STAT 847: Final Project

Making Data-Driven Decisions When Planning Your Next Ski Trip

Jacky Chen, j57chen@uwaterloo.ca (mailto:j57chen@uwaterloo.ca)

Deadline April 23rd. Your deliverable will be an RMD file and a PDF emailed to me at mj2davis@uwaterloo.ca (mailto:mj2davis@uwaterloo.ca) Total word count target is 1000 words.

Find a dataset we didn't cover in class (I recommend the SSC case study competition, or the latest on Kaggle datasets under EDA)

It has to be large enough with enough features to fill at least 6 of 8 tasks. (Notice that 4 are mandatory, 4 are optional)

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0     ✔ purrr   1.0.1
## ✔ tibble  3.1.8     ✔ dplyr   1.1.0
## ✔ tidyr   1.3.0     ✔ stringr 1.5.0
## ✔ readr   2.1.3     ✔ forcats 1.0.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(skimr)
library(lubridate)
```

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

```
library(sp)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(rworldmap)
```

```
## ### Welcome to rworldmap ###
## For a short introduction type :   vignette('rworldmap')
```

```
library(rpart)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(ggplot2)
library(rpart.plot)
library(leaflet)
library(sf)
```

```
## Linking to GEOS 3.10.2, GDAL 3.4.2, PROJ 8.2.1; sf_use_s2() is TRUE
```

1) Describe and justify two different topics or approaches you might want to consider for this dataset and task. You don't have to use these tasks in the actual analysis. (Mandatory)

The data I choose for the final project is world-wide ski resort data (ticket price, location, number of lifts, etc) and amount of snow fall data. The dataset is publicly avaiable on Kaggle (https://www.kaggle.com/datasets/ulrikthygepedersen/ski-resorts?select=resorts.csv)

Two tasks I can think of are:

1.1) Find the optimal ski trip destination at a certain time of a year

Skiing is a popular winter activity enjoyed by millions of people around the world. However, planning a ski trip can be expensive, especially if someone does not have enough amount of information about ski resorts. With this dataset, we can provide data-driven insights to help skiers/snowboarders plan their trips more efficiently and cost-effectively.

This task would require building an evaluation function to rank ski resorts based on key factors of a ski trip. The factors may include accumulative amount of snowfall since the start of the season, forecast of fresh snow, area of accessible terrain (depending on skier/snowboarder skill level), resort ticket price, travel expense, ski school price (optional), etc. This would be a very useful tool for ski/snowboard enthusiasts.

1.2) Ski resort price analysis and prediction

Ski resort pricing can be highly variable and complex, with factors such as location, area and maintenance costs. By analyzing pricing data, we can identify patterns and trends that help us understand the drivers of these price fluctuations and make better decisions about when and where to book our ski trips. We can analyze the key factors that affect ticket price by building a regression model about price.

2) Describe and show the code used to clean and collect the data. (Optional)

2.1) Read CSV

We firstly read CSV files into dataframes

```
resorts_raw <- read.csv("resorts.csv")
snow_raw <- read.csv("snow.csv")
```

2.2) Clean snow dataset

Take a look at snow dataset. As shown below, the amount of snow in this Kaggle dataset seems to be normalized to be in the range of [0, 100].

```
snow_df = snow_raw
snow_df$Month = as_date(snow_df$Month)
summary(snow_df)
```

```
##      Month               Latitude        Longitude            Snow
##  Min.   :2022-01-01   Min.   :-55.38   Min.   :-179.88   Min.   :  0.39
##  1st Qu.:2022-02-01   1st Qu.: 50.88   1st Qu.: -78.62   1st Qu.: 59.45
##  Median :2022-04-01   Median : 60.62   Median :  36.88   Median :100.00
##  Mean   :2022-05-23   Mean   : 57.62   Mean   :  13.29   Mean   : 78.02
##  3rd Qu.:2022-10-01   3rd Qu.: 67.38   3rd Qu.:  98.38   3rd Qu.:100.00
##  Max.   :2022-12-01   Max.   : 83.38   Max.   : 179.88   Max.   :100.00
```

Map coordinates in snow dataset to continent & country. This will be needed in Q5.2.

```
# Ref: https://stackoverflow.com/questions/21708488/get-country-and-continent-from-longitude-and-latitude-point-in-r

# The single argument to this function, points, is a data.frame in which:
#   - column 1 contains the longitude in degrees
#   - column 2 contains the latitude in degrees
coords2continent = function(points)
{
  # countriesSP <- getMap(resolution='low')
  countriesSP <- getMap(resolution='high') #you could use high res map from rworldxtra if you were concerned about detail

  # converting points to a SpatialPoints object
  # setting CRS directly to that from rworldmap
  pointsSP = SpatialPoints(points, proj4string=CRS(proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = over(pointsSP, countriesSP)

  #indices$continent   # returns the continent (6 continent model)
  # indices$REGION   # returns the continent (7 continent model)
  return (list("continent" = indices$REGION, "country" = indices$ADMIN))
}

temp = coords2continent(snow_raw[, c("Longitude", "Latitude")])

snow_df[, "continent"] = temp$continent
snow_df[, "country"] = temp$country

summary(snow_df)
```

```
##      Month               Latitude        Longitude            Snow
##  Min.   :2022-01-01   Min.   :-55.38   Min.   :-179.88   Min.   :  0.39
##  1st Qu.:2022-02-01   1st Qu.: 50.88   1st Qu.: -78.62   1st Qu.: 59.45
##  Median :2022-04-01   Median : 60.62   Median :  36.88   Median :100.00
##  Mean   :2022-05-23   Mean   : 57.62   Mean   :  13.29   Mean   : 78.02
##  3rd Qu.:2022-10-01   3rd Qu.: 67.38   3rd Qu.:  98.38   3rd Qu.:100.00
##  Max.   :2022-12-01   Max.   : 83.38   Max.   : 179.88   Max.   :100.00
##
##          continent                           country
##  Europe       :440552   Russia                     :303691
##  North America:246543   Canada                     :177592
##  Asia         :113261   Greenland                  : 88454
##  South America: 11473   United States of America: 68951
##  Australia    :  1337   China                      : 54638
##  (Other)      :   787   (Other)                    :120627
##  NA's         :  6569   NA's                       :  6569
```

Some snow report cannot be matched to a continent or country. We will leave it for now.

2.3) Clean resort dataset

Now we take a look at resort dataset

```
summary(resorts_raw)
```

```
##        ID           Resort             Latitude        Longitude      
##  Min.   :  1.0   Length:499         Min.   :-45.05   Min.   :-149.741  
##  1st Qu.:125.5   Class :character   1st Qu.: 43.67   1st Qu.:   1.381  
##  Median :250.0   Mode  :character   Median : 46.35   Median :   8.206  
##  Mean   :250.0                      Mean   : 43.21   Mean   :  -6.007  
##  3rd Qu.:374.5                      3rd Qu.: 47.33   3rd Qu.:  12.429  
##  Max.   :499.0                      Max.   : 67.78   Max.   : 176.877  
##    Country           Continent             Price           Season         
##  Length:499         Length:499         Min.   :  0.00   Length:499        
##  Class :character   Class :character   1st Qu.: 36.00   Class :character  
##  Mode  :character   Mode  :character   Median : 45.00   Mode  :character  
##                                        Mean   : 48.72                     
##                                        3rd Qu.: 54.00                     
##                                        Max.   :141.00                     
##  Highest.point   Lowest.point  Beginner.slopes  Intermediate.slopes
##  Min.   : 163   Min.   :  36   Min.   :  0.00   Min.   :  0.00     
##  1st Qu.:1594   1st Qu.: 800   1st Qu.: 10.00   1st Qu.: 12.00     
##  Median :2175   Median :1121   Median : 18.00   Median : 25.00     
##  Mean   :2161   Mean   :1201   Mean   : 31.82   Mean   : 37.92     
##  3rd Qu.:2700   3rd Qu.:1500   3rd Qu.: 30.00   3rd Qu.: 45.00     
##  Max.   :3914   Max.   :3286   Max.   :312.00   Max.   :239.00     
##  Difficult.slopes  Total.slopes     Longest.run      Snow.cannons    
##  Min.   :  0.00   Min.   :  1.00   Min.   : 0.000   Min.   :   0.0  
##  1st Qu.:  3.00   1st Qu.: 30.00   1st Qu.: 0.000   1st Qu.:   0.0  
##  Median :  9.00   Median : 55.00   Median : 3.000   Median :  15.0  
##  Mean   : 16.16   Mean   : 85.91   Mean   : 3.545   Mean   : 179.1  
##  3rd Qu.: 21.00   3rd Qu.:100.00   3rd Qu.: 6.000   3rd Qu.: 180.0  
##  Max.   :126.00   Max.   :600.00   Max.   :16.000   Max.   :2383.0  
##  Surface.lifts    Chair.lifts     Gondola.lifts     Total.lifts     
##  Min.   : 0.00   Min.   : 0.000   Min.   : 0.000   Min.   :  0.00  
##  1st Qu.: 3.00   1st Qu.: 3.000   1st Qu.: 0.000   1st Qu.: 10.00  
##  Median : 7.00   Median : 6.000   Median : 1.000   Median : 15.00  
##  Mean   :11.28   Mean   : 9.721   Mean   : 3.259   Mean   : 24.26  
##  3rd Qu.:14.00   3rd Qu.:11.500   3rd Qu.: 4.000   3rd Qu.: 26.00  
##  Max.   :89.00   Max.   :74.000   Max.   :40.000   Max.   :174.00  
##  Lift.capacity    Child.friendly      Snowparks         Nightskiing       
##  Min.   :     0   Length:499         Length:499         Length:499        
##  1st Qu.: 11645   Class :character   Class :character   Class :character  
##  Median : 18510   Mode  :character   Mode  :character   Mode  :character  
##  Mean   : 31651                                                          
##  3rd Qu.: 32829                                                          
##  Max.   :252280                                                          
##  Summer.skiing     
##  Length:499        
##  Class :character  
##  Mode  :character  
##                    
##                    
##
```

Based on the summary above, we notice some resorts have $0 price. I will inspect these outliers below.

```
resorts_raw |> filter(Price <= 0) |> arrange(Continent) |> select(Resort, Country, Conti
nent, Price)
```

Resort <chr>	Country <chr>	Continent <chr>	Price <int>
Uludag?-Bursa	Turkey	Asia	0
Palando?ken-Ejder 3200 World Ski Center-	Turkey	Asia	0
High1 Resort	South Korea	Asia	0
Sun Mountain-Yabuli	China	Asia	0
Pragelato	Italy	Europe	0
Alpika Service	Russia	Europe	0
Puigmal	France	Europe	0
Yellowstone Club	United States	North America	0
Perisher	Australia	Oceania	0

9 rows

For this study, I am especially interested in Canadian ski resorts. I could manually fix the outliers but I would not spend much effort to do so if they are not about Canadian ski resorts. None of the outliers is Canadian resort so I will drop them.

```
resorts_df = resorts_raw
resorts_df = resorts_df[resorts_df$Price > 0, ]
```

Season column in *resorts.csv* is descriptive, such as “April”, “December - April”, “November - May, June - August”, “Year-round”, and “Unknown”. This is not useful at all for our analysis. We are going to parse the number of months open from season strings in the this section.

We firstly define a function to parse season strings. I will try to make it as general as possible so that it can be reused when data is updated.

```
parse_season = function(season_str){
  num_months_open = 0
  binary_encoder = rep(0:0, each=12) # 0/1 indicating if resort is open in each month
  season_str = str_replace_all(season_str, " ", "")
  if(tolower(season_str) == "year-round"){
    binary_encoder = rep(1:1, each=12)
  }
  else if (tolower(season_str) == "unknown"){
    binary_encoder = rep(0:0, each=12)
  }
  else if (!str_detect(season_str, "-")){
    # open single month
    month = match(season_str, month.name) # convert month name to number
    binary_encoder = replace(binary_encoder, month, 1)
  }
  else{
    parts = strsplit(season_str, ",")
    for (part in parts[[1]]){
      tokens = str_split_fixed(part, "-", 2)

      # convert month name to number
      start = match(tokens[1,1], month.name)
      end = match(tokens[1,2], month.name)
      # print(cat(start,"-",end))

      if (end < start) {
        end = end + 12
      }
      months = seq(start, end, by=1) %% 12 |> unique()
      months = replace(months, which(months == 0), 12) # change month 0 to month 12
      binary_encoder = replace(binary_encoder, months, 1)
    }
  }
  num_months_open = sum(binary_encoder)
  return(list("season_length" = num_months_open, "months_open"=binary_encoder))
}
```

Now we apply parse_season function to the Season column of the resorts dataframe.

```
month_names = month(seq(1,12, by=1), label=TRUE, abbr=FALSE) |> as.character()
for (i in 1:nrow(resorts_df)){
  results = parse_season(resorts_df[i,c("Season")])
  resorts_df[i,c("season_length")] = results$season_length
  resorts_df[i,month_names] = results$months_open # binary encoder for 12 months
}
```

We need to check if our parsing function works as expected. The result below looks fine.

```
resorts_df |> select(Season, season_length) |> unique() |> arrange(season_length)
```

Season <chr>	season_length <dbl>
Unknown	0
July	1
April	1
May	1
March	1
December	1
July - September	3
June - September	4
December - March	4
July - October	4

1-10 of 31 rows Previous **1** 2 3 4 Next

The resort dataframe has the latitudes of the highest and lowest points. We will use them to calculate resort vertical elevation (i.e. highest point - lowest point).

```
resorts_df$height = resorts_df$Highest.point - resorts_df$Lowest.point
```

2.4) Join snow and resort datasets

I want to know average amount of snow fall for each resort. To do this, we will need to join resort dataframe and snow dataframe together using longitude and latitude. However, the precision of data in two dataframes are different, making it none join results. To solve this problem, I will round the longitude and latitude.

Round longitude and latitude and merge dataframes.

```
# round lat and long in resort dataset
resorts_df$Lat_rounded = round(resorts_df$Latitude, digits=1)
resorts_df$Long_rounded = round(resorts_df$Longitude, digits=1)

# round lat and long in snow dataset
snow_df$Lat_rounded = round(snow_raw$Latitude, digits=1)
snow_df$Long_rounded = round(snow_raw$Longitude, digits=1)

mean_snow_df = snow_df |> group_by(Lat_rounded, Long_rounded) |> summarise(mean_snow=mea
n(Snow))
```

```
## `summarise()` has grouped output by 'Lat_rounded'. You can override using the
## `.groups` argument.
```

```
resorts_df |>
  left_join (
    mean_snow_df,
    by=c('Lat_rounded'='Lat_rounded', 'Long_rounded'='Long_rounded')
  ) |>
  filter(is.na(mean_snow)) |>
  count()
```

n
<int>
408

1 row

408 resorts cannot be matched with snow fall data if we round to 1 decimal digit. We will try rounding longitude and latitude to integers.

```
# round lat and long in resort dataset
resorts_df$Lat_rounded = round(resorts_df$Latitude, digits=0)
resorts_df$Long_rounded = round(resorts_df$Longitude, digits=0)

# round lat and long in snow dataset
snow_df$Lat_rounded = round(snow_raw$Latitude, digits=0)
snow_df$Long_rounded = round(snow_raw$Longitude, digits=0)

mean_snow_df = snow_df |> group_by(Lat_rounded, Long_rounded) |> summarise(mean_snow=mea
n(Snow), sd_snow=sd(Snow))
```

```
## `summarise()` has grouped output by 'Lat_rounded'. You can override using the
## `.groups` argument.
```

```
resorts_df |>
  left_join (
    mean_snow_df,
    by=c('Lat_rounded'='Lat_rounded', 'Long_rounded'='Long_rounded')
  ) |>
  filter(is.na(mean_snow)) |>
  count()
```

n
<int>
2

1 row

Now there are only 2 resorts do not have snow fall data.

Certainly, we would like our data as accurate as possible. We will accept rounding to integers because off by 1 latitude/longitude corresponds to off by 111 km, which is acceptable for snow fall estimation.

Now I will create a monthly snow report for every ski resort. I use snow dataset to left join the resort dataset and filter the rows that have a match. Then I group the dataframe by resort and month to calculate monthly snowfall for each resort.

```
snow_resort_df = snow_df |>
  left_join (
    resorts_df,
    by=c('Lat_rounded'='Lat_rounded', 'Long_rounded'='Long_rounded')
  ) |>
  filter(!is.na(Resort)) |>
  group_by(Month, Resort) |>
  summarise(monthly_snow=mean(Snow)) |>
  left_join (
    resorts_df,
    by=c('Resort'='Resort')
  )
```

```
## `summarise()` has grouped output by 'Month'. You can override using the
## `.groups` argument.
```

Add a column to indicate if the ski resort is open in current month and 2 columns to indicate the end of the month and snow amount (for creating graphs).

```
snow_resort_df = cbind(snow_resort_df, is_open_curr_month=NA)
snow_resort_df = cbind(snow_resort_df, Month_end=NA)
snow_resort_df = cbind(snow_resort_df, month_end_snow=NA)

current_resort = ""

# order df by resort then by month
snow_resort_df = snow_resort_df[order(snow_resort_df[,"Resort"], snow_resort_df[,"Mont
h"]), ]

for (i in 1:nrow(snow_resort_df)){
  # find if resort is open in that month
  month_name = month(snow_resort_df[i,]$Month, label = TRUE, abbr = FALSE)
  is_open = "Yes"
  if (snow_resort_df[i, c(month_name)] == 0){
    is_open = "No"
  }
  snow_resort_df[i,]$is_open_curr_month = is_open

  # if resort is changed, reset variables
  if (snow_resort_df[i,"Resort"] != current_resort){
    # reset variables
    current_resort = snow_resort_df[i,"Resort"]
  }

  # if next row is not out of bound and it's about the same ski resort, use next month's
snow as the snow of current month end
  if (i+1 <= nrow(snow_resort_df) & snow_resort_df[i+1,"Resort"] == current_resort){
    snow_resort_df[i,]$Month_end = snow_resort_df[i+1,]$Month
    snow_resort_df[i,]$month_end_snow = snow_resort_df[i+1,]$monthly_snow
  }
  # if next row is not out of bound but it's about a new resort, we reached the end of c
urrent resort
  # use this month's snow as the snow of current month end
  else if (i+1 <= nrow(snow_resort_df) & snow_resort_df[i+1,"Resort"] != current_resort)
{
    snow_resort_df[i,]$Month_end = ceiling_date(ymd(snow_resort_df[i,]$Month), 'month')
- days(1) # end of month
    snow_resort_df[i,]$month_end_snow = snow_resort_df[i,]$monthly_snow
  }
  # we reached the last row of df
  else if (i+1 > nrow(snow_resort_df)){
    snow_resort_df[i,]$Month_end = ceiling_date(ymd(snow_resort_df[i,]$Month), 'month')
- days(1) # end of month
    snow_resort_df[i,]$month_end_snow = snow_resort_df[i,]$monthly_snow
  }
}
```

Make sure everything looks fine with joined dataframe

```
summary(snow_resort_df)
```

```
##      Month                Resort           monthly_snow          ID
##  Min.   :2022-01-01   Length:4738        Min.   :  0.39   Min.   :  1.0
##  1st Qu.:2022-03-01   Class :character   1st Qu.: 10.16   1st Qu.:116.2
##  Median :2022-06-01   Mode  :character   Median : 38.68   Median :237.5
##  Mean   :2022-06-10                      Mean   : 42.36   Mean   :241.4
##  3rd Qu.:2022-10-01                      3rd Qu.: 72.72   3rd Qu.:364.8
##  Max.   :2022-12-01                      Max.   :100.00   Max.   :499.0
##     Latitude        Longitude          Country           Continent
##  Min.   :-45.05   Min.   :-149.741   Length:4738        Length:4738
##  1st Qu.: 44.87   1st Qu.:   2.072   Class :character   Class :character
##  Median : 46.43   Median :   8.076   Mode  :character   Mode  :character
##  Mean   : 43.99   Mean   :  -7.735
##  3rd Qu.: 47.29   3rd Qu.:  11.910
##  Max.   : 67.78   Max.   : 176.877
##      Price            Season           Highest.point   Lowest.point
##  Min.   : 14.00   Length:4738        Min.   : 163   Min.   :  36
##  1st Qu.: 38.00   Class :character   1st Qu.:1678   1st Qu.: 812
##  Median : 46.00   Mode  :character   Median :2225   Median :1137
##  Mean   : 49.55                      Mean   :2216   Mean   :1205
##  3rd Qu.: 54.00                      3rd Qu.:2749   3rd Qu.:1500
##  Max.   :141.00                      Max.   :3914   Max.   :3286
##  Beginner.slopes Intermediate.slopes Difficult.slopes  Total.slopes
##  Min.   :  0.0   Min.   :  0.00      Min.   :  0.00   Min.   :  1.00
##  1st Qu.: 10.0   1st Qu.: 12.00      1st Qu.:  3.00   1st Qu.: 33.00
##  Median : 20.0   Median : 26.00      Median :  9.00   Median : 57.00
##  Mean   : 34.5   Mean   : 41.08      Mean   : 16.99   Mean   : 92.57
##  3rd Qu.: 35.0   3rd Qu.: 46.00      3rd Qu.: 22.00   3rd Qu.:103.00
##  Max.   :312.0   Max.   :239.00      Max.   :126.00   Max.   :600.00
##   Longest.run      Snow.cannons    Surface.lifts    Chair.lifts
##  Min.   : 0.000   Min.   :   0.0   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 0.000   1st Qu.:   0.0   1st Qu.: 3.00   1st Qu.: 3.00
##  Median : 3.000   Median :  28.0   Median : 7.00   Median : 6.00
##  Mean   : 3.892   Mean   : 194.6   Mean   :11.88   Mean   :10.19
##  3rd Qu.: 7.000   3rd Qu.: 200.0   3rd Qu.:14.00   3rd Qu.:12.00
##  Max.   :16.000   Max.   :2383.0   Max.   :89.00   Max.   :74.00
##  Gondola.lifts     Total.lifts     Lift.capacity    Child.friendly
##  Min.   : 0.000   Min.   :  0.00   Min.   :     0   Length:4738
##  1st Qu.: 0.000   1st Qu.: 10.00   1st Qu.: 11620   Class :character
##  Median : 1.000   Median : 15.00   Median : 18808   Mode  :character
##  Mean   : 3.706   Mean   : 25.78   Mean   : 33924
##  3rd Qu.: 4.000   3rd Qu.: 28.00   3rd Qu.: 35624
##  Max.   :40.000   Max.   :174.00   Max.   :252280
##   Snowparks         Nightskiing        Summer.skiing      season_length
##  Length:4738        Length:4738        Length:4738        Min.   : 0.000
##  Class :character   Class :character   Class :character   1st Qu.: 5.000
##  Mode  :character   Mode  :character   Mode  :character   Median : 5.000
##                                                           Mean   : 5.278
##                                                           3rd Qu.: 6.000
##                                                           Max.   :12.000
##     January          February          March            April
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:1.0000   1st Qu.:1.0000   1st Qu.:1.0000   1st Qu.:1.0000
```

```
##  Median :1.0000   Median :1.0000   Median :1.0000   Median :1.0000
##  Mean   :0.9059   Mean   :0.9059   Mean   :0.9092   Mean   :0.8364
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       May              June              July             August
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.0000   Median :0.00000   Median :0.00000   Median :0.00000
##  Mean   :0.1463   Mean   :0.05931   Mean   :0.05614   Mean   :0.05361
##  3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##    September          October           November         December
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:1.0000
##  Median :0.00000   Median :0.00000   Median :0.0000   Median :1.0000
##  Mean   :0.05762   Mean   :0.08168   Mean   :0.3561   Mean   :0.9101
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##      height      Lat_rounded      Long_rounded      is_open_curr_month
##  Min.   :   4   Min.   :-45.00   Min.   :-150.000   Length:4738
##  1st Qu.: 665   1st Qu.: 45.00   1st Qu.:   2.000   Class :character
##  Median : 957   Median : 46.00   Median :   8.000   Mode  :character
##  Mean   :1011   Mean   : 43.97   Mean   :  -7.711
##  3rd Qu.:1300   3rd Qu.: 47.00   3rd Qu.:  12.000
##  Max.   :2509   Max.   : 68.00   Max.   : 177.000
##    Month_end          month_end_snow
##  Min.   :2022-02-01   Min.   :  0.39
##  1st Qu.:2022-04-01   1st Qu.:  9.84
##  Median :2022-08-01   Median : 37.20
##  Mean   :2022-07-18   Mean   : 42.17
##  3rd Qu.:2022-11-01   3rd Qu.: 72.02
##  Max.   :2022-12-31   Max.   :100.00
```

```
annual_snow_resort_df = snow_resort_df |>
  filter(is_open_curr_month == "Yes") |>
  group_by(Resort) |>
  summarise(annual_mean_snow=mean(monthly_snow), annual_total_snow=sum(monthly_snow)) |>
  left_join (
    resorts_df,
    by=c('Resort'='Resort')
  )
```

```
summary(annual_snow_resort_df)
```

```
##     Resort          annual_mean_snow annual_total_snow       ID
##  Length:463         Min.   : 0.39    Min.   :  0.39    Min.   :  1.0
##  Class :character   1st Qu.:44.61    1st Qu.:229.79    1st Qu.:117.5
##  Mode  :character   Median :63.55    Median :348.66    Median :239.0
##                     Mean   :60.81    Mean   :327.64    Mean   :242.8
##                     3rd Qu.:81.43    3rd Qu.:428.82    3rd Qu.:365.5
##                     Max.   :98.70    Max.   :724.65    Max.   :499.0
##     Latitude        Longitude           Country           Continent
##  Min.   :-45.05   Min.   :-149.7407   Length:463         Length:463
##  1st Qu.: 44.24   1st Qu.:   0.5582   Class :character   Class :character
##  Median : 46.40   Median :   8.0651   Mode  :character   Mode  :character
##  Mean   : 43.46   Mean   :  -8.6273
##  3rd Qu.: 47.33   3rd Qu.:  12.1637
##  Max.   : 67.78   Max.   : 176.8767
##      Price           Season          Highest.point   Lowest.point
##  Min.   : 14.00   Length:463         Min.   : 163   Min.   :  36
##  1st Qu.: 37.00   Class :character   1st Qu.:1628   1st Qu.: 800
##  Median : 46.00   Mode  :character   Median :2190   Median :1130
##  Mean   : 50.21                      Mean   :2187   Mean   :1209
##  3rd Qu.: 55.50                      3rd Qu.:2728   3rd Qu.:1500
##  Max.   :141.00                      Max.   :3914   Max.   :3286
##  Beginner.slopes  Intermediate.slopes Difficult.slopes  Total.slopes
##  Min.   :  0.00   Min.   :  0.00      Min.   :  0.00   Min.   :  1.0
##  1st Qu.: 10.00   1st Qu.: 12.00      1st Qu.:  3.00   1st Qu.: 32.0
##  Median : 19.00   Median : 26.00      Median :  9.00   Median : 57.0
##  Mean   : 32.95   Mean   : 39.53      Mean   : 16.82   Mean   : 89.3
##  3rd Qu.: 32.00   3rd Qu.: 45.00      3rd Qu.: 22.00   3rd Qu.:101.0
##  Max.   :312.00   Max.   :239.00      Max.   :126.00   Max.   :600.0
##   Longest.run      Snow.cannons    Surface.lifts    Chair.lifts
##  Min.   : 0.000   Min.   :   0.0   Min.   : 0.00   Min.   : 0
##  1st Qu.: 0.000   1st Qu.:   0.0   1st Qu.: 3.00   1st Qu.: 4
##  Median : 3.000   Median :  30.0   Median : 7.00   Median : 6
##  Mean   : 3.793   Mean   : 190.9   Mean   :11.57   Mean   :10
##  3rd Qu.: 6.000   3rd Qu.: 201.0   3rd Qu.:14.00   3rd Qu.:12
##  Max.   :16.000   Max.   :2383.0   Max.   :89.00   Max.   :74
##  Gondola.lifts     Total.lifts     Lift.capacity    Child.friendly
##  Min.   : 0.000   Min.   :  1.00   Min.   :   900   Length:463
##  1st Qu.: 0.000   1st Qu.: 10.00   1st Qu.: 11714   Class :character
##  Median : 1.000   Median : 15.00   Median : 19200   Mode  :character
##  Mean   : 3.406   Mean   : 24.98   Mean   : 32814
##  3rd Qu.: 4.000   3rd Qu.: 27.00   3rd Qu.: 34296
##  Max.   :40.000   Max.   :174.00   Max.   :252280
##   Snowparks         Nightskiing        Summer.skiing      season_length
##  Length:463         Length:463         Length:463         Min.   : 1.000
##  Class :character   Class :character   Class :character   1st Qu.: 5.000
##  Mode  :character   Mode  :character   Mode  :character   Median : 5.000
##                                                           Mean   : 5.477
##                                                           3rd Qu.: 6.000
##                                                           Max.   :12.000
##     January          February          March           April
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:1.0000   1st Qu.:1.0000   1st Qu.:1.000   1st Qu.:1.0000
```

```
##  Median :1.0000   Median :1.0000   Median :1.000   Median :1.0000
##  Mean   :0.9417   Mean   :0.9417   Mean   :0.946   Mean   :0.8639
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
##       May              June              July             August
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.0000   Median :0.00000   Median :0.00000   Median :0.00000
##  Mean   :0.1447   Mean   :0.06263   Mean   :0.05832   Mean   :0.05616
##  3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##    September          October           November         December
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:1.000
##  Median :0.00000   Median :0.00000   Median :0.0000   Median :1.000
##  Mean   :0.06048   Mean   :0.08207   Mean   :0.3737   Mean   :0.946
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   Max.   :1.000
##      height        Lat_rounded      Long_rounded
##  Min.   :  20.0   Min.   :-45.00   Min.   :-150.000
##  1st Qu.: 625.0   1st Qu.: 44.00   1st Qu.:   1.000
##  Median : 930.0   Median : 46.00   Median :   8.000
##  Mean   : 977.9   Mean   : 43.44   Mean   :  -8.609
##  3rd Qu.:1255.0   3rd Qu.: 47.00   3rd Qu.:  12.000
##  Max.   :2509.0   Max.   : 68.00   Max.   : 177.000
```

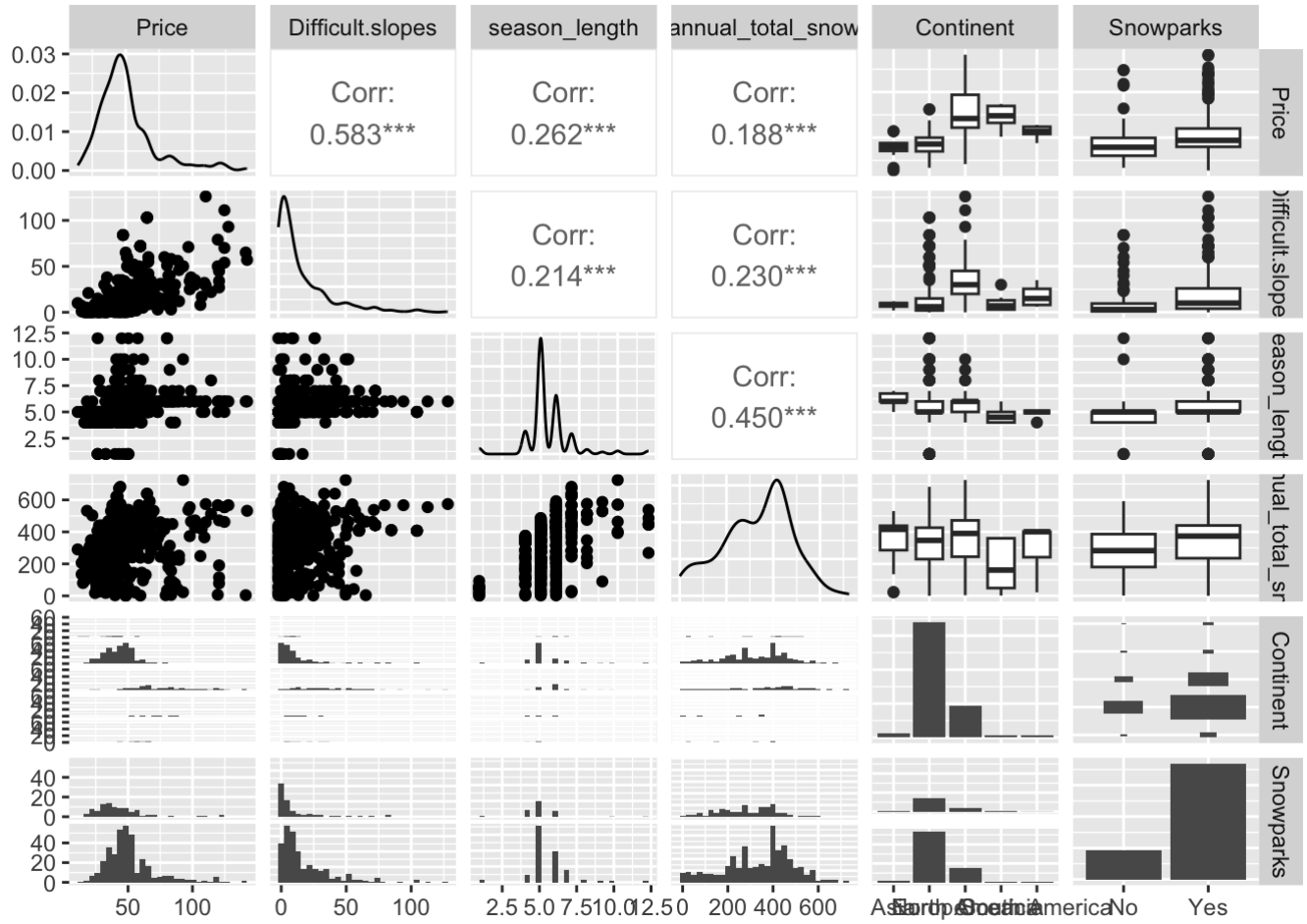
3) Give a ggpairs plot of what you think are the six most important variables. At least one must be categorical, and one continuous. Explain your choice of variables and the trends between them. (Mandatory)

I want to know what are the factors that correlates with price. My initial assumption is seasonality, snow condition and location are the main factors.

I also want to know what are the factors that determine if a ski resort wants to build terrain parks in Q4, so `Snowparks` is also included in this ggpairs plot.

```
ggpairs(annual_snow_resort_df,
             columns = c("Price", "Difficult.slopes", "season_length", "annual_total_sn
ow", "Continent", "Snowparks"),
             progress = FALSE)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



3.1) Price vs Continent

In our dataset, Europe has the most number of ski resorts, followed by North America and Asia.

North America has the widest spread of price, followed by Europe. Asia and South America price are the most centralized.

North American resorts are significantly more expensive than resorts in other continents. Almost 50% of North America resorts are more expensive than the top-end (expensive) ski resorts in Europe, Asia, and South America.

I select two most expensive ski resorts from each continent. Now I start to question if the price I payed was worth it 🧐 , because the price of, say Lake Louise, is more expensive than the most expensive resort in Asian and Sounth America, and it's almost the same price of the most luxurious resort in Europe.

```
df_list = split(annual_snow_resort_df, annual_snow_resort_df$Continent)
df_top = lapply(df_list, function(x) head(x[order(x$Price, decreasing = TRUE), ], 2))
df_final = do.call(rbind, df_top) |> select(Resort, Continent, Country, Price) |> arrang
e(Price)
df_final
```

Resort <chr>	Continent <chr>	Country <chr>	Price <int>
Niseko	Asia	Japan	57
YongPyong Resort	Asia	South Korea	57
El Colorado-?Farellones	South America	Chile	64
Valle Nevado	South America	Chile	64
Cervinia	Europe	Switzerland	81
Zermatt - Matterhorn	Europe	Switzerland	81
Falls Creek	Oceania	Australia	87
Mt. Buller	Oceania	Australia	87
Breckenridge	North America	United States	140
Beaver Creek	North America	United States	141

1-10 of 10 rows

3.2) Price vs other variables

There is a correlation between price and number of difficult slopes. For the majority of ski resorts, the number of difficult slopes is below 10.

Surprisingly, there is no correlation between price and season length or annual total snow.

Whether a ski resort has terrain park or not does not affect the price.

4) Build a classification tree of one of the six variables from the last part as a function of the other five, and any other explanatory variables you think are necessary. Show code, explain reasoning, and show the tree as a simple (ugly) plot. Show the confusion matrix. Give two example predictions and follow them down the tree. (Mandatory)

I am curious what are the factors that decide if a ski resort is willing to build a terrain park. My assumption is the difficulty of the ski resort, season length, and the amount of snow might be the main factors. I will pass those 3 attributes and other attributes to build a tree model and see what attributes will be used by the mode.

Feature engineering and train test split:

```
# convert string features to factors
model_data_df = annual_snow_resort_df
model_data_df$Continent = factor(model_data_df$Continent)
model_data_df$Snowparks = factor(model_data_df$Snowparks)

# make this example reproducible
set.seed(1)

# Train test split
sample = sample(c(TRUE, FALSE), nrow(model_data_df), replace=TRUE, prob=c(0.8,0.2))
train_df = model_data_df[sample, ]
test_df = model_data_df[!sample, ]
```

4.1) Build classification tree

Build a classification tree and evaluate its performance

```
# build a classification tree
tree_model = rpart(Snowparks ~ Price + Difficult.slopes + season_length + annual_total_s
now + Continent + season_length, data=train_df, method = "class")
# make prediction
pred = predict(tree_model, test_df, type="class")
# evaluate
confusionMatrix(pred, test_df$Snowparks, mode = "everything", positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No   2   5
##        Yes 15  65
##
##                Accuracy : 0.7701
##                  95% CI : (0.6675, 0.8536)
##     No Information Rate : 0.8046
##     P-Value [Acc > NIR] : 0.82896
##
##                   Kappa : 0.0595
##
##  Mcnemar's Test P-Value : 0.04417
##
##             Sensitivity : 0.9286
##             Specificity : 0.1176
##          Pos Pred Value : 0.8125
##          Neg Pred Value : 0.2857
##               Precision : 0.8125
##                  Recall : 0.9286
##                      F1 : 0.8667
##              Prevalence : 0.8046
##          Detection Rate : 0.7471
##    Detection Prevalence : 0.9195
##       Balanced Accuracy : 0.5231
##
##        'Positive' Class : Yes
##
```

Print a table of optimal prunings based on a complexity parameter (CP).

```
printcp(tree_model)
```

```
##
## Classification tree:
## rpart(formula = Snowparks ~ Price + Difficult.slopes + season_length +
##     annual_total_snow + Continent + season_length, data = train_df,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] annual_total_snow Continent         Difficult.slopes  Price
##
## Root node error: 77/376 = 0.20479
##
## n= 376
##
##         CP nsplit rel error xerror    xstd
## 1 0.058442      0   1.00000 1.0000 0.10162
## 2 0.032468      3   0.81818 1.0130 0.10211
## 3 0.017316      5   0.75325 1.0649 0.10399
## 4 0.010000     11   0.64935 1.0390 0.10306
```

In the original classification tree, accuracy is 0.7701 and F1 is 0.8667. 4 variables are used in the tree:

- annual_total_snow
- Continent
- Difficult.slopes
- Price

The CP was 0.01 at the end, and 0.017 when there was six branches left to grow. I will try pruning technique by setting cp > 0.017 to see if we can prevent overfitting.

```
# train pruned model
tree_model_pruned = prune(tree_model, cp=0.02)
# make prediction
pred = predict(tree_model_pruned, test_df, type="class")
# evaluate
confusionMatrix(pred, test_df$Snowparks, mode = "everything", positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No   1   1
##        Yes 16  69
##
##                Accuracy : 0.8046
##                  95% CI : (0.7057, 0.8819)
##     No Information Rate : 0.8046
##     P-Value [Acc > NIR] : 0.564403
##
##                   Kappa : 0.0669
##
##  Mcnemar's Test P-Value : 0.000685
##
##             Sensitivity : 0.98571
##             Specificity : 0.05882
##          Pos Pred Value : 0.81176
##          Neg Pred Value : 0.50000
##               Precision : 0.81176
##                  Recall : 0.98571
##                      F1 : 0.89032
##              Prevalence : 0.80460
##          Detection Rate : 0.79310
##    Detection Prevalence : 0.97701
##       Balanced Accuracy : 0.52227
##
##        'Positive' Class : Yes
##
```

```
printcp(tree_model_pruned)
```

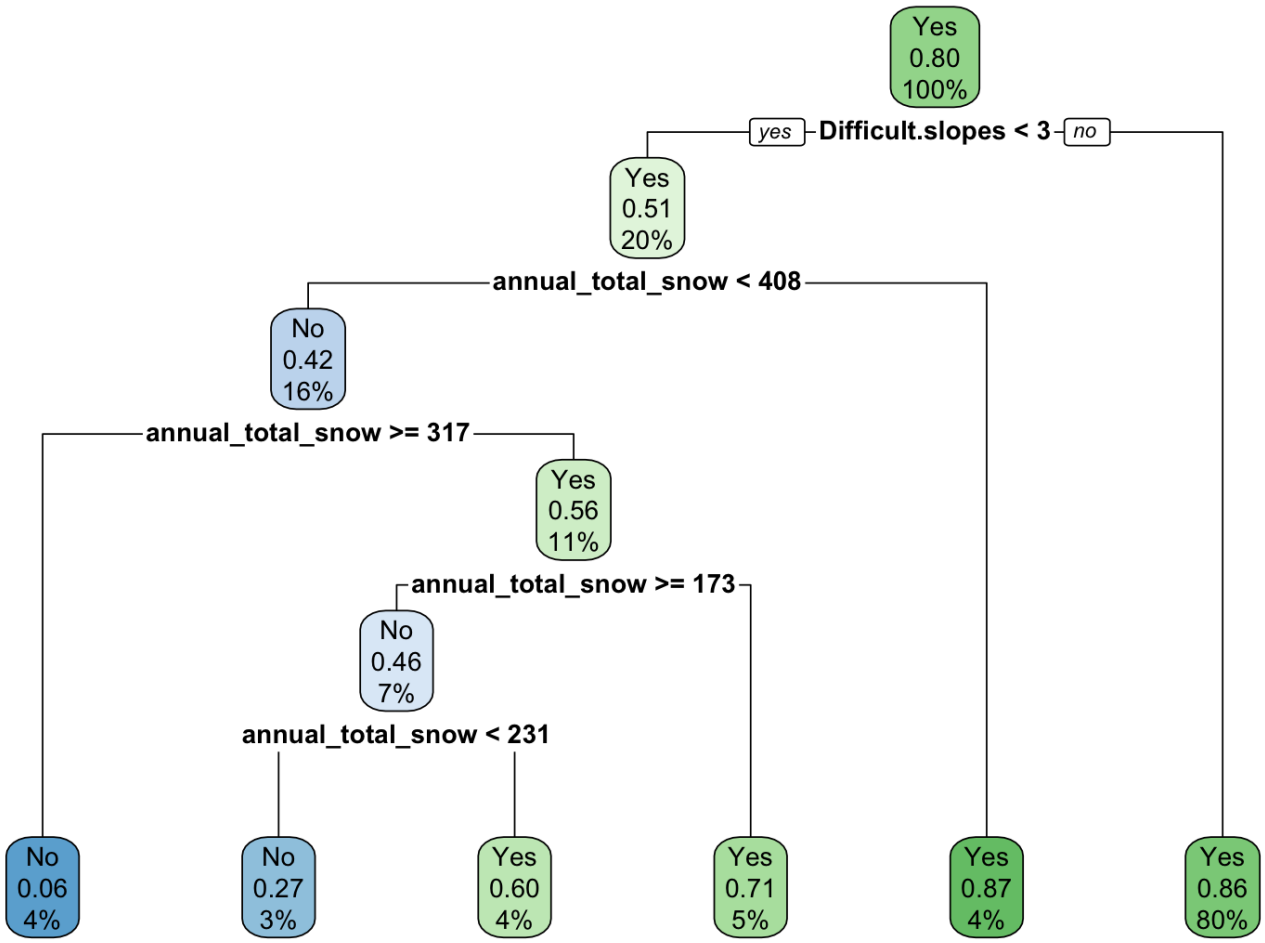
```
##
## Classification tree:
## rpart(formula = Snowparks ~ Price + Difficult.slopes + season_length +
##     annual_total_snow + Continent + season_length, data = train_df,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] annual_total_snow Difficult.slopes
##
## Root node error: 77/376 = 0.20479
##
## n= 376
##
##         CP nsplit rel error xerror    xstd
## 1 0.058442      0   1.00000 1.0000 0.10162
## 2 0.032468      3   0.81818 1.0130 0.10211
## 3 0.020000      5   0.75325 1.0649 0.10399
```

The pruned model improves F1 score from 0.8667 to 0.89032, and accuracy from 0.7701 to 0.8046. In addition, only 2 variables (annual_total_snow and Difficult.slopes) are used to construct the tree model, which simplifies model interpretability. Therefore, we will choose the pruned tree as our final model.

4.2) Visualize the classification tree

Visualize the pruned tree:

```
rpart.plot(tree_model_pruned, uniform=TRUE)
```



4.3) Follow example predictions down the tree

Below are five examples and I will follow their prediction down the tree.

```
test_samples = test_df |> filter(Country == "Canada" | Resort == "Furano") |> arrange(Di
fficult.slopes) |> select(Resort, Price, Difficult.slopes, season_length,annual_total_sn
ow, Continent, season_length, Snowparks)
test_samples
```

Resort	Price	Difficult.slopes	season_length	annual_total_snow
<chr>	<int>	<int>	<dbl>	<dbl>
Furano	42	2	7	442.6160

Resort <chr>	Price <int>	Difficult.slopes <int>	season_length <dbl>	annual_total_snow <dbl>
Panorama	62	25	5	427.0330
Fernie	67	42	5	473.3134
Lake Louise	66	42	7	638.8596
Red Mountain Resort-Rossland	60	50	5	377.6451

5 rows | 1-6 of 7 columns

```
predict(tree_model_pruned, test_samples, type="class")
```

```
##   1   2   3   4   5
## Yes Yes Yes Yes Yes
## Levels: No Yes
```

Furano ski resort in Japan only has 2 difficult slopes so we go down the left branch of the root node. Its annual total snow is 442, which is greater than 408 so we go down the right branch of the 2-level node. Now we reach the leaf node of Yes category.

Panorama, Fernie, Lake Louise, and Red Mountain ski resorts in Canada have at least 20 difficult slopes so we go down the right branch of the root node. Now we reach the leaf node of Yes category.

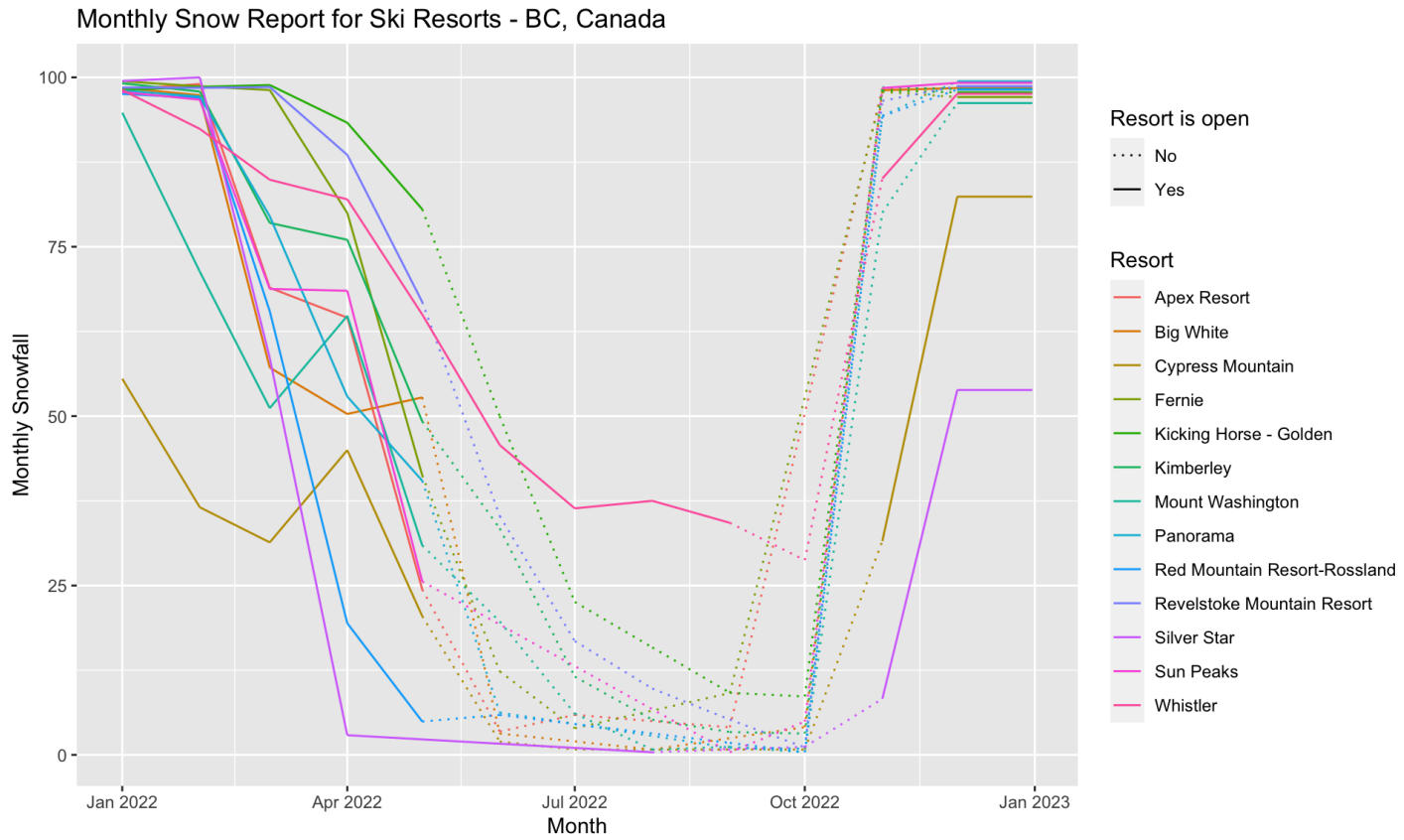
5) Build a visually impressive ggplot to show the relationship between at least three variables. (Optional)

5.1) Build a ggplot to show relationship between Canadian ski resorts, month and snow fall.

Since BC has most of the ski resorts in Canada, I will split Canadian dataset into BC and other provinces

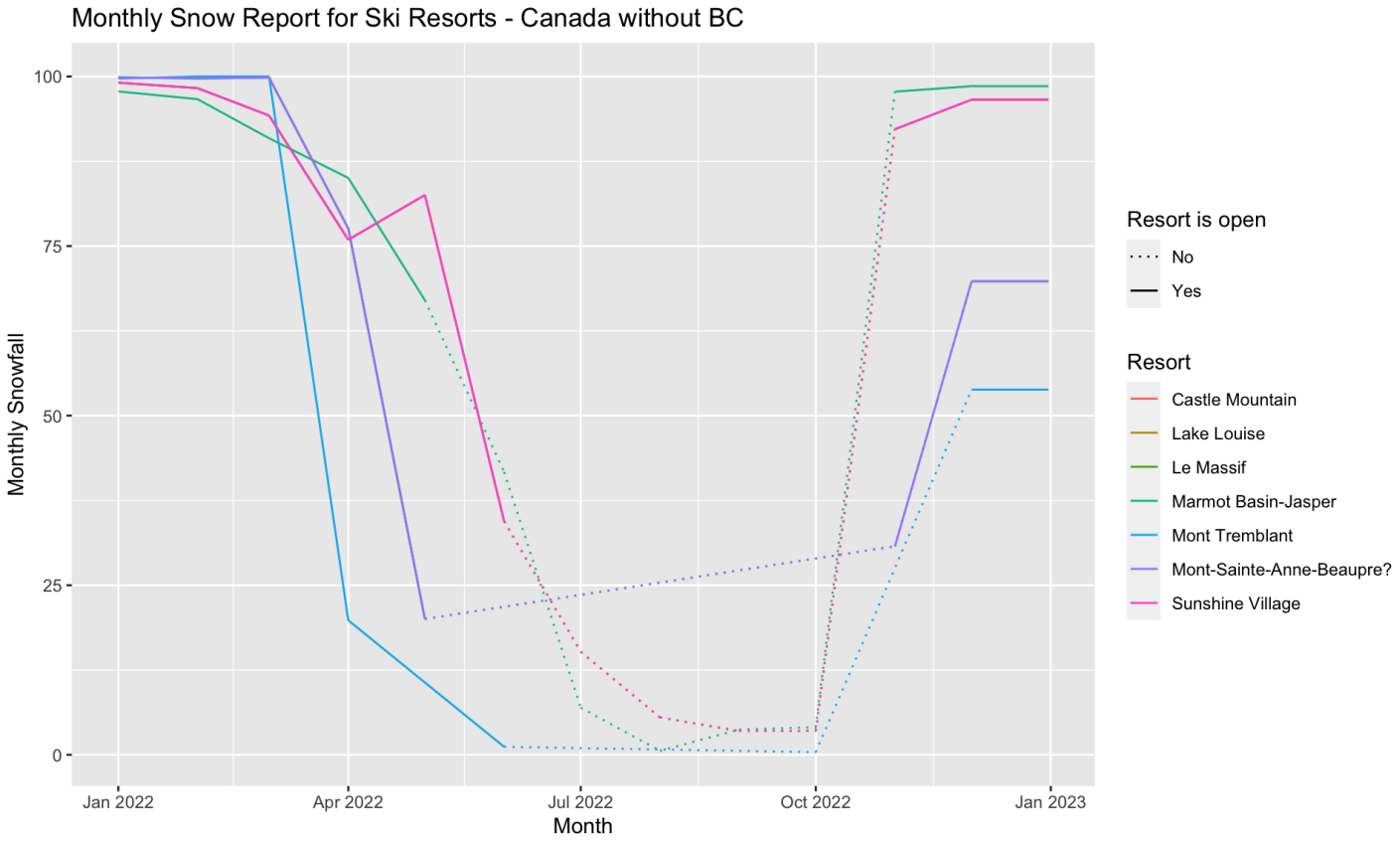
```
snow_canada_df = snow_resort_df |> filter(Country == "Canada")
snow_canada_BC_df = snow_canada_df |> filter(Resort %in% c("Whistler","Sun Peaks", "Silv
er Star", "Revelstoke Mountain Resort", "Red Mountain Resort-Rossland", "Panorama", "Mou
nt Washington", "Kimberley", "Kicking Horse - Golden", "Fernie", "Cypress Mountain", "Bi
g White", "Apex Resort"))
snow_canada_Other_df = snow_canada_df |> filter(Resort %in% c("Sunshine Village","Marmot
Basin-Jasper","Lake Louise","Castle Mountain", "Mont-Sainte-Anne-Beaupre?","Mont Trembla
nt","Le Massif"))
```

```
ggplot() +
  geom_segment(data = snow_canada_BC_df,
               aes(x = Month, y = monthly_snow, xend = Month_end, yend = month_end_snow,
colour=Resort, linetype = as.character(is_open_curr_month))) +
  scale_linetype_manual(name = "Resort is open", values=c("dotted", "solid")) +
  xlab("Month") +
  ylab("Monthly Snowfall") +
  ggtitle("Monthly Snow Report for Ski Resorts - BC, Canada")
```



As shown in the plot, Whistler has the longest season among all Canadian resorts. It is the only place to go if you want to ski in early spring and summer. Kicking horse in Golden, BC consistently has the most amount of snowfall when it is open. For most of the time, Cypress Mountain should be avoided for people who are looking for fresh snow. Even though Silver Star and Red Mountain also seem to be not having too much snow in early of the year, I think this is because March snow fall data is missing for those areas (they have descent amount of snow in Jan and Feb).

```
ggplot() +
  geom_segment(data = snow_canada_Other_df,
               aes(x = Month, y = monthly_snow, xend = Month_end, yend = month_end_snow,
colour=Resort, linetype = as.character(is_open_curr_month))) +
  scale_linetype_manual(name = "Resort is open", values=c("dotted", "solid")) +
  xlab("Month") +
  ylab("Monthly Snowfall") +
  ggtitle("Monthly Snow Report for Ski Resorts - Canada without BC")
```



Sunshine Village, Lake Louise, and Castle Mountain are very close to each other so their lines overlap. Le Massif and Mont Sainte Anne also overlap.

Alberta ski resorts generally have more snow than Quebec ski resorts. From January to March are the best months for Mont Tremblant, but it has the least amount of snow in other months.

5.2) Build an interactive map to visualiza North America ski resorts and snowfall data

I will build a map in the following section to visualize ski resorts and snow fall data. This is especially helpful for ski enthusiastics to determine where to go for their next trip.

I chose Leaflet as the map engine for its interactivity, customizability and rich community supports (such as terrain base map). References are listed here:

- tutorial (https://rstudio.github.io/leaflet/)
- base map providers (http://leaflet-extras.github.io/leaflet-providers/preview/index.html)

```
suppressMessages(library(tidyverse))
```

Create spatial dataframe

```
resort_spatial_df = st_as_sf(
  annual_snow_resort_df,
  coords = c("Longitude", "Latitude"),
  crs = 4326
)

snow_spatial_df = snow_df |> group_by(Latitude, Longitude, continent, country) |> summar
ise(mean_snow=mean(Snow)) |> st_as_sf(
  coords = c("Longitude", "Latitude"),
  crs = 4326
)
```

```
## `summarise()` has grouped output by 'Latitude', 'Longitude', 'continent'. You
## can override using the `.groups` argument.
```

```
snow_spatial_df$mean_snow = round(snow_spatial_df$mean_snow, digits = 0)
```

Create cusomized color palettes

```
custom_snow_palette <- colorBin("BuGn", snow_spatial_df$mean_snow, bins = 5)
custom_price_palette <- colorBin("YlOrRd", resort_spatial_df$Price, bins = 3)
```

Create dataframes for North America data

```
north_america.snow_spatial_df = snow_spatial_df |> filter(continent == "North America")
north_america.resort_spatial_df = resort_spatial_df |> filter(Continent == "North Americ
a")
```

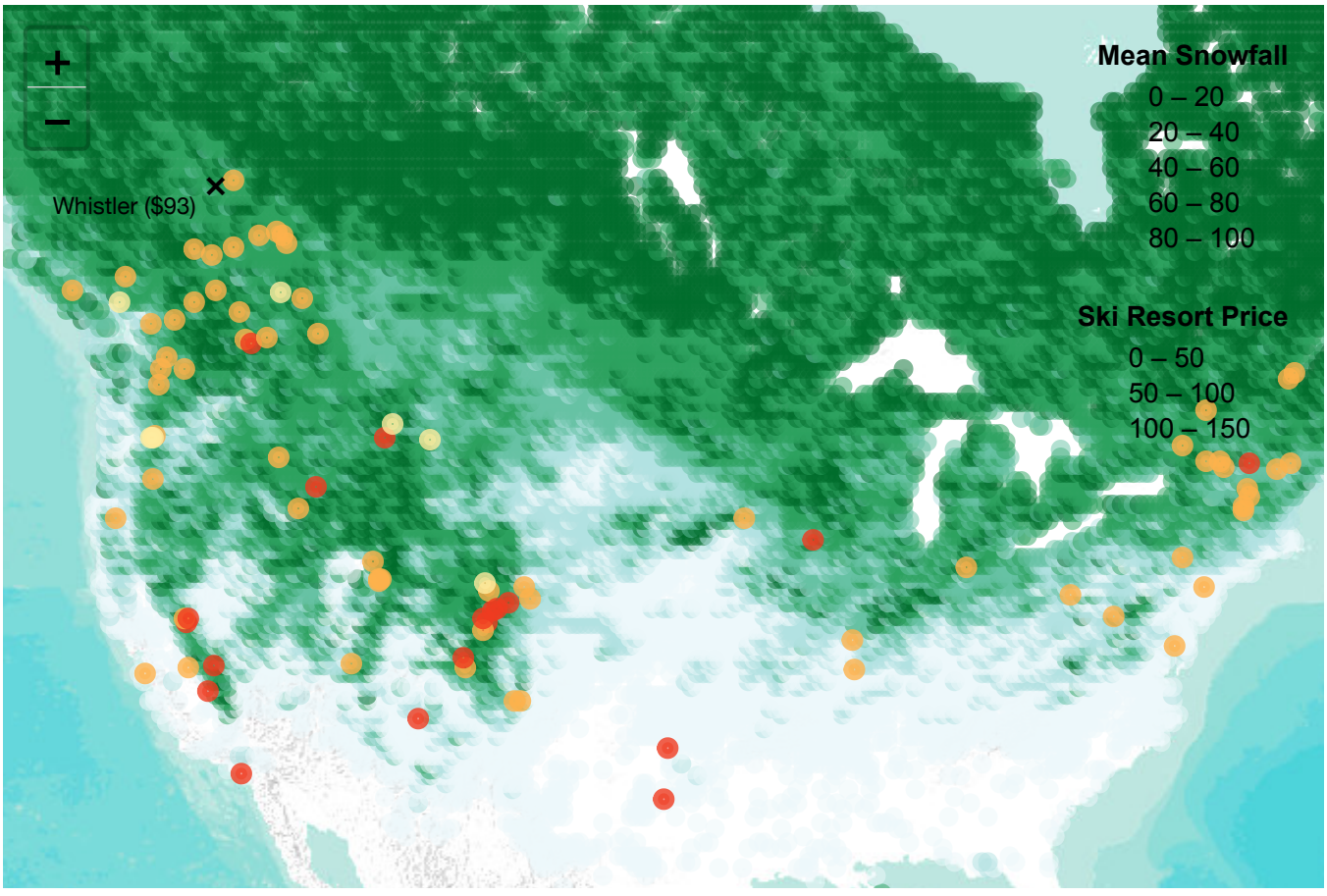
The code below shows ski resorts and average snow fall in North America

```
basemap = leaflet() %>%
          addProviderTiles(providers$Esri.WorldTerrain) |> # Stamen.Terrain
          setView(lng = -98, lat = 44, zoom = 4)
basemap  %>%
  addCircleMarkers(data = north_america.snow_spatial_df,
                   color = custom_snow_palette(north_america.snow_spatial_df$mean_snow),
                   fillOpacity = 0.5,
                  opacity = 0.5,
                    stroke = FALSE,
                   radius = 5) %>%

            addCircleMarkers(data = north_america.resort_spatial_df,
                   color = custom_price_palette(north_america.resort_spatial_df$Price),
                   fillOpacity = 0.5,
                  opacity = 0.8,
                  stroke = TRUE,
                  popup = ~as.character(paste(north_america.resort_spatial_df$Resort, "
($", north_america.resort_spatial_df$Price, ")", sep = "")),
                   radius = 3) |>

          addLegend(pal = custom_snow_palette,
                title = "Mean Snowfall",
                values = north_america.snow_spatial_df$mean_snow) |>

          addLegend(pal = custom_price_palette,
                title = "Ski Resort Price",
                values = north_america.resort_spatial_df$Price)
```



This **interactive** map shows snowfall and ski resorts in North America. You can zoom in/out, drag, click on a ski resort to see its name and price, etc., so it is better to view it in HTML than in PDF.

I choose only to show North America data because showing world-wide data in Leaflet makes R Markdown lag. There are too many data points rendering in Leaflet at the same time regardless of zoom level. I think replacing snow fall data points with raster data or base map could improve this issue, but I couldn't find a raster data nor a base map layer representing snow fall. Therefore, using dots is the best visualization I can do.

6) Build another model using one of the continuous variables from your six most important. This time use your model selection and dimension reduction tools, and include at least one non-linear term. (Mandatory)

Create model evaluation function

```
eval_results <- function(true, predicted) {
  SST <- sum((true - mean(true))^2)
  SSE <- sum((predicted - true)^2)

  R_square <- 1 - SSE / SST
  MSE = SSE/length(true)
  RMSE = sqrt(MSE)
  MAE = mean(abs(pred - true))

  # Model performance metrics
  data.frame(
    Rsquare = R_square,
    RMSE = RMSE,
    MSE = MSE,
    MAE = MAE
  )
}
```

Feature engineering: convert string to factor

```
model_data_df = annual_snow_resort_df

categorical_cols = c("Continent", "Country", "Child.friendly", "Snowparks", "Nightskiin
g", "Summer.skiing")
model_data_df[categorical_cols] = lapply(model_data_df[categorical_cols], factor)
```

6.1) Include a non-linear term

In Q2, we notice the range of Lift.capacity is from 0 to 252280 in our dataset but the price is only from 0 to 140. I will apply a non-linear transformation to Lift.capacity to bring the number down. I will try log and square root and let feature selection technique to find which one is better.

```
model_data_df$Lift.capacity_log = log(model_data_df$Lift.capacity)
model_data_df$Lift.capacity_sqrt = sqrt(model_data_df$Lift.capacity)
```

6.2) Train test split

```
#make this example reproducible
set.seed(1)

# train test split
sample = sample(c(TRUE, FALSE), nrow(model_data_df), replace=TRUE, prob=c(0.8, 0.2))
train_df = model_data_df[sample, ]
test_df = model_data_df[!sample, ]
```

6.3) Stepwise regression to find the most important features

I pass all features to stepwise regression and use AIC as criteria to find the best features

```
train_df = train_df[, !names(train_df) %in% c("Season", "Resort", "ID")]
fit = lm(Price ~ ., data = train_df)
stepwise_fit_AIC = step(fit, k=2, trace=0)
summary(stepwise_fit_AIC)
```

```
##
## Call:
## lm(formula = Price ~ annual_mean_snow + Latitude + Country +
##     Highest.point + Difficult.slopes + Longest.run + Surface.lifts +
##     Nightskiing + season_length + June + Lift.capacity_log, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.783  -3.687  -0.025   3.420  49.254
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.688316  15.026870   0.046 0.963492
## annual_mean_snow         0.063296   0.026172   2.418 0.016122 *
## Latitude                -0.981090   0.237151  -4.137 4.46e-05 ***
## CountryArgentina       -87.596759  22.081929  -3.967 8.91e-05 ***
## CountryAustralia       -38.457041  20.673892  -1.860 0.063740 .
## CountryAustria           9.357037   5.055471   1.851 0.065071 .
## CountryBulgaria         -8.996167   6.885768  -1.306 0.192286
## CountryCanada           24.141024   5.856179   4.122 4.74e-05 ***
## CountryChile           -66.308246  19.749191  -3.358 0.000877 ***
## CountryCzech Republic   10.787040   8.496986   1.270 0.205142
## CountryFinland          36.551857  12.016674   3.042 0.002538 **
## CountryFrance            0.841015   4.980849   0.169 0.866017
## CountryGeorgia         -24.483060  10.815164  -2.264 0.024229 *
## CountryGermany           9.039562   5.509937   1.641 0.101823
## CountryItaly             6.223689   5.199636   1.197 0.232176
## CountryJapan            -5.077214   6.394999  -0.794 0.427797
## CountryLiechtenstein    12.650948  10.769979   1.175 0.240972
## CountryLithuania        21.889888  11.864773   1.845 0.065930 .
## CountryNew Zealand     -56.714630  21.952531  -2.584 0.010204 *
## CountryNorway           32.121767   7.132588   4.504 9.25e-06 ***
## CountryPoland           -3.602467  10.850901  -0.332 0.740100
## CountryRomania           6.160578  10.754399   0.573 0.567137
## CountryRussia          -11.694637   7.392781  -1.582 0.114619
## CountrySerbia           -4.548851  10.715812  -0.424 0.671475
## CountrySlovakia          6.406583   6.929599   0.925 0.355881
## CountrySlovenia          6.795303   8.369323   0.812 0.417410
## CountrySouth Korea      18.223655  10.873237   1.676 0.094672 .
## CountrySpain             2.253762   6.197522   0.364 0.716346
## CountrySweden           29.591571   8.494357   3.484 0.000560 ***
## CountrySwitzerland      16.718645   5.118841   3.266 0.001204 **
## CountryTurkey          -29.448363  10.792572  -2.729 0.006697 **
## CountryUkraine           0.200418  10.871100   0.018 0.985302
## CountryUnited Kingdom   21.858151   8.008633   2.729 0.006683 **
## CountryUnited States    39.427545   5.204624   7.575 3.55e-13 ***
## Highest.point            0.003327   0.001013   3.283 0.001136 **
## Difficult.slopes         0.207092   0.042018   4.929 1.31e-06 ***
## Longest.run              0.281995   0.157247   1.793 0.073826 .
## Surface.lifts           -0.245865   0.068241  -3.603 0.000363 ***
## NightskiingYes          -2.084251   1.169033  -1.783 0.075513 .
## season_length           -0.688737   0.500902  -1.375 0.170055
```

```
## June                    10.117418   3.669041   2.758 0.006145 **
## Lift.capacity_log        7.266188   0.954301   7.614 2.75e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.521 on 334 degrees of freedom
## Multiple R-squared:  0.821,  Adjusted R-squared:  0.799
## F-statistic: 37.36 on 41 and 334 DF,  p-value: < 2.2e-16
```

Calculate the R-square of the AIC model

```
# Best stepwise model using AIC
AIC_step_model = lm (Price ~ annual_mean_snow + Latitude + Country +
    Highest.point + Difficult.slopes + Longest.run + Surface.lifts +
    Nightskiing + season_length + June + Lift.capacity_log,
                 data = train_df)

summary(AIC_step_model)$r.squared
```

```
## [1] 0.8209787
```

11 features are selected by AIC stepwise regression and the R-squared is 0.8209787. In addition, the model prefers to use log of Lift.capacity over Lift.capacity or square root of Lift.capacity.

Now let's try using BIC as the criteria to reduce the number of features

```
stepwise_fit_BIC = step(fit, k=log(nrow(train_df)), trace=0)
summary(stepwise_fit_BIC)
```

```
##
## Call:
## lm(formula = Price ~ annual_mean_snow + Latitude + Country +
##     Highest.point + Difficult.slopes + Surface.lifts + June +
##     Lift.capacity_log, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.082  -4.043   0.000   3.665  50.697
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -1.344e+00  1.473e+01  -0.091 0.927321
## annual_mean_snow       7.076e-02  2.613e-02   2.708 0.007124 **
## Latitude              -9.988e-01  2.377e-01  -4.202 3.40e-05 ***
## CountryArgentina      -9.026e+01  2.188e+01  -4.125 4.67e-05 ***
## CountryAustralia      -3.875e+01  2.059e+01  -1.882 0.060658 .
## CountryAustria         1.001e+01  5.084e+00   1.970 0.049703 *
## CountryBulgaria       -8.893e+00  6.836e+00  -1.301 0.194195
## CountryCanada          2.405e+01  5.891e+00   4.082 5.58e-05 ***
## CountryChile          -6.553e+01  1.969e+01  -3.329 0.000969 ***
## CountryCzech Republic  9.311e+00  8.539e+00   1.090 0.276310
## CountryFinland         3.496e+01  1.207e+01   2.896 0.004021 **
## CountryFrance          4.673e-01  4.995e+00   0.094 0.925524
## CountryGeorgia        -2.733e+01  1.080e+01  -2.530 0.011878 *
## CountryGermany         8.906e+00  5.540e+00   1.608 0.108837
## CountryItaly           5.899e+00  5.231e+00   1.128 0.260304
## CountryJapan          -7.430e+00  6.381e+00  -1.164 0.245104
## CountryLiechtenstein   1.330e+01  1.085e+01   1.226 0.221169
## CountryLithuania       1.662e+01  1.172e+01   1.418 0.157116
## CountryNew Zealand    -5.742e+01  2.199e+01  -2.612 0.009418 **
## CountryNorway          3.066e+01  7.158e+00   4.284 2.40e-05 ***
## CountryPoland         -5.046e+00  1.091e+01  -0.463 0.643972
## CountryRomania         4.055e+00  1.080e+01   0.376 0.707461
## CountryRussia         -1.133e+01  7.442e+00  -1.522 0.128981
## CountrySerbia         -6.312e+00  1.076e+01  -0.587 0.557699
## CountrySlovakia        6.687e+00  6.980e+00   0.958 0.338744
## CountrySlovenia        5.904e+00  8.406e+00   0.702 0.482913
## CountrySouth Korea     1.565e+01  1.091e+01   1.434 0.152439
## CountrySpain           1.511e+00  6.223e+00   0.243 0.808277
## CountrySweden          2.863e+01  8.514e+00   3.363 0.000859 ***
## CountrySwitzerland     1.710e+01  5.152e+00   3.320 0.001000 **
## CountryTurkey         -3.013e+01  1.083e+01  -2.783 0.005696 **
## CountryUkraine        -2.509e+00  1.091e+01  -0.230 0.818164
## CountryUnited Kingdom  2.179e+01  8.050e+00   2.707 0.007143 **
## CountryUnited States   3.825e+01  5.222e+00   7.324 1.78e-12 ***
## Highest.point          3.716e-03  9.762e-04   3.806 0.000167 ***
## Difficult.slopes       2.081e-01  4.214e-02   4.939 1.24e-06 ***
## Surface.lifts         -2.697e-01  6.662e-02  -4.048 6.42e-05 ***
## June                   8.970e+00  3.126e+00   2.870 0.004366 **
## Lift.capacity_log      7.115e+00  8.805e-01   8.081 1.16e-14 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.592 on 337 degrees of freedom
## Multiple R-squared:  0.8167, Adjusted R-squared:  0.796
## F-statistic:  39.5 on 38 and 337 DF,  p-value: < 2.2e-16
```

Calculate the R-square of the BIC model

```
# Best stepwise model using BIC
BIC_step_model = lm (Price ~ annual_mean_snow + Latitude + Country +
    Highest.point + Difficult.slopes + Surface.lifts + June +
    Lift.capacity_log,
                 data = train_df)

summary(BIC_step_model)$r.squared
```

```
## [1] 0.8166614
```

BIC stepwise regression eliminates 3 features (Longest.run, Nightskiing and season_length). The remaining features are annual mean snow, latitude, country, highest point, number of difficult slopes, number of surface lifts, whether resort is open in June, and log of lift capacity. The R-square decreases from 0.8209787 to 0.8166614, which is acceptable.

6.4) Evaluation on test set

Let's exam our model on test dataset.

```
test_df_new = test_df[test_df$Country %in% unique(train_df$Country), ]
pred = predict(BIC_step_model, newdata = test_df_new)

eval_results(test_df_new$Price, pred)
```

Rsquare	RMSE	MSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>
0.7716454	9.858073	97.18161	6.633442

1 row

Our model achieves R-square of 0.77 on test set. On average, our price prediction is off by $6.6 from the true price. I would say this is a descent regression model to predict world-wide ski resort price.

7) Discuss briefly the steps you would take to make sure your analysis is reproducible and easy to evaluate by others, even if the data is updated later. (Option)

Skipped.

8) Discuss briefly any ethical concerns like residual disclosure that might arise from the use of your data set, possibly in combination with some additional data outside your dataset. (Option)

There is little ethical concern about this dataset as the information about ski resorts is publicly available and the information expose no harm or risk to any individual or institution.