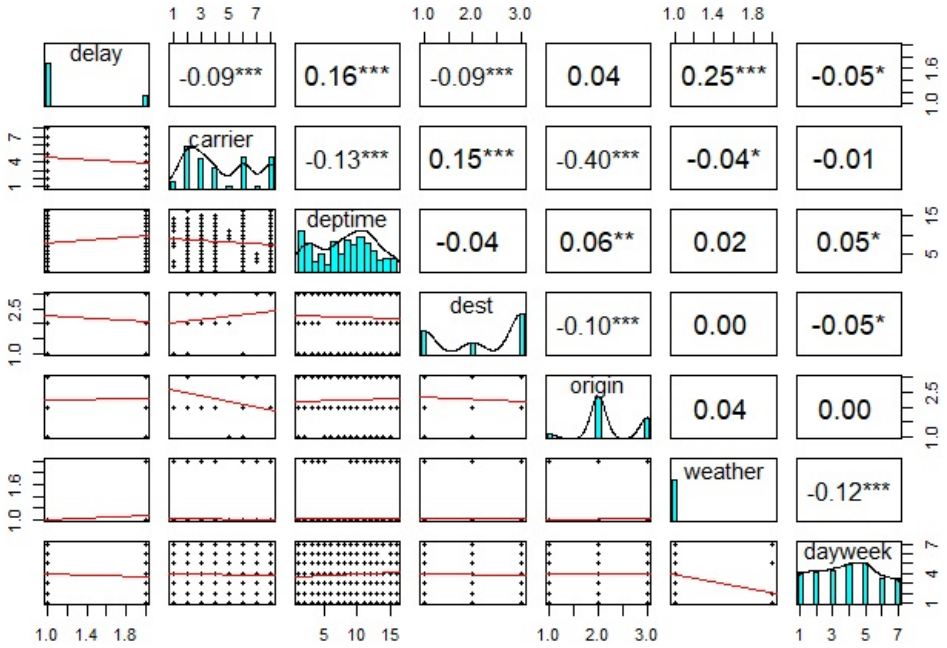
이렇게 여러가지 변수들의 연착 비율을 통해 각 범주에서 비율의 차이가 많이 나는 변수가 존재하기도 하지만 각 범주에서 크게 차이나지 않은 변수도 존재함을 확인할 수 있었다.

3번

7개의 모든 변수들 간의 상관관계를 시각화해보자. 어떤 특성을 관찰할 수 있는가?

[7개의 변수 모두 비교]

```
pairs.panels(df_fly, lm=TRUE, ellipses=FALSE, rug=FALSE,
             stars=TRUE)
```



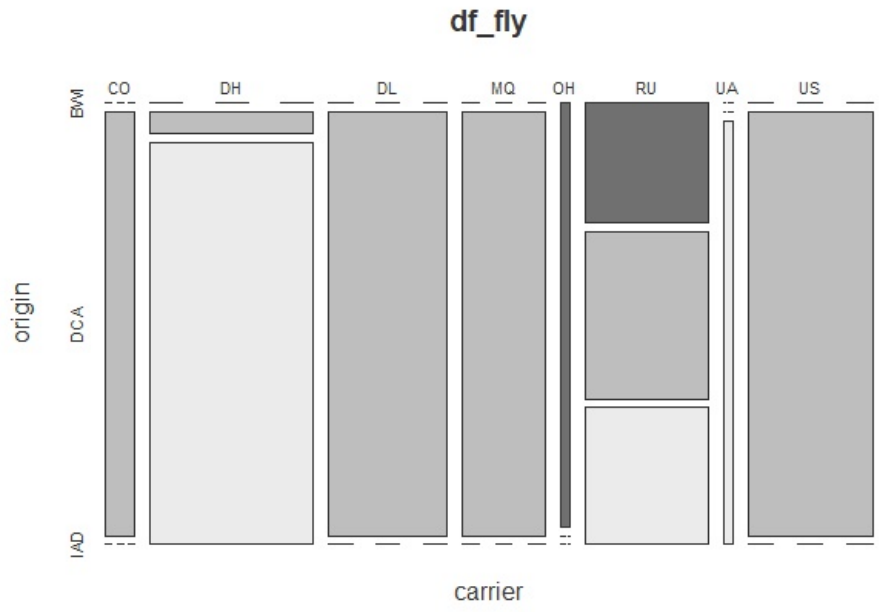
먼저 delay와 다른 변수들을 살펴보면, 위에서 각 범주 간의 차이가 많이 났던 weather과 같은 경우 높은 delay 변수와 높은 상관관계를 보인다. 또한 다른 변수들에 비해 각 범주 간의 유의미한 차이가 보이지 않는다고 판단했던 출발공항(origin)과 도착공항(dest)는 다른 변수에 비해 delay 변수와 낮은 상관관계를 갖음을 알 수 있다.

다음으로 나머지 변수들을 살펴보면, carrier 변수와 weather 변수 간의 상관관계가 유의미하게 높으며, dest와 weather의 상관관계가 아주 낮은 것을 확인할 수 있었다.

하지만 원래 factor였던 변수들을 numeric으로 고려하여 상관관계를 고려한 방법은 완벽하게 신뢰하기 어렵다고 생각이 되었다. 따라서 범주형 간의 상관관계를 시각화하여 비교할 수 있는 모자이크 플롯을 통해 추가적으로 비교해보고자 한다.

[모자이크 플롯 시각화]

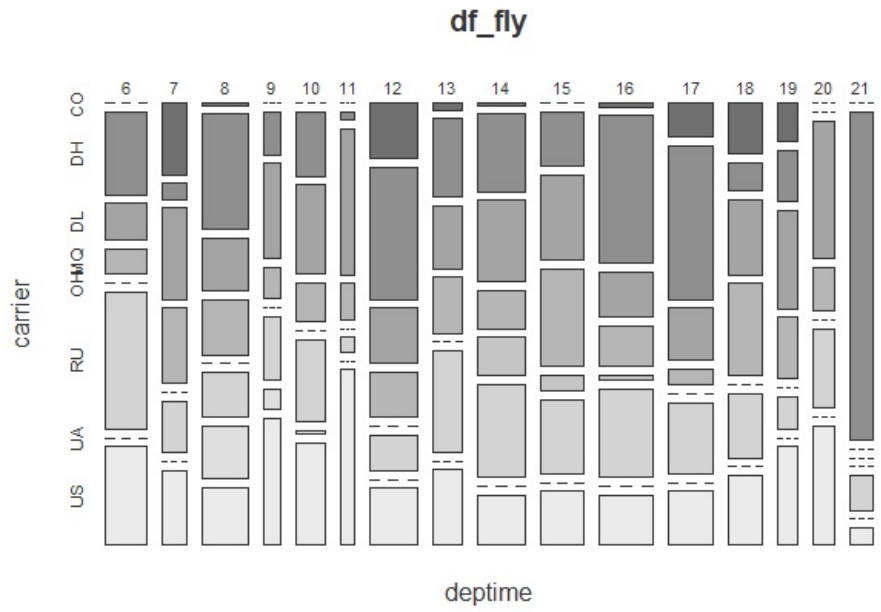
```
mosaicplot(~carrier+origin, data=df_fly, color=TRUE)
```



먼저 위의 그림에서 상관관계가 높다고 확인된 origin 변수와 carrier 변수를 비교하였다. 각 출발 공항 별로 차지하고 있는 항공사의 비율

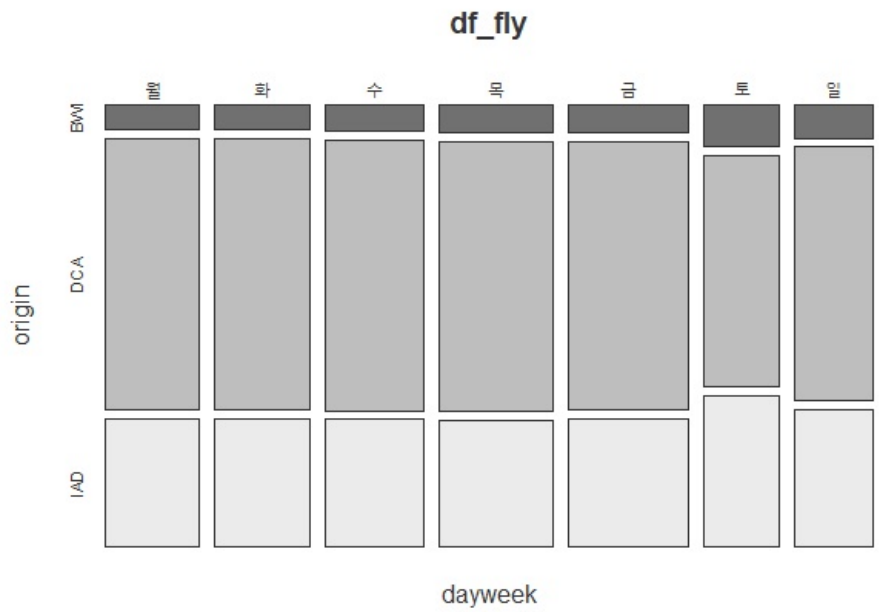
이 상이하게 다름을 확인할 수 있다.

```
mosaicplot(~deptime+carrier, data=df_fly, color=TRUE)
```



또한 위의 변수들보다는 상관관계가 작지만 상대적으로 높게 구해진 carrier과 deptime 변수들도 추가로 비교해보았다. 그 결과 위의 그래프 처럼 차이가 크지 않지만 육안으로 판단할 수 있을 정도로 차이가 존재한다는 것을 알 수 있다.

```
mosaicplot(~dayweek+origin, data=df_fly, color=TRUE)
```



추가적으로 상관관계가 높지 않다고 구해진 변수들의 분포를 확인해보고 싶어 상관관계가 0이었던 origin과 dayweek를 비교해보았다. 그 결과 요일별로 출발 공항의 분포가 거의 비슷하다는 것을 확인하였다. 따라서 각 범주의 그룹에 속하는 수들이 비슷하면 상관관계가 낮다고 판단되며, 각 그룹에 속한 개수가 범주에 따라 차이가 클수록 상관관계가 높다고 판단된다.

4번

데이터셋을 70:30 비율로 training set과 test set으로 분할하자. 이때 stratified sampling을 활용하여 두 set에서 delay 변수의 분포가 크게 차이가 없도록 분할하자

```
# 데이터 셋을 delay 분포에 맞게 분할
set.seed(1); split = initial_split(df_fly, prop=0.7, strata='delay')
train = training(split)
test = testing(split)

# 각 셋의 연착 비율이 비슷하게 분할 되었는지 확인
# y 축의 길이를 동일하게 설정함으로써 비율 확인
p1 = ggplot(train, aes(x=delay)) +
  geom_bar(fill='#66a182') +
  labs(title='train') +
  theme_minimal()


p2 = ggplot(test, aes(x=delay)) +
  geom_bar(fill='#66a182') +
  labs(title='test') +
  theme_minimal()

grid.arrange(p1, p2, ncol = 2)
```

5번

데이터시각화로부터 weather 변수가 “Bad” 인 경우에는 항상 항공기가 연착되는 것을 관찰할 수 있다. 따라서 weather가 Bad이면 항공기가 연착되고, weather가 OK일 경우 항공기가 연착되지 않는 것으로 예측하는 단순한 모델을 baseline model이라 하자. Training set에 대해 baseline model을 적용했을 때의 confusion matrix를 계산해보자.

```
# delay변수와 weather 변수만 추출 후 factor 변환
df_con = df_fly[, c(6, 1)]
df_con$delay = as.factor(as.numeric(df_fly$delay))
df_con$weather = as.factor(as.numeric(df_fly$weather))
```

```
cm_5 = confusionMatrix(df_con$weather, df_con$delay)
cm_5
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2
##          1 1750  377
##          2    0   32
##
##                Accuracy : 0.8254
##                  95% CI : (0.8087, 0.8412)
##     No Information Rate : 0.8106
##     P-Value [Acc > NIR] : 0.04076
##
##                   Kappa : 0.121
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 1.00000
##             Specificity : 0.07824
##          Pos Pred Value : 0.82276
##          Neg Pred Value : 1.00000
##              Prevalence : 0.81056
##          Detection Rate : 0.81056
##    Detection Prevalence : 0.98518
##       Balanced Accuracy : 0.53912
##
##        'Positive' Class : 1
##
```

먼저 weather의 변수에서 'Bad'의 수는 32였다. 이는 곧 연착이라고 예측되며 TN은 32가 된다. 하지만 나머지 연착은 모두 예측하지 못하여 FP는 377을 갖게 되었다. 따라서 confusionmatrix 결과에서 specificity(TN/(TN+FP))는 32/(377+32)로 아주 낮은 성능을 얻게 된다. 만약 날씨 변수에서 날씨가 안 좋은(Bad) 비율이 더 높다면 specificity는 올라가게 될 것이다.

6번

Training set을 대상으로, 연착여부(delay)를 나머지 모든 변수를 사용하여 예측하기 위한 logistic regression model을 수립해보자.

6.1번

변수 deptime19의 regression coefficient에 대한 추정값은 얼마인가? 이 추정값을 바탕으로 출발 시각이 19시대인 항공기에 대해서 어떠한 해석을 할 수 있는가? (Hint: 범주형 변수 deptime을 model에 추가할 때 deptime6을 제외한 deptime7 ~ deptime21에 대한 dummy 변수가 만들어진다.)

```
# 로지스틱 모델 생성
# 들어가는 변수가 모두 범주형이기 때문에 scale은 따로 진행하지 않음
logi_model = glm(delay~., family='binomial', data=train)
summary(logi_model)
```

```
##
## Call:
## glm(formula = delay ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6410  -0.6476  -0.4568  -0.2298   2.9608
```

```
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.079312   0.614511  -0.129 0.897306
## carrierDH    -0.265515   0.559264  -0.475 0.634959
## carrierDL    -0.801805   0.511844  -1.567 0.117231
## carrierMQ     0.451104   0.492555   0.916 0.359749
## carrierOH    -1.833673   0.995881  -1.841 0.065584 .
## carrierRU    -0.508312   0.435079  -1.168 0.242677
## carrierUA    -0.468049   0.959052  -0.488 0.625526
## carrierUS    -1.503518   0.536240  -2.804 0.005050 **
## deptime7      0.643284   0.464886   1.384 0.166436
## deptime8     -0.539483   0.516207  -1.045 0.295981
## deptime9      1.384167   0.499032   2.774 0.005542 **
## deptime10    -0.302257   0.498781  -0.606 0.544521
## deptime11     0.904276   0.582807   1.552 0.120761
## deptime12    -0.518047   0.462189  -1.121 0.262350
## deptime13     0.630576   0.441250   1.429 0.152985
## deptime14     0.025443   0.414957   0.061 0.951109
## deptime15     1.811871   0.389478   4.652 3.29e-06 ***
## deptime16     0.832359   0.394906   2.108 0.035054 *
## deptime17     0.931721   0.381308   2.443 0.014546 *
## deptime18     0.988167   0.419242   2.357 0.018422 *
## deptime19     2.190687   0.447553   4.895 9.84e-07 ***
## deptime20     1.724618   0.452214   3.814 0.000137 ***
## deptime21     0.284414   0.471402   0.603 0.546285
## destJFK      -0.528154   0.337167  -1.566 0.117245
## destLGA      -0.019062   0.333761  -0.057 0.954455
## originDCA    -1.499723   0.381175  -3.934 8.34e-05 ***
## originIAD    -0.679665   0.366917  -1.852 0.063973 .
## weatherBad   17.641842 516.765061   0.034 0.972766
## dayweek화    -0.553007   0.264751  -2.089 0.036728 *
## dayweek수    -0.640062   0.266153  -2.405 0.016178 *
## dayweek목    -0.697716   0.255843  -2.727 0.006389 **
## dayweek금    -0.519302   0.245454  -2.116 0.034372 *
## dayweek토    -1.431100   0.348849  -4.102 4.09e-05 ***
## dayweek일     0.001517   0.262047   0.006 0.995380
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1466.2  on 1510  degrees of freedom
## Residual deviance: 1198.1  on 1477  degrees of freedom
## AIC: 1266.1
##
## Number of Fisher Scoring iterations: 15
```

[Logistic regressio의 Coefficient 해석 방법]

로지스틱 회귀 모형에서는 연속형 변수가 들어갈 경우 변수가 하나가 추가가 되며, 범주형 변수가 들어갈 경우 해당 변수의 level-1개의 변수가 모델의 결과에 출력된다. 따라서해당 모형에서는 변수가 연속형인지, 범주형인지에 따라 계수를 다르게 해석해야 한다. 두 가지 해석 방법을 살펴보면, 아래와 같이 다르게 해석할 수 있다.

- 연속형의 coefficient : 변수가 한 단위 증가했을 때 log(odds)의 증가량
- 범주형의 coefficient : rank2는 rank1에서 rank2로 바뀌었을 때, log(odds)의 변화량

따라서, deptime19의 coefficient를 살펴보면, deptime6에서 deptime19로 바뀌었을 때, log(odds)의 변화가 2.19만큼 증가한다고 해석할 수 있다. 따라서 deptime19의 coefficient가 양수이기 때문에deptime19가 deptime6에 비해 더 유의한 영향을 준다고 판단할 수 있다.

[odds ratio을 exp로 변환]

위에서 설명했듯이 로지스틱 회귀 모형의 coefficient를 통해 로그 오즈비가 얼마나 변화하는 지를 알 수 있었다. 하지만 로그 오즈비로 증감을 해석하는 것은 직관적이지 못하다. 따라서 coefficient에 exp 함수를 취함으로써 다르게 해석하여 비교해보고자 한다. 아래는 각 방법에 대한 해석 방법의 차이이다.

- 일반 coefficient : 해당 변수의 값이 한 단위 증가할 때 로그 오즈 비가 얼마나 변화하는가
- exp(coefficient) : 해당 독립 변수가 한 단위 증가할 때 종속변수의 오즈비가 몇 배 증가하는가

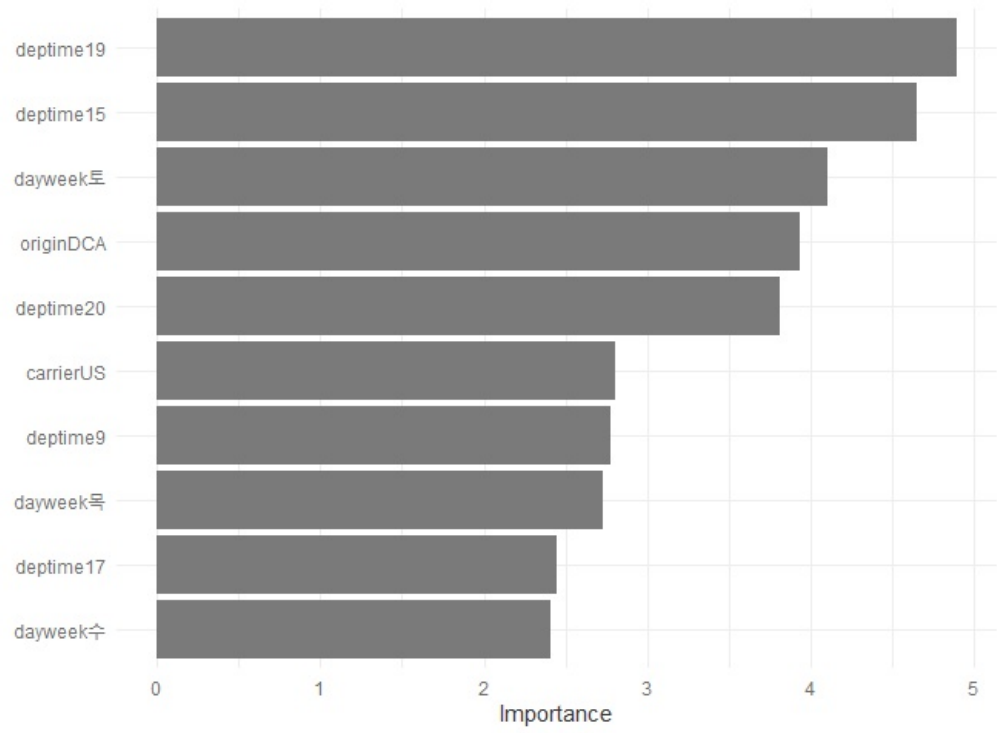
```
exp(coef(logi_model))
```

```
##  (Intercept)    carrierDH    carrierDL    carrierMQ    carrierOH    carrierRU
## 9.237514e-01 7.668107e-01 4.485188e-01 1.570044e+00 1.598254e-01 6.015102e-01
##    carrierUA    carrierUS     deptime7     deptime8     deptime9    deptime10
## 6.262226e-01 2.223467e-01 1.902719e+00 5.830499e-01 3.991500e+00 7.391481e-01
##    deptime11    deptime12    deptime13    deptime14    deptime15    deptime16
## 2.470143e+00 5.956826e-01 1.878692e+00 1.025769e+00 6.121894e+00 2.298735e+00
##    deptime17    deptime18    deptime19    deptime20    deptime21      destJFK
## 2.538875e+00 2.686305e+00 8.941350e+00 5.610378e+00 1.328983e+00 5.896926e-01
##      destLGA    originDCA    originIAD   weatherBad    dayweek화    dayweek수
## 9.811185e-01 2.231920e-01 5.067869e-01 4.589388e+07 5.752178e-01 5.272598e-01
##    dayweek목    dayweek금    dayweek토    dayweek일
## 4.977209e-01 5.949359e-01 2.390459e-01 1.001518e+00
```

위에서 구한 로지스틱 회귀 모형의 coefficient에 exp를 취한 결과는 위와 같다. 따라서 deptime19는 deptime6보다 odds가 8.941배 높다고 판단할 수 있다.

[변수의 영향력 평가]

```
# 변수의 영향력 시각화
vip(logi_model) + theme_minimal()
```




```
# 계수들의 크기 비교
sort(coef(logi_model), decreasing = TRUE)
```

```
##   weatherBad    deptime19    deptime15    deptime20     deptime9    deptime18
```

```
## 17.641842322  2.190686614  1.811871471  1.724618140  1.384167186  0.988166635
##    deptime17    deptime11    deptime16     deptime7    deptime13    carrierMQ
##  0.931721183  0.904275861  0.832359017  0.643283771  0.630575715  0.451103544
##    deptime21    deptime14    dayweek일      destLGA  (Intercept)    carrierDH
##  0.284413719  0.025442832  0.001517199 -0.019062032 -0.079312329 -0.265515309
##    deptime10    carrierUA    carrierRU    deptime12    dayweek금      destJFK
## -0.302257023 -0.468049420 -0.508311741 -0.518047352 -0.519301650 -0.528153838
##     deptime8    dayweek화    dayweek수    originIAD    dayweek목    carrierDL
## -0.539482567 -0.553006516 -0.640061916 -0.679664688 -0.697715764 -0.801804775
##    dayweek토    originDCA    carrierUS    carrierOH
## -1.431099595 -1.499722839 -1.503517532 -1.833673337
```

먼저 vip 그래프를 통해 영향력이 높은 변수들을 확인할 수 있었다. 하지만 밑의 계수를 큰 값으로 정렬하면, deptime19가 아닌 weatherbad 변수의 계수가 더 큰 것을 확인할 수 있다. 이는 계수들이 각 더미변수로 구성되고 변수내에서의 계수를 비교하는 것이기 때문에 계수의 값이 전체 변수들의 영향력을 판단할 수는 없다고 추측할 수 있다.하지만 같은 변수 내의 level끼리는 계수가 클수록 모델에 미치는 영향력이 높은 것을 확인할 수 있다.

6.2번

날씨에 문제가 없는 목요일 15시에 IAD에서 출발하여 EWR로 도착한 Delta 항공기가 연착될 확률은 얼마로 예측되는가?

```
# 문제에서 주어진 조건에 맞게 새로운 데이터 생성
new_data = data.frame(carrier='DL', deptime="15", dest='EWR', origin='IAD', weather='OK', dayweek='목')

# 확률 예측
pred = predict(logi_model, new_data, type='response')

# 결과 출력
as.numeric(pred)
```

```
## [1] 0.3901632
```

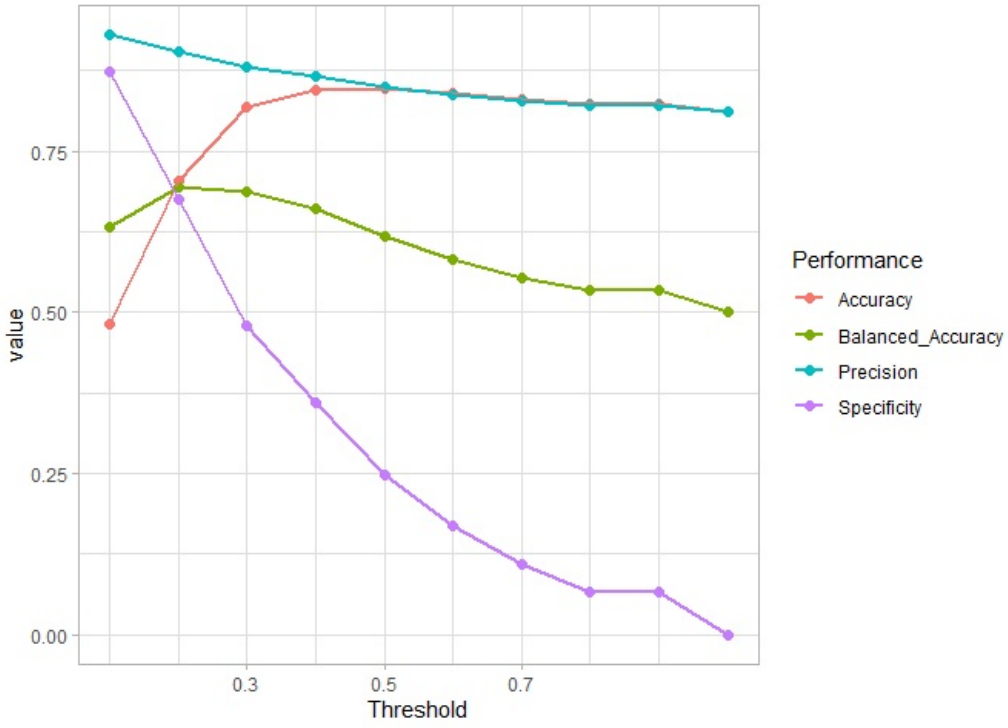
문제에서 주어진 조건에 맞게 '날씨에 문제가 없는 목요일 15시에 IAD에서 출발하여 EWR로 도착한 Delta 항공기'가 연착될 확률은 0.39로 계산되었다.

6.3번

Threshold 에 대해서 각각 training set에 대한 confusion matrix를 계산해 보자. 어떠한 경향을 관찰할 수 있는가?

```
# 각 모델의 결과값을 저장할 빈 데이터 프레임 생성
df_result = data.frame(Threshold=numeric(), Accuracy=numeric(),
                       Sensitivity=numeric(), Specificity=numeric())

# 추가로 0.1~1까지의 Threshold 비교
# 반복문과 데이터프레임 병합을 통해 각 Threshold의 결과 비교
for (i in 1:10){
  train_prod = predict(logi_model, train, type='response')
  train_pred = rep('ontime', 1511)
  train_pred[train_prod > i/10] = 'delayed'
  table(train_pred)
  cm = confusionMatrix(factor(train_pred, levels=c('ontime', 'delayed')),
                       train$delay, positive='ontime', mode = "everything")
  df_result = rbind(df_result, data.frame(Threshold=i/10, Accuracy=cm$overall["Accuracy"],
                       Specificity=cm$byClass["Specificity"], Balanced_Accuracy = cm$byClass['Balanced
        Accuracy'],
                       Precision = cm$by['Precision']))
}
```

```
# plot을 그리기 위해 데이터프레임 형태 변형
df_gather = gather(df_result, Performance, value, Accuracy, Specificity, Balanced_Accuracy, Precision)

# Threshold 변화에 따른 각 performance 비교
ggplot(df_gather, aes(x=Threshold, y=value, color=Performance)) +
  geom_point(size=2) +
  geom_line(size=1)+
  theme_light() +
  scale_x_continuous(breaks=c(0.3, 0.5, 0.7))
```



0.1부터 1까지의 Threshold의 변화에 따른 각 performance를 비교해보았다. 그 결과 Accuracy는 Threshold가 커질수록 증가하는 양상을 보이지만 Specificity는 크게 감소하는 양상을 보였다. 이는 타겟변수의 ontime과 delayed의 데이터 분포가 불균형하기 때문에 발생한다. 따라서 추가적으로 'Balanced Accuracy'라는 지표를 같이 비교해보았다. Balanced Accuracy란 불균형한 클래스 비율을 가진 데이터 셋에서 모델의 예측 성능을 평가하기 위한 지표 중 하나이다. 클래스 간 샘플 수의 차이를 고려하여 각 클래스의 recall값을 가중 평균한 값으로 계산한다. 이 지표와 같은 경우 Threshold가 0.2일때 가장 높은 것을 확인할 수 있다.

하지만 공항의 경우 고객들에게 연착이 되는데 되지 않을 것이라고 잘못 전달하는 것을 줄이기 위해서는 FP를 줄여야한다. 즉 연착이라고 예측한 경우 중에서 실제 연착인 비율을 높이거나 제시간에 도착할 것이라고 예측한 경우 중에서 실제로 제시간에 도착한 비율을 높여야 한다. 이를 위해서는 Specificity나 Precision값을 높여야한다. 이처럼 모델의 성능을 평가하는 기준은 여러가지이며 전체 성능이 중요한지, 잘못 전달을 막는 것이 중요한지, 각각의 목표에 맞게 평가 지표를 선정하고 Threshold를 결정해야할 것이다.

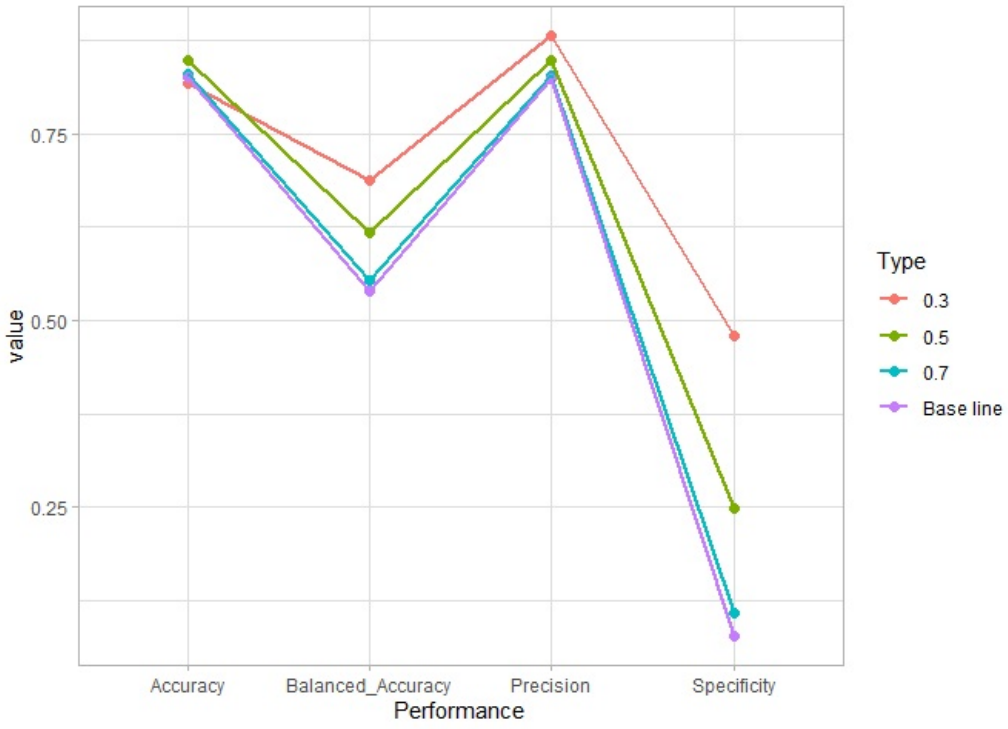
(10번 문제에서 다른 모델들은 Accuracy를 기준으로 best model을 선정하였다. 따라서 logistic model은 Accuracy가 가장 높은 0.5 Threshold를 가진 모델을 사용하겠다.)

6.4번

위의 결과를 바탕으로 Baseline model과 logistic regression model의 성능을 비교해보자.

```
# Threshold가 0.3, 0.5, 0.7인 데이터만 비교하기 위해 추출
df_com = subset(df_gather, Threshold==0.3 | Threshold==0.5 | Threshold ==0.7)
df_com = df_com %>% mutate(Threshold=as.character(Threshold))
colnames(df_com)[1] = 'Type'

# baseline의 성능 데이터 추가
df_com = rbind(df_com, data.frame(Type=c('Base line', 'Base line', 'Base line', 'Base line'),
                                  Performance=c('Accuracy','Balanced_Accuracy','Specificity', 'Precision'),
                                    value=c(cm_5$overall["Accuracy"],
                                            cm_5$byClass["Balanced Accuracy"],
```

```
                                        cm_5$byClass["Specificity"],
                                        cm_5$byClass["Precision"])))

# Threshold가 0.3, 0.5, 0.7인 데이터와 baseline의 각 성능 비교
ggplot(df_com, aes(x=Performance, y=value, color=Type, group=Type)) +
  geom_point(size=2) +
  geom_line(size=1)+
  theme_light()
```

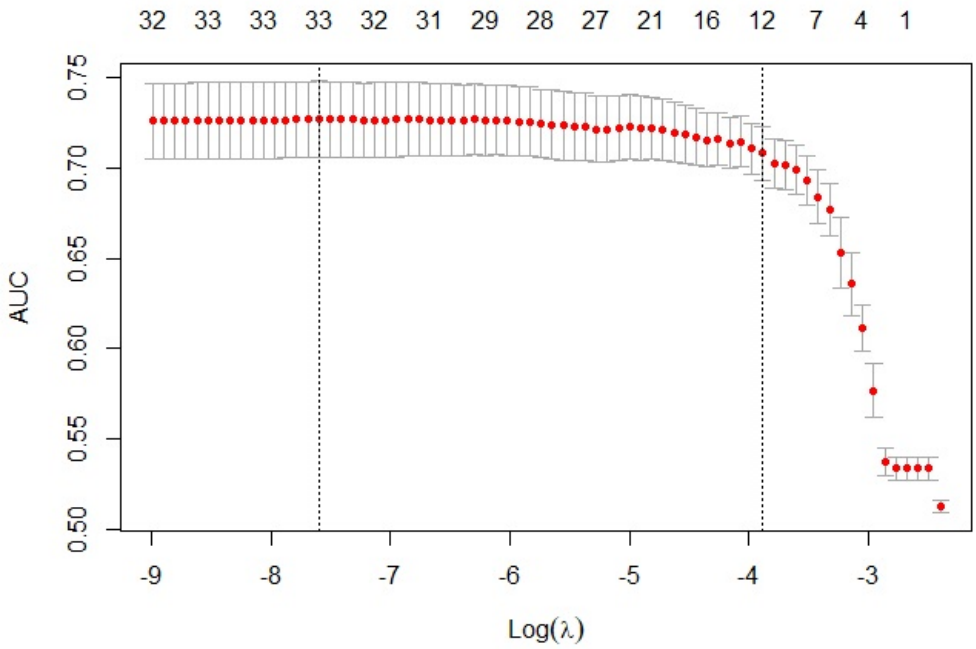


위의 그림을 통해 base line 모델과 같은 경우 평균적으로 모든 평가지표에 대한 성능이 안좋으며, Specificity가 매우 낮은 것을 확인할 수 있다. 이는 모델이 ontime이라고 예측했는데 실제로는 연착이 경우가 많다는 것을 의미한다. 하지만 모델의 불균형으로 ontime의 데이터 수가 더 많기 때문에 다른 모델들과 Accuracy에서는 큰 차이가 없다는 것을 알 수 있다.

7번

Training set을 대상으로 Lasso regression을 적용하여 logistic regression model을 수립해보자. CV의 결과 바탕으로 모델에 포함되는 feature의 수와 예측정확도를 모두 고려했을 때 가장 적합한 모델을 선택하자.

```
# 먼저 matrix 형태로 데이터 변환
trainx = model.matrix(delay~., data=train)[, -1]
trainy = train$delay

testx = model.matrix(delay~., data=test)[, -1]
testy = test$delay

# cv.glmnet은 자동으로 스케일링됨 (standardize=TRUE)
set.seed(1); cv_lasso = cv.glmnet(x=trainx, y=trainy, alpha=1, family='binomial',
                                  type.measure='auc', nfolds=10)
plot(cv_lasso)
```



위의 그림은 람다 값에 따른 AUC값의 변화이다. 왼쪽 점선을 best model에서의 람다값과 선택된 개수를 확인할 수 있으며, 오른쪽 점선을 통해 one-standard-error-rule을 적용한 경우의 결과를 알 수 있다.

7.1번

어떠한 기준으로 모델을 선택하였으며, 최종적으로 모델에 어떠한 변수들이 포함되었는가?

위의 그림을 통해 best model과 simple 모델의 성능은 크게 차이나지 않지만 변수는 33개에서 12개로 크게 차이난다는 것을 알 수 있었다. 따라서 성능은 크게 차이나지 않지만 적은 변수로 더 해석하기 용이한 one-standard-error-rule 적용 model을 최종 모델로 선정하였다.

[선택된 Coefficient 확인]

```
# best model
lambda_min = cv_lasso$lambda.min
coef_1 = coef(cv_lasso, s=lambda_min)

# one standard error rule model
lambda_sim = cv_lasso$lambda.1se
coef_2 = coef(cv_lasso, s=lambda_sim)

# 두 개의 결과 비교
cbind(coef_1, coef_2)
```

```
## 34 x 2 sparse Matrix of class "dgCMatrix"
##                       s1          s1
## (Intercept) -0.32497552 -1.32975517
## carrierDH   -0.02545336  .
## carrierDL   -0.62429571 -0.13475775
## carrierMQ    0.59479839  0.23380861
## carrierOH   -1.47228556  .
## carrierRU   -0.27827564  .
## carrierUA   -0.16073768  .
## carrierUS   -1.31346765 -0.52369804
```

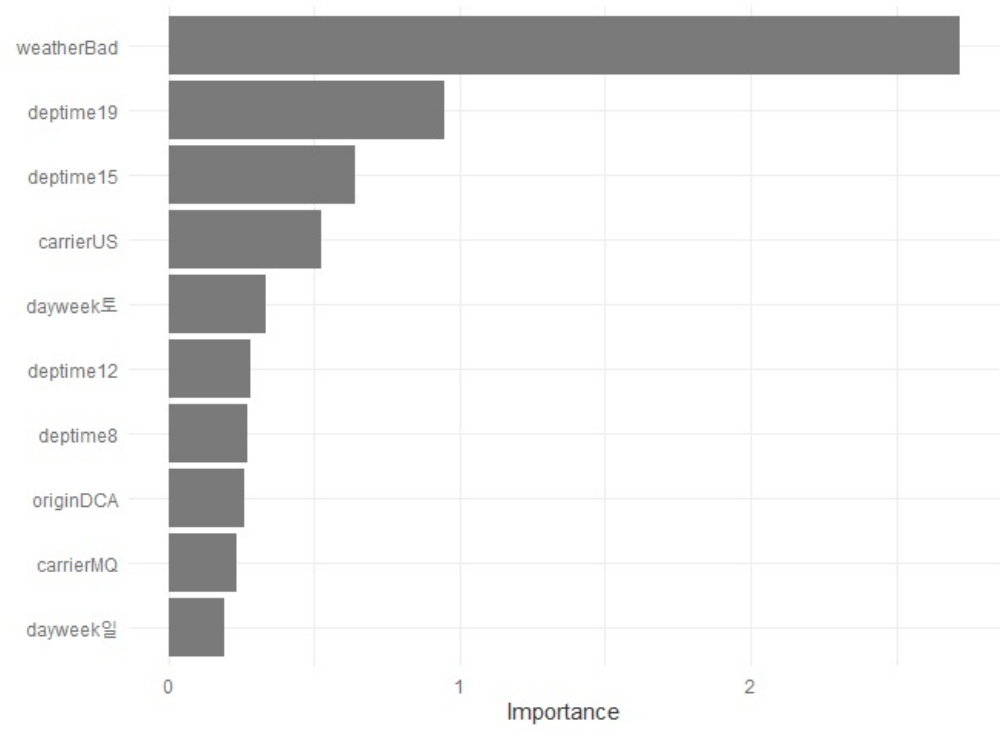
```
## deptime7     0.55567641  .
## deptime8    -0.63486119 -0.26830761
## deptime9     1.22187920  .
## deptime10   -0.34596669  .
## deptime11    0.77569243  .
## deptime12   -0.57638126 -0.28067596
## deptime13    0.49343676  .
## deptime14   -0.05057884  .
## deptime15    1.67542232  0.63854633
## deptime16    0.69487003  .
## deptime17    0.83540790  .
## deptime18    0.88549705  .
## deptime19    2.07780622  0.94538164
## deptime20    1.56697560  0.05484067
## deptime21    0.15986369  .
## destJFK     -0.52163539  .
## destLGA     -0.03795347  .
## originDCA   -1.32403247 -0.25933263
## originIAD   -0.60285974  .
## weatherBad   6.60770812  2.72120443
## dayweek화   -0.50600495  .
## dayweek수   -0.59277811  .
## dayweek목   -0.64773262  .
## dayweek금   -0.47205069  .
## dayweek토   -1.37364790 -0.33636693
## dayweek일    0.04214630  0.18893012
```

위의 표는 best model에서의 선택된 변수들과 simple model에서의 선택된 변수들이다. 33개 중 많은 변수들이 제외된 것을 확인할 수 있다. simple model에서 남은 변수들은 총 12 개이며, “carrier_DL, carrier_MQ, carrier_US, deptime_8, deptime_12, deptime_15, deptime_19, deptime_20, origin_DCA, weather_Bad, dayweek_토, dayweek_일” 등이 있다.

[변수 중요도 비교]

계수의 크기를 통해 변수의 중요도를 알 수 있지만 모든 변수에서의 중요도를 구하기 위해서 vip를 실행하여 비교해보았다.

```
vip(cv_lasso) + theme_minimal()
```

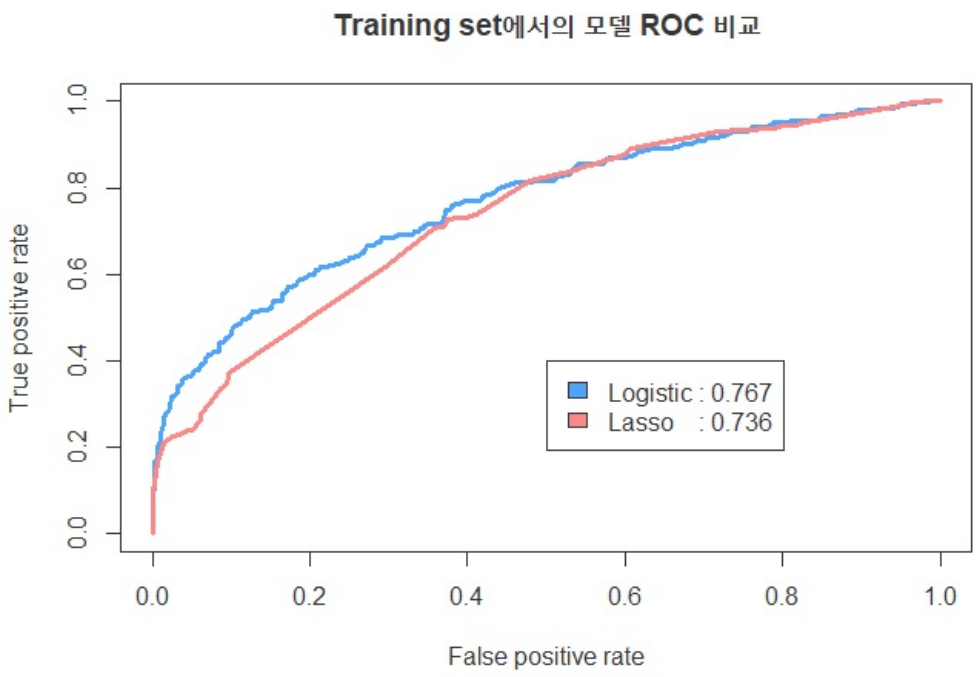


7.2번

기본 logistic regression model과 Lasso를 적용한 logistic regression model의 성능을 나타내는 ROC Curve warning=FALSE,를 하나의 그래프로 시각화하고, AUC값을 비교해 보자. Lasso regression의 효과가 있다고 말할 수 있는가? (training set과 test set에 대해서 각각 비교해보자.)

[train set]

```
# training
prob = predict(logi_model, train, type='response')
pred = prediction(prob, train$delay, c('ontime', 'delayed'))
perf = performance(pred, measure='tpr', x.measure = 'fpr')
auc = performance(pred, measure='auc')
plot(perf, col='Dodger Blue', lwd=3, main='Training set에서의 모델 ROC 비교')
auc = performance(pred, measure='auc')
logi_auc = round(auc@y.values[[1]][1], 3)

prob = predict(cv_lasso, newx=trainx, s=lambda_sim, type='response')
pred = prediction(prob, train$delay, c('ontime', 'delayed'))
perf = performance(pred, measure='tpr', x.measure = 'fpr')
auc = performance(pred, measure='auc')
lasso_auc = round(auc@y.values[[1]][1], 3)
plot(perf, col='Indian Red1', lwd=3, add=TRUE)

s1 = sprintf('Logistic : %g', logi_auc)
s2 = sprintf('Lasso    : %g', lasso_auc)

legend(0.5, 0.4, legend=c(s1,s2), fill=c('Dodger Blue','Indian Red1'))
```

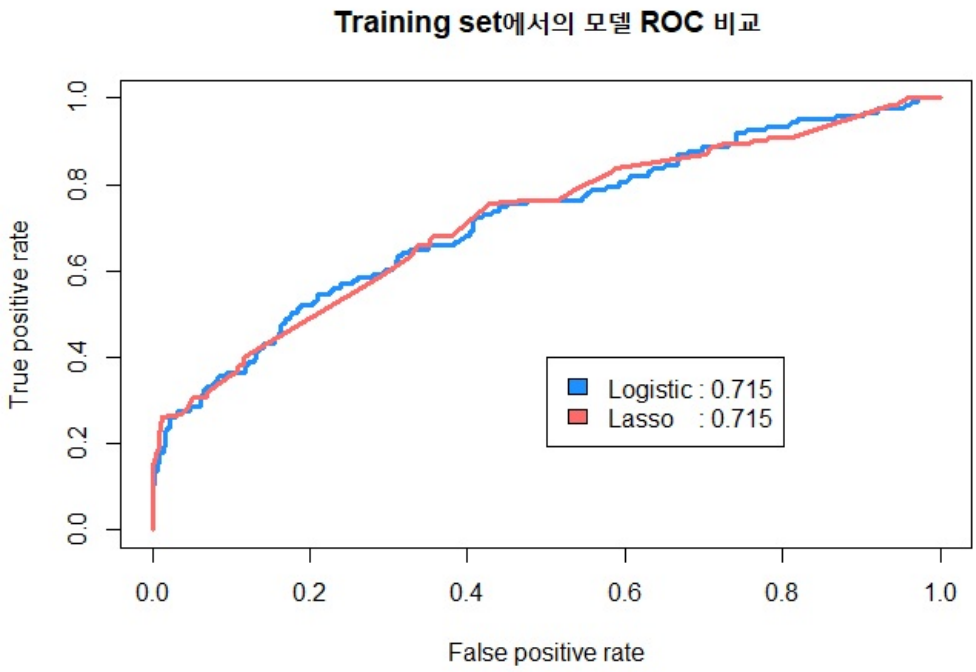


먼저 train set을 사용하여 Logistic 모델과 Lasso logistic 모델을 비교한 결과 Logistic 모델 AUC값이 더 높기 때문에 더 성능이 좋다고 판단할 수 있다. 이는 Logistic 모델이 더 많은 변수를 갖고 있기 때문이라고 판단된다.

[test set]

```
# test
prob = predict(logi_model, test, type='response')
```

```
pred = prediction(prob, test$delay, c('ontime', 'delayed'))
perf = performance(pred, measure='tpr', x.measure = 'fpr')
auc = performance(pred, measure='auc')
plot(perf, col='Dodger Blue', lwd=3,
     main='Training set에서의 모델 ROC 비교')
auc = performance(pred, measure='auc')
logi_auc = round(auc@y.values[[1]][1], 3)

prob = predict(cv_lasso, newx=testx, s=lambda_sim, type='response')
pred = prediction(prob, test$delay, c('ontime', 'delayed'))
perf = performance(pred, measure='tpr', x.measure = 'fpr')
auc = performance(pred, measure='auc')
lasso_auc = round(auc@y.values[[1]][1], 3)
plot(perf, col='Indian Red1', lwd=3, add=TRUE)

s1 = sprintf('Logistic : %g', logi_auc)
s2 = sprintf('Lasso    : %g', lasso_auc)

legend(0.5, 0.4, legend=c(s1,s2), fill=c('Dodger Blue','Indian Red1'))
```



train set과 다르게 test set에 대해 ROC를 비교해본 결과 모양이 거의 유사하며, auc값도 동일하다는 것을 확인할 수 있다. 각 데이터셋에서의 Logistic 모델의 성능 차이가 유의미하게 크지는 않아 과적합 되었다고 판단할 수는 없다. 하지만 중요한 변수만 선별한 Lasso가 일반화 성능이 더 좋다고 판단할 수 있다. 따라서 test set의 결과를 통해 적은 변수로도 좋은 모델을 가질 수 있음을 확인하였다.

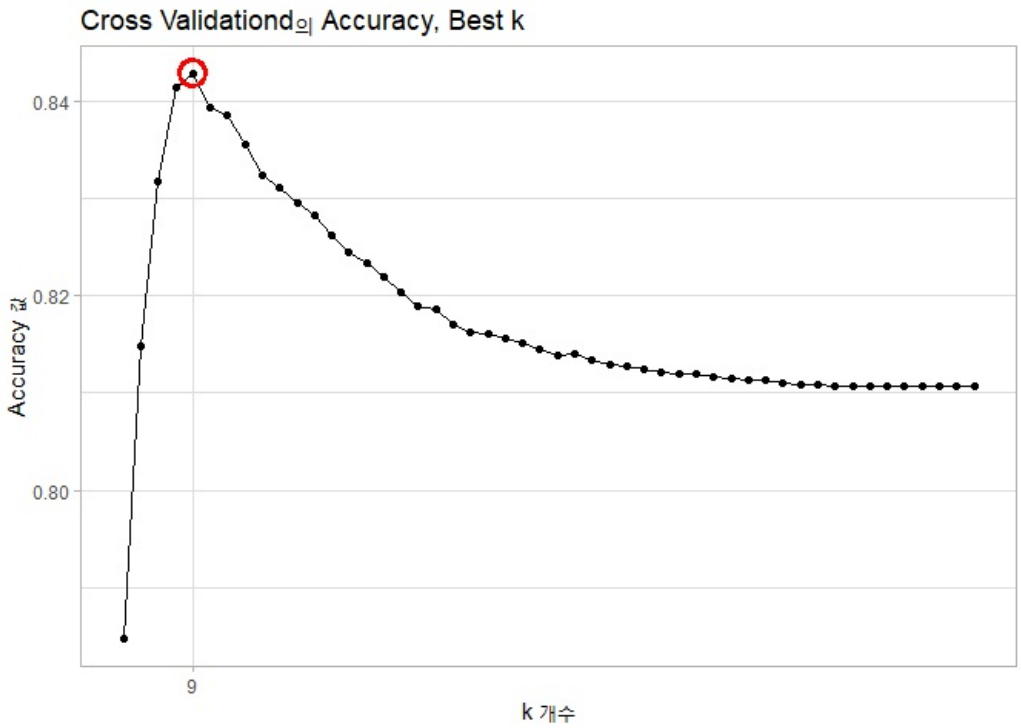
8번

Training set을 대상으로 k-nn을 적용해보자. 이때 cross validation으로 Accuracy가 가장 높은 best 값을 찾는다. best 값은 얼마인가?

[Accuracy]

```
z_normalized = c('center', 'scale')
cv = trainControl(method='repeatedcv', number=5, repeats=5)
tune_grid = expand.grid(k=seq(1, 99, 2))

# 정확도를 기준으로 cross validation
set.seed(1); ac_knn_fit = train(data=train, delay~., method='knn', trControl=cv,
                preProcess=z_normalized, tuneGrid=tune_grid, metric='Accuracy')
```

```
# 그래프에 최적 k를 표시하기 위해 데이터 추출
df = ac_knn_fit$results
max_row = df[df$Accuracy==max(df$Accuracy), ]
max_k = max_row$k

# 그래프 그리기
ggplot(ac_knn_fit) +
  geom_line() +
  geom_point() +
  theme_light() +
  labs(title="Cross Validationd의 Accuracy, Best k",
       x ="k 개수", y = "Accuracy 값") +
  geom_point(aes(x = max_k, y = max(df$Accuracy)), size = 5, color = "red", shape=1, stroke=2) +
  scale_x_continuous(breaks=max_k)
```

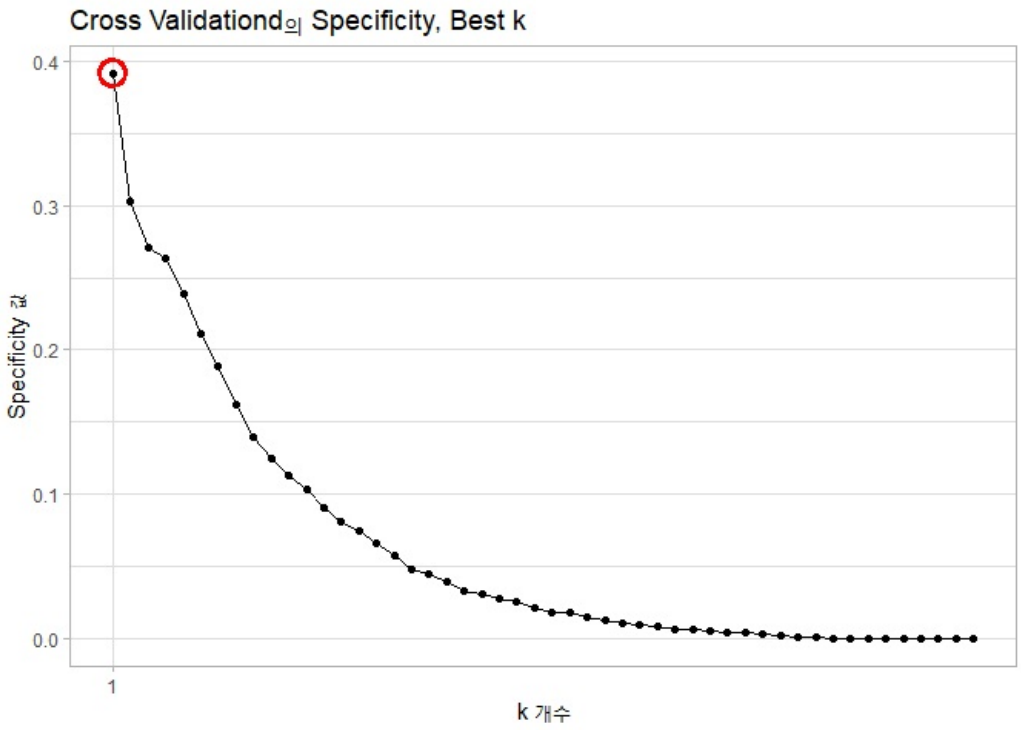


Train set을 대상으로 knn을 적용해본 결과 Accuracy값이 가장 높은 best k값은 9로 구할 수 있었다. 따라서 문제에서 원하는 best 모델은 k=9인 모델이지만, FP값을 줄이는 것이 목적일때, 다른 평가지표에서의 최적 K값을 찾아보도록 하겠다.

[Specificity]

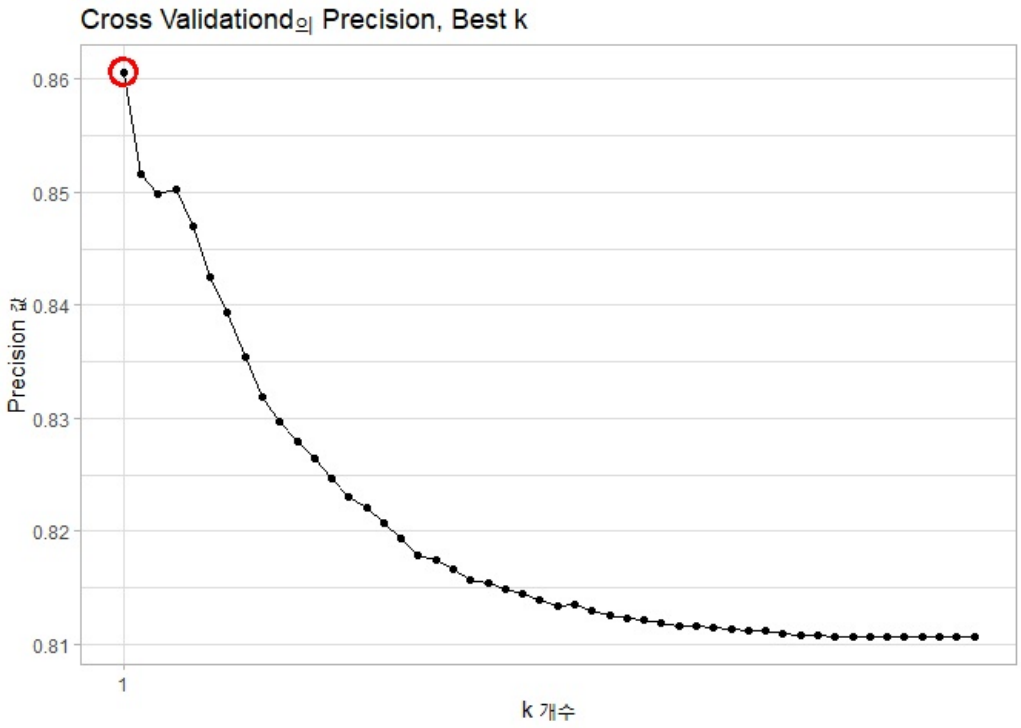
```
SpecFunction <- function(data, lev = NULL, model = NULL){
    cm <- confusionMatrix(data = data$pred, reference = data$obs, mode = "everything")
    out <- cm$byClass["Specificity"]
    names(out) <- 'Specificity'
    out
  }

cv <- trainControl(method = "repeatedcv", number = 5, repeats = 5,
                   summaryFunction = SpecFunction)


set.seed(1); knn_fit = train(data=train, delay~., method='knn', trControl=cv,
                preProcess=z_normalized, tuneGrid=tune_grid, metric='Specificity')

df = knn_fit$results
max_row = df[df$Specificity==max(df$Specificity), ]
max_k = max_row$k

ggplot(knn_fit) +
```

```
  geom_line() +
  geom_point() +
  theme_light() +
  labs(title="Cross Validationd의 Specificity, Best k",
       x ="k 개수", y = "Specificity 값") +
  geom_point(aes(x = max_k, y = max(df$Specificity)), size = 5, color = "red", shape=1, stroke=2) +
  scale_x_continuous(breaks=max_k)
```



[Precision]

```
#### F1 score
Prefunc <- function(data, lev = NULL, model = NULL) {
  cm <- confusionMatrix(data = data$pred, reference = data$obs, mode='everything')
  out <- cm$byClass["Precision"]
  names(out) <- 'Precision'
  out
}

cv <- trainControl(method = "repeatedcv", number = 5, repeats = 5,
                   summaryFunction = Prefunc)

set.seed(1); knn_fit <- train(delay ~ ., data = train, method = "knn", trControl = cv,
                 preProcess = z_normalized, tuneGrid = tune_grid, metric = "Precision")


df = knn_fit$results
max_row = df[df$Precision==max(df$Precision), ]
max_k = max_row$k


# 그래프 그리기
ggplot(knn_fit) +
  geom_line() +
  geom_point() +
  theme_light() +
  labs(title="Cross Validationd의 Precision, Best k",
       x ="k 개수", y = "Precision 값") +
  geom_point(aes(x = max_k, y = max(df$Precision)), size = 5, color = "red", shape=1, stroke=2) +
  scale_x_continuous(breaks=max_k)
```

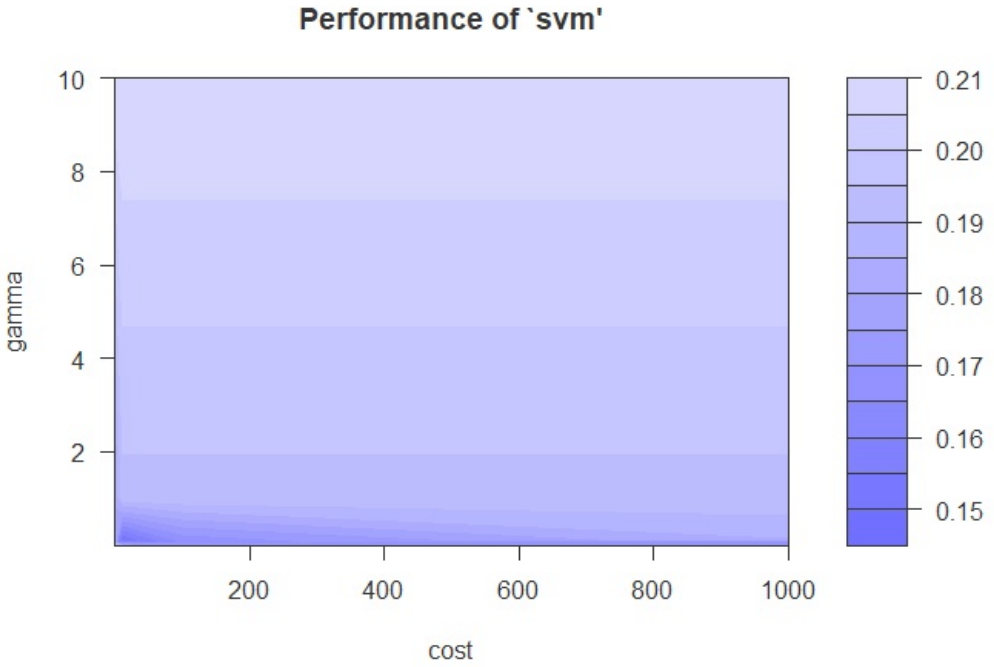


[Specificity & Precision 결과 비교]

위의 결과를 통해 k=1일때 Specificity와 Precision 모두 best model로 구해졌다. 이와 같은 결과가 나온 원인을 추측해보자면, knn과 같은 경우 주위 데이터를 기준으로 판단하기 때문에 데이터 수가 불균형할 경우 데이터분포가 적은 클래스는 주위 개수를 많이 고려할수록 데이터가 많은 클래스가 있을 확률이 높기 때문에 성능이 떨어지는 것으로 추측할 수 있다. 따라서 데이터 불균형이 심하고 적은 데이터를 잘 예측하기 위해서는 자신과 가장 가까운 데이터 1개 만을 고려했을때 성능이 가장 좋다고 판단할 수 있다.

9번

```
set.seed(1); tune.out = tune(svm, delay~., data=train, kernel='radial',
                ranges=list(cost=10^(-2:3), gamma=10^(-4:1)))
tune_sum = summary(tune.out)

# 그림으로 확인
plot(tune.out)
```



```
# 최적 파라미터
tune_sum$best.parameters
```

```
##    cost gamma
## 22   10   0.1
```

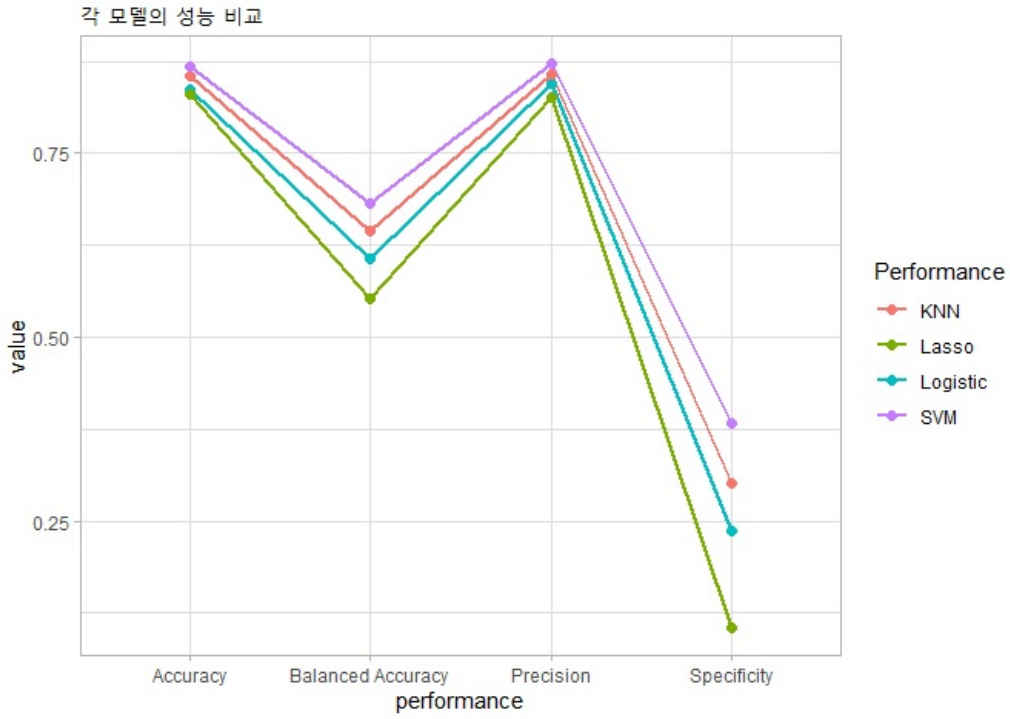
```
# 최고 성능
tune_sum$best.performance
```

```
## [1] 0.1495861
```

tune함수를 사용해서 svm을 실행한 결과 c=10, gamma=0.1일 때 최적의 파라미터를 갖으며, 최고 성능은 0.149로 구할 수 있었다.

10번

```
# Logistic
prob = predict(logi_model, newdata=test, type='response')
pred = rep('ontime', 648)
pred[prob>0.5] = 'delayed'
logi_cm = confusionMatrix(factor(pred), test$delay, positive='ontime',  mode = "everything")

# Lasso
pred = predict(cv_lasso, newx=testx, s=lambda_sim, type='class')
lasso_cm = confusionMatrix(factor(pred, levels=c('ontime', 'delayed')),
                           test$delay, positive='ontime',  mode = "everything")

# KNN
# 여러 모델 중 Accuracy가 가장 높은 모델 : ac_knn_fit
pred = predict(ac_knn_fit, test)
knn_cn = confusionMatrix(pred, test$delay, positive='ontime',  mode = "everything")


# SVM
svm_pred = predict(tune.out$best.model, newdata=test)
svm_cm = confusionMatrix(svm_pred, as.factor(test$delay),  mode = "everything")

# 모든 데이터 평가 지표 추출 및 결합
df_all = data.frame(
      performance = c(rep('Accuracy', 4), rep('Specificity', 4), rep('Balanced Accuracy', 4),
         rep('Precision', 4)),
      model = c(rep(c('Logistic', 'Lasso', 'KNN', 'SVM'), 4)),
      value = c(logi_cm$overall["Accuracy"][[1]], lasso_cm$overall["Accuracy"][[1]],
         knn_cn$overall["Accuracy"][[1]],svm_cm$overall["Accuracy"][[1]],
                logi_cm$byClass["Specificity"][[1]], lasso_cm$byClass["Specificity"][[1]],
         knn_cn$byClass["Specificity"][[1]], svm_cm$byClass["Specificity"][[1]],
                logi_cm$byClass["Balanced Accuracy"][[1]], lasso_cm$byClass["Balanced Accuracy"][[1]],
                knn_cn$byClass["Balanced Accuracy"][[1]], svm_cm$byClass["Balanced Accuracy"][[1]],
                logi_cm$byClass["Precision"][[1]], lasso_cm$byClass["Precision"][[1]],
         knn_cn$byClass["Precision"][[1]], svm_cm$byClass["Precision"][[1]])
)
ggplot(df_all, aes(x=performance, y=value, color=model, group=model)) +
  geom_line(linewidth=1) +
  geom_point(size=2) +
  theme_light() +
  labs(color="Performance", title='각 모델의 성능 비교')
```



(문제에서는 Lasso를 포함하지 않았지만, 추가로 넣어 비교하였다.) 그래프를 통해 살펴본 결과, SVM 모델의 성능이 모든 평가지표에서 우수한 것을 확인할 수 있었다. 다음으로 KNN이 높았으며, Lasso 모델의 성능이 가장 좋지 않았다.

또한 평가지표 별로 모델간의 성능 차이가 많이 나는 곳과 적게 나는 곳이 존재한다. 특히 Specificity를 보면 Accuracy에서는 별 차이가 없던 Lasso의 성능이 해당 지표에서는 아주 낮은 것을 확인할 수 있다. 따라서 공항이 단순히 accuracy를 판단하는 것이 목적이라면 변수를 적게 사용하는 간단한 모델인 Lasso을 사용하는 것이 이득일 수 있지만, 모델이 '제시간에 올 것이라고 예측했는데 실제로 연착한 경우'를 잘 판단하는 것이 목적이라면 SVM이나 KNN을 쓰는 것이 옳은 선택일 것이다.

7.2번에서 생각했던 것과 다르게 test set에 대해 성능을 비교해보면 Lasso 모델이 Logistic에 비해 성능이 좋지 않음을 바로 판단할 수 있다. 이는 7.2번에서 test set에 대해 두 모델의 auc값을 비교했을 때와 다른 결과이다. 따라서 모델의 성능을 평가하는데 있어서 AUC만으로 완전히 판단해서는 안된다는 것을 알 수 있다.