STA380 Excercises

Visual story telling

One of the worst part of flying is all the extra time spent dealing with delays. They can make you late for important deadlines and cause unnecessary stress in an already stressful experience. We will analyze this dataset of flights to and from the Austin airport to provide helpful insights on trends in delayed flights.

A key for airlines abbreviations to help for the rest of the visuals: American Airlines (AA),

```
library(mosaic)
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

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

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

## Loading required package: lattice

## Loading required package: ggformula

## Loading required package: ggplot2

## Loading required package: ggstance

##
## Attaching package: 'ggstance'

## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh

##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")
```

```
## Loading required package: mosaicData

## Loading required package: Matrix

## Registered S3 method overwritten by 'mosaic':
##   method                           from
##   fortify.SpatialPolygonsDataFrame ggplot2

##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.
##
## Have you tried the ggformula package for your plots?

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following object is masked from 'package:ggplot2':
##
##     stat

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------

## v tibble  3.0.1     v purrr   0.3.4
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

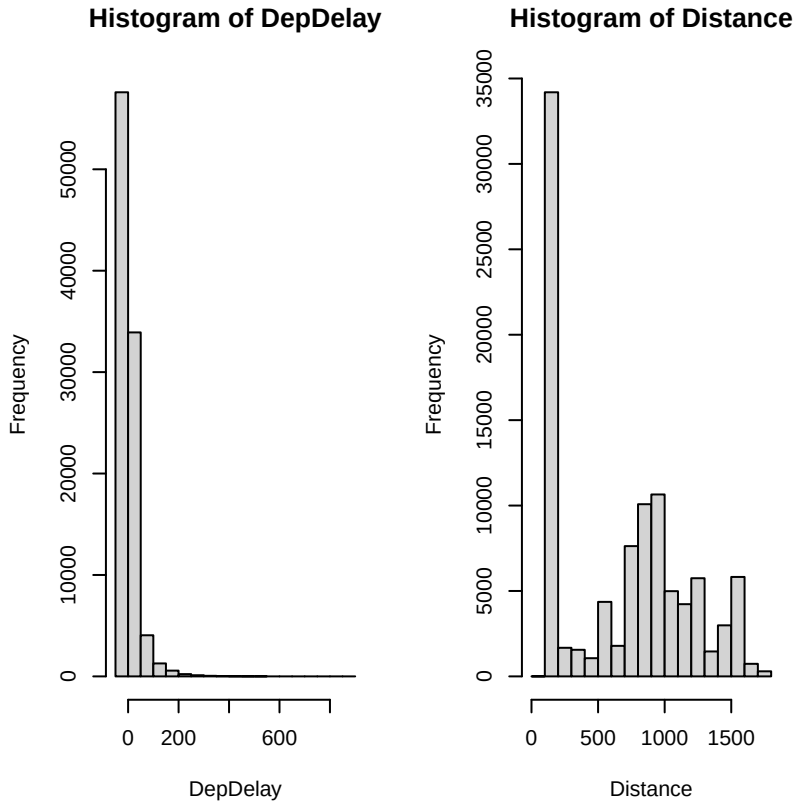
```
## -- Conflicts ------------------------------------------------------------------------------- tidyv
## x mosaic::count()            masks dplyr::count()
## x purrr::cross()             masks mosaic::cross()
## x mosaic::do()               masks dplyr::do()
## x tidyr::expand()            masks Matrix::expand()
## x dplyr::filter()            masks stats::filter()
## x ggstance::geom_errorbarh() masks ggplot2::geom_errorbarh()
## x dplyr::lag()               masks stats::lag()
## x tidyr::pack()              masks Matrix::pack()
## x mosaic::stat()             masks ggplot2::stat()
## x mosaic::tally()            masks dplyr::tally()
## x tidyr::unpack()            masks Matrix::unpack()
```

```
library(ggplot2)
abia = read.csv("~/MSBAsummer20/STA380-master/data/ABIA.csv", stringsAsFactors=FALSE)
attach(abia)
depDelay_clean = abia[!is.na(abia$DepDelay), ]
arrDelay_clean = abia[!is.na(abia$ArrDelay), ]
summary(abia)
```

```
##       Year          Month          DayofMonth      DayOfWeek        DepTime
##  Min.   :2008   Min.   : 1.00   Min.   : 1.00   Min.   :1.000   Min.   :   1
##  1st Qu.:2008   1st Qu.: 3.00   1st Qu.: 8.00   1st Qu.:2.000   1st Qu.: 917
##  Median :2008   Median : 6.00   Median :16.00   Median :4.000   Median :1329
##  Mean   :2008   Mean   : 6.29   Mean   :15.73   Mean   :3.902   Mean   :1329
##  3rd Qu.:2008   3rd Qu.: 9.00   3rd Qu.:23.00   3rd Qu.:6.000   3rd Qu.:1728
##  Max.   :2008   Max.   :12.00   Max.   :31.00   Max.   :7.000   Max.   :2400
##                                                                 NA's   :1413
##    CRSDepTime      ArrTime       CRSArrTime   UniqueCarrier        FlightNum
##  Min.   :  55   Min.   :   1   Min.   :   5   Length:99260       Min.   :   1
##  1st Qu.: 915   1st Qu.:1107   1st Qu.:1115   Class :character   1st Qu.: 640
##  Median :1320   Median :1531   Median :1535   Mode  :character   Median :1465
##  Mean   :1320   Mean   :1487   Mean   :1505                      Mean   :1917
##  3rd Qu.:1720   3rd Qu.:1903   3rd Qu.:1902                      3rd Qu.:2653
##  Max.   :2346   Max.   :2400   Max.   :2400                      Max.   :9741
##                 NA's   :1567
##    TailNum          ActualElapsedTime CRSElapsedTime     AirTime
##  Length:99260       Min.   : 22.0     Min.   : 17.0   Min.   :  3.00
##  Class :character   1st Qu.: 57.0     1st Qu.: 58.0   1st Qu.: 38.00
##  Mode  :character   Median :125.0     Median :130.0   Median :105.00
##                     Mean   :120.2     Mean   :122.1   Mean   : 99.81
##                     3rd Qu.:164.0     3rd Qu.:165.0   3rd Qu.:142.00
##                     Max.   :506.0     Max.   :320.0   Max.   :402.00
##                     NA's   :1601      NA's   :11      NA's   :1601
##     ArrDelay          DepDelay          Origin              Dest
##  Min.   :-129.000   Min.   :-42.000   Length:99260       Length:99260
##  1st Qu.:  -9.000   1st Qu.: -4.000   Class :character   Class :character
##  Median :  -2.000   Median :  0.000   Mode  :character   Mode  :character
##  Mean   :   7.065   Mean   :  9.171
##  3rd Qu.:  10.000   3rd Qu.:  8.000
##  Max.   : 948.000   Max.   :875.000
##  NA's   :1601       NA's   :1413
##     Distance        TaxiIn           TaxiOut         Cancelled
##  Min.   :  66   Min.   :  0.000   Min.   :  1.00   Min.   :0.00000
```

```
##  1st Qu.: 190   1st Qu.:  4.000   1st Qu.:  9.00   1st Qu.:0.00000
##  Median : 775   Median :  5.000   Median : 12.00   Median :0.00000
##  Mean   : 705   Mean   :  6.413   Mean   : 13.96   Mean   :0.01431
##  3rd Qu.:1085   3rd Qu.:  7.000   3rd Qu.: 16.00   3rd Qu.:0.00000
##  Max.   :1770   Max.   :143.000   Max.   :305.00   Max.   :1.00000
##                 NA's   :1567      NA's   :1419
##  CancellationCode      Diverted         CarrierDelay     WeatherDelay
##  Length:99260       Min.   :0.000000   Min.   :  0.00   Min.   :  0.00
##  Class :character   1st Qu.:0.000000   1st Qu.:  0.00   1st Qu.:  0.00
##  Mode  :character   Median :0.000000   Median :  0.00   Median :  0.00
##                     Mean   :0.001824   Mean   : 15.39   Mean   :  2.24
##                     3rd Qu.:0.000000   3rd Qu.: 16.00   3rd Qu.:  0.00
##                     Max.   :1.000000   Max.   :875.00   Max.   :412.00
##                                        NA's   :79513    NA's   :79513
##     NASDelay      SecurityDelay    LateAircraftDelay
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00
##  Median :  2.00   Median :  0.00   Median :  6.00
##  Mean   : 12.47   Mean   :  0.07   Mean   : 22.97
##  3rd Qu.: 16.00   3rd Qu.:  0.00   3rd Qu.: 30.00
##  Max.   :367.00   Max.   :199.00   Max.   :458.00
##  NA's   :79513    NA's   :79513    NA's   :79513
```

```
par(mfrow=c(1,3))
hist(DepDelay)
hist(Distance)
```

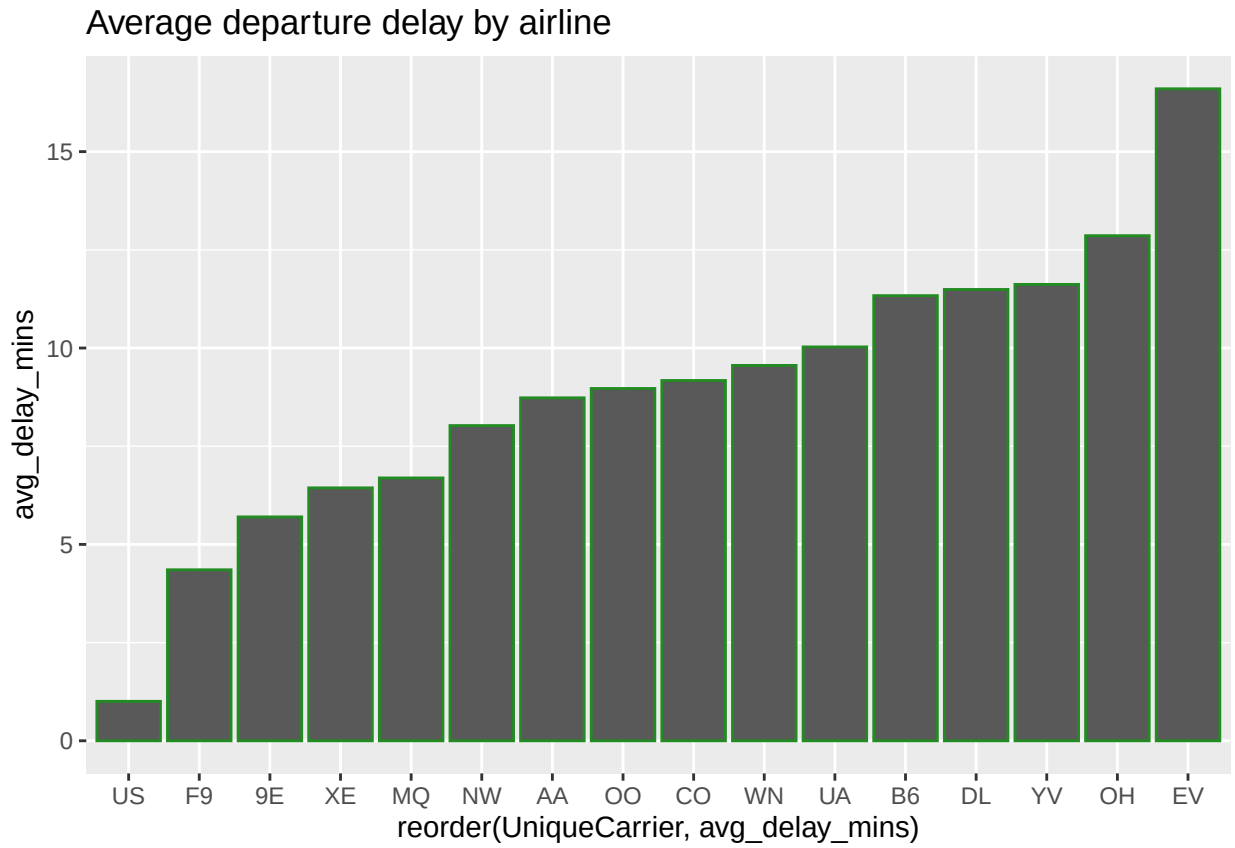


Which airline has the worst delays? (nned to add arrival delays)

```
depDelay_clean = abia[!is.na(abia$DepDelay), ]
airline_delay = depDelay_clean %>%
  group_by(UniqueCarrier)  %>%
  summarize(avg_delay_mins = mean(DepDelay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
ggplot(airline_delay, aes(x=reorder(UniqueCarrier, avg_delay_mins), y=avg_delay_mins)) +
  geom_bar(stat='identity', color='forest green') +
  labs(title = "Average departure delay by airline")
```

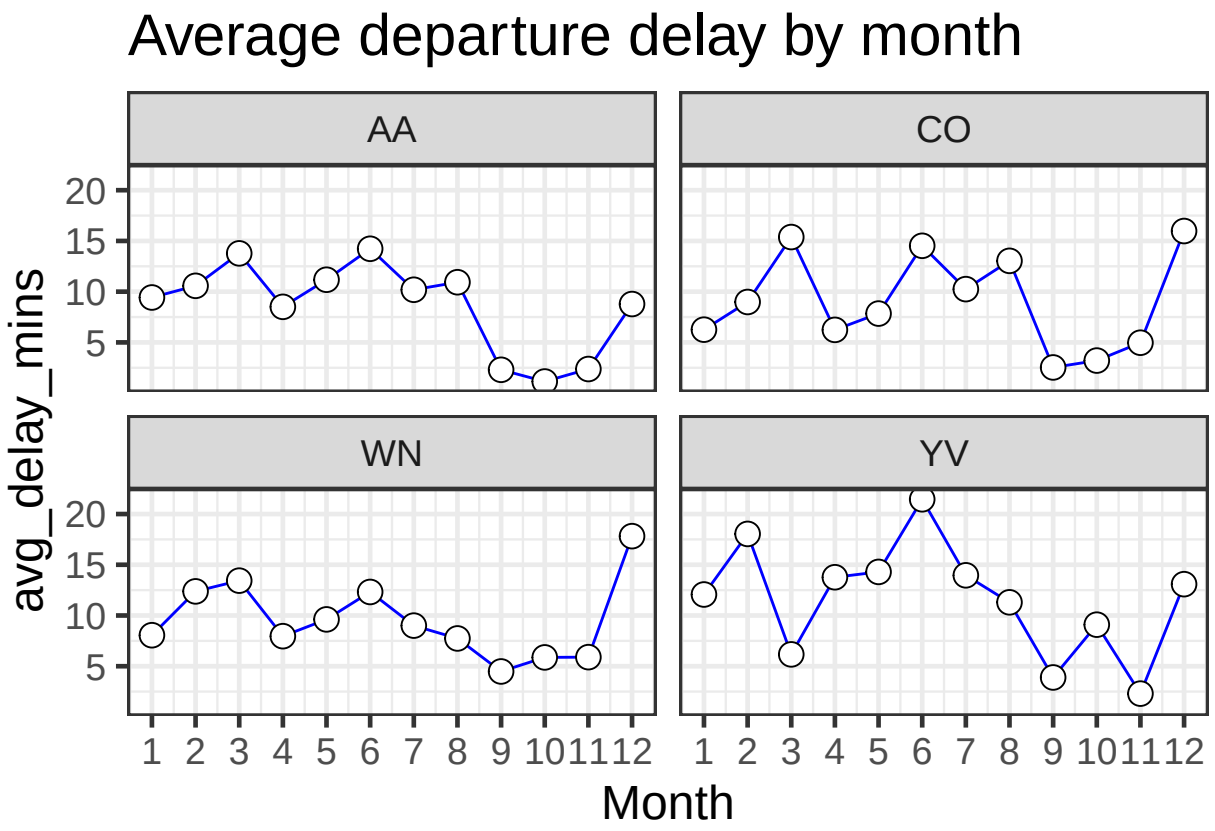


When should you leave?

```
airline_list = c('AA', 'WN', 'CO', 'YV')

flights_per_month = depDelay_clean %>%
  filter(UniqueCarrier %in% airline_list) %>%
  group_by(UniqueCarrier, Month) %>%
  summarize(avg_delay_mins = mean(DepDelay))
```

```
## `summarise()` regrouping output by 'UniqueCarrier' (override with `.groups` argument)
```

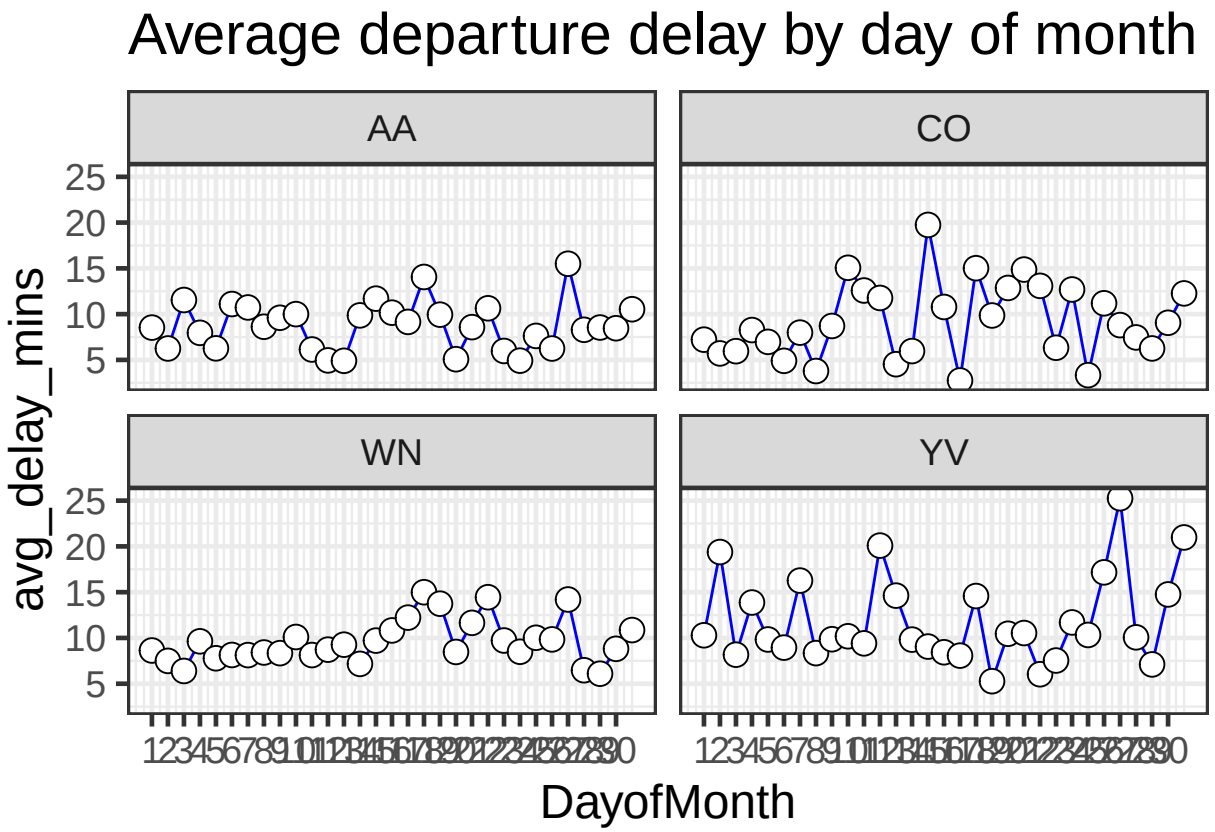
```
ggplot(flights_per_month, aes(x=Month, y=avg_delay_mins)) +
  geom_line( color='blue') +
  geom_point( size=4, shape=21, fill="white") +
  facet_wrap(~ UniqueCarrier, nrow = 2) +
  theme_bw(base_size=18) +
  scale_x_continuous(breaks = 1:12) +
  labs(title = "Average departure delay by month")
```




```
flights_per_day = depDelay_clean %>%
  filter(UniqueCarrier %in% airline_list) %>%
  group_by(UniqueCarrier, DayofMonth) %>%
  summarize(avg_delay_mins = mean(DepDelay))
```

```
## `summarise()` regrouping output by 'UniqueCarrier' (override with `.groups` argument)
```

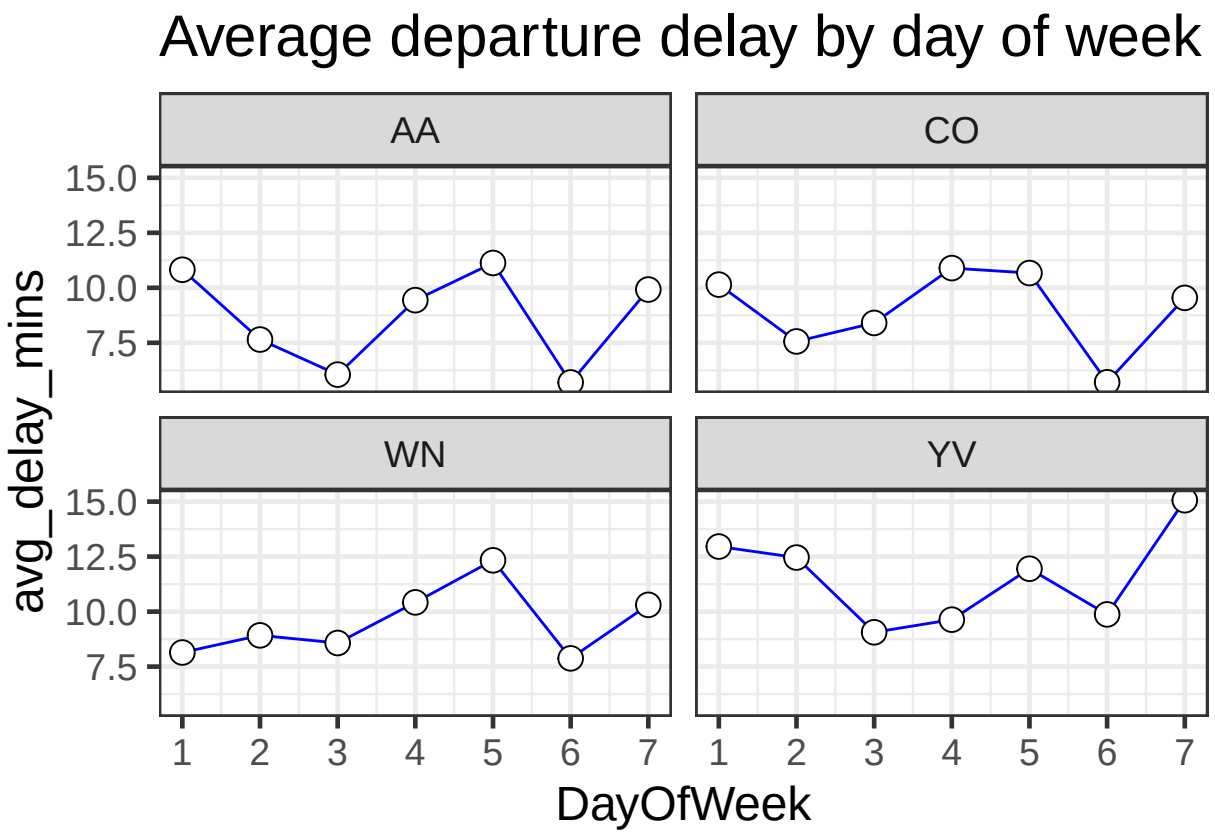
```
ggplot(flights_per_day, aes(x=DayofMonth, y=avg_delay_mins)) +
  geom_line( color='blue') +
  geom_point( size=4, shape=21, fill="white") +
  facet_wrap(~ UniqueCarrier, nrow = 2) +
  theme_bw(base_size=18) +
  scale_x_continuous(breaks = 1:30) +
  labs(title = "Average departure delay by day of month")
```




```
flights_per_week = depDelay_clean %>%
  filter(UniqueCarrier %in% airline_list) %>%
  group_by(UniqueCarrier, DayOfWeek) %>%
  summarize(avg_delay_mins = mean(DepDelay))
```

```
## `summarise()` regrouping output by 'UniqueCarrier' (override with `.groups` argument)
```

```
ggplot(flights_per_week, aes(x=DayOfWeek, y=avg_delay_mins)) +
  geom_line( color='blue') +
  geom_point( size=4, shape=21, fill="white") +
  facet_wrap(~ UniqueCarrier, nrow = 2) +
  theme_bw(base_size=18) +
  scale_x_continuous(breaks = 1:31) +
  labs(title = "Average departure delay by day of week")
```

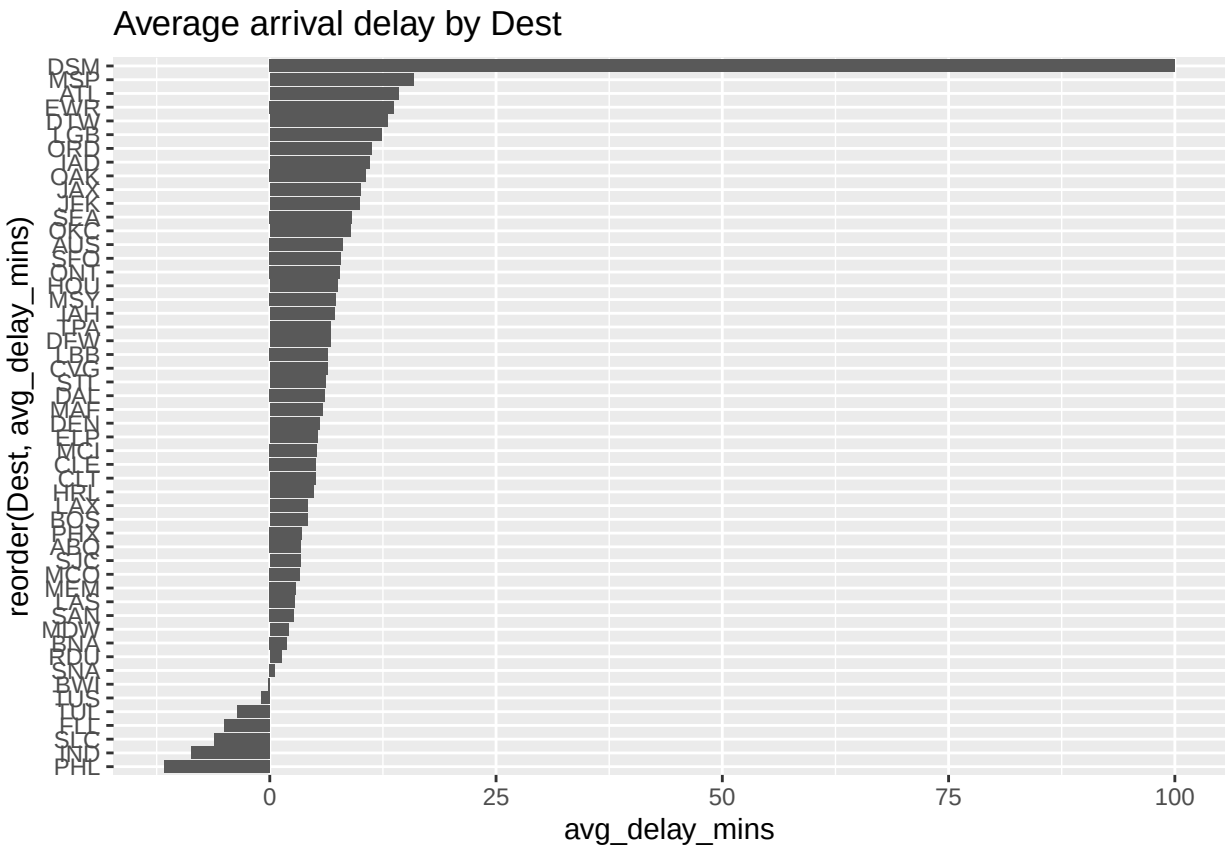


Which destinations are the worst in delaying when you arrive?

```
arr_delay_dest = arrDelay_clean %>%
  group_by(Dest) %>%
  summarize(avg_delay_mins = mean(ArrDelay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
ggplot(arr_delay_dest, aes(x=reorder(Dest, avg_delay_mins), y=avg_delay_mins)) +
  geom_bar(stat='identity') +
  labs(title = "Average arrival delay by Dest") +
  coord_flip()
```

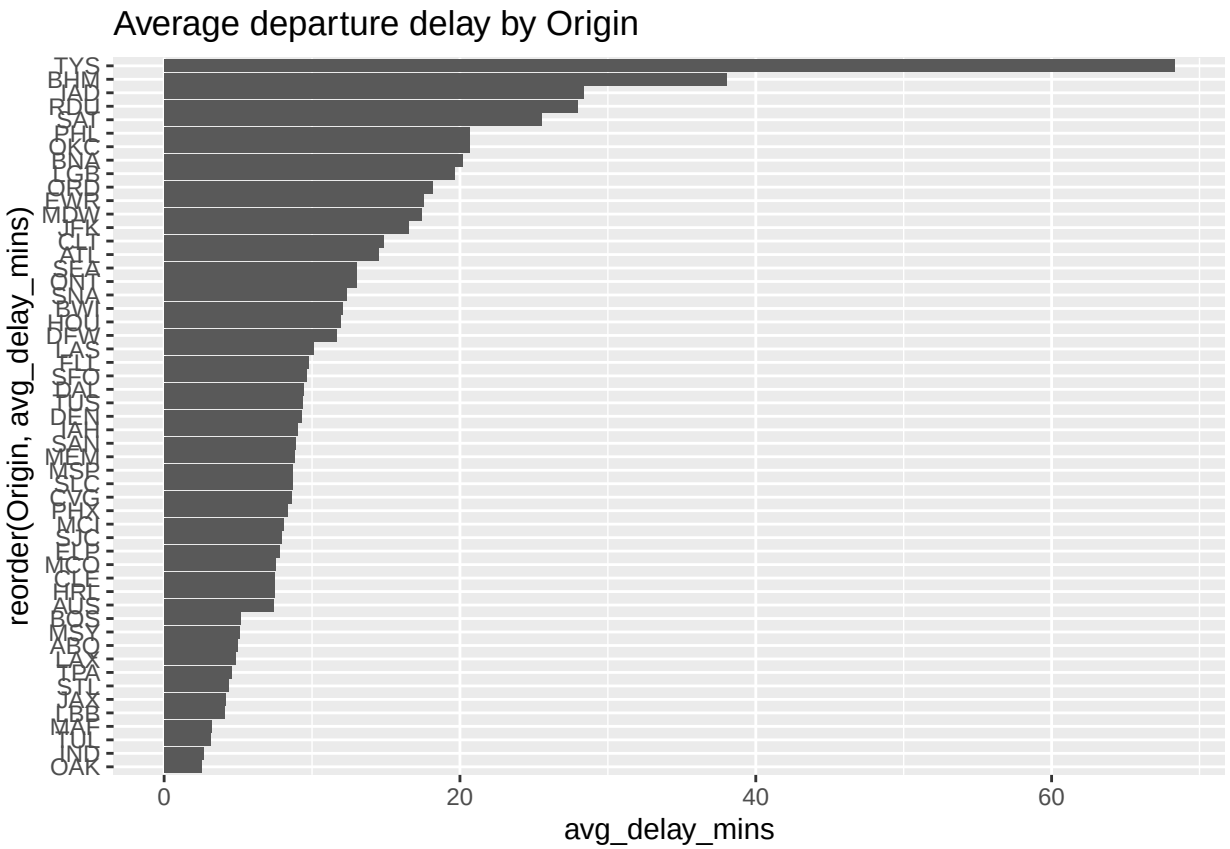


Which places are the worst to leave from in terms of departure delay?

```
dep_delay_org = depDelay_clean %>%
  group_by(Origin) %>%
  summarize(avg_delay_mins = mean(DepDelay))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
ggplot(dep_delay_org, aes(x=reorder(Origin, avg_delay_mins), y=avg_delay_mins)) +
  geom_bar(stat='identity') +
  labs(title = "Average departure delay by Origin") +
  coord_flip()
```



Association Rule Mining

This data set contains 9835 baskets of groceries. The most frequent items are seen below with a minimum support of 0.1 meaning they show up in 10% of all baskets.

```
library(arules)
```

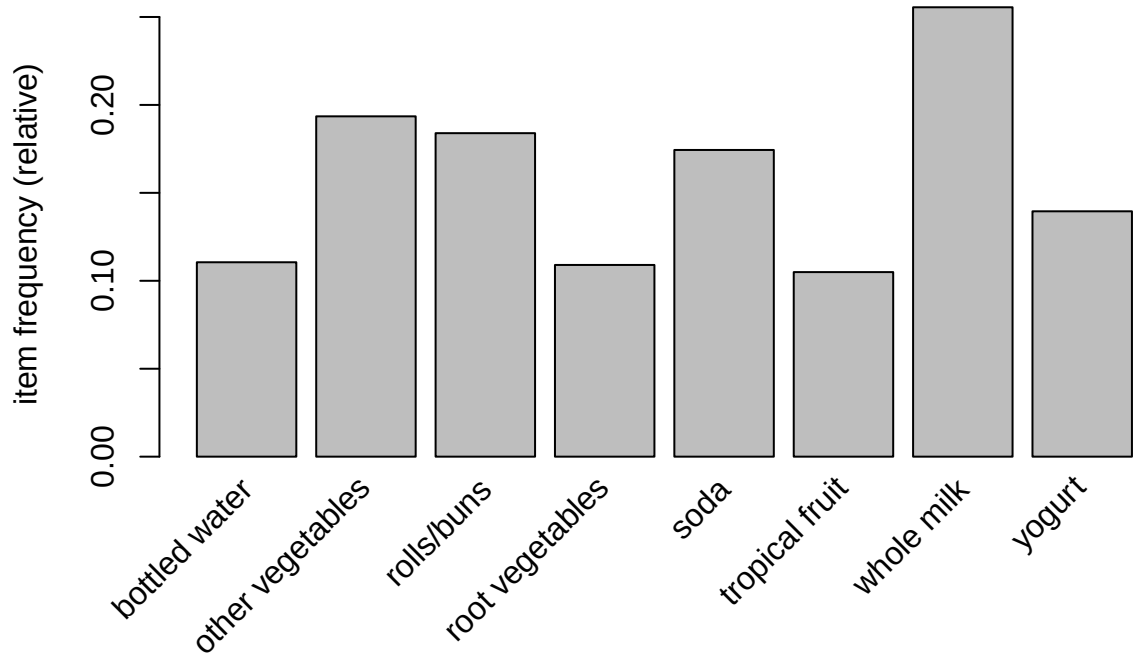
```
##
## Attaching package: 'arules'

## The following objects are masked from 'package:mosaic':
##
##     inspect, lhs, rhs

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
library(arulesViz)
```

```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

```
groceries = read.transactions("~/MSBAsummer20/groceries.txt", rm.duplicates=TRUE, format="basket", sep=
itemFrequencyPlot(groceries, support=0.1)
```



Next, lets examine the most common rules seen in the shopping trends with a support of 0.05 and a confidence of 0.1. Confidence indicates how often the rule is true, so this will give us rules that will be true 10% of the time. A high support and confidence will select out the most common purchases made together most of the time. From this subset, we sort the top 10 by lift to examine the rules that have the strongest correlations between the left hand items and right hand items of the rule.

```
grocery_rules = apriori(groceries, parameter=list(support=.05, confidence=.1))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.1    0.1    1 none FALSE            TRUE       5    0.05      1
```

```
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 491
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [28 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [14 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(grocery_rules, by = "lift"), 10))
```

```
##      lhs                   rhs                support    confidence coverage
## [1]  {yogurt}           => {whole milk}       0.05602440 0.4016035  0.1395018
## [2]  {whole milk}       => {yogurt}           0.05602440 0.2192598  0.2555160
## [3]  {other vegetables} => {whole milk}       0.07483477 0.3867578  0.1934926
## [4]  {whole milk}       => {other vegetables} 0.07483477 0.2928770  0.2555160
## [5]  {rolls/buns}       => {whole milk}       0.05663447 0.3079049  0.1839349
## [6]  {whole milk}       => {rolls/buns}       0.05663447 0.2216474  0.2555160
## [7]  {}                 => {yogurt}           0.13950178 0.1395018  1.0000000
## [8]  {}                 => {rolls/buns}       0.18393493 0.1839349  1.0000000
## [9]  {}                 => {bottled water}    0.11052364 0.1105236  1.0000000
## [10] {}                 => {tropical fruit}   0.10493137 0.1049314  1.0000000
##      lift     count
## [1]  1.571735  551
## [2]  1.571735  551
## [3]  1.513634  736
## [4]  1.513634  736
## [5]  1.205032  557
## [6]  1.205032  557
## [7]  1.000000 1372
## [8]  1.000000 1809
## [9]  1.000000 1087
## [10] 1.000000 1032
```

Whole Milk Subset

Focusing on the most common item, whole milk, we generate a set of rules with a support of 0.001 and a confidence of 0.08. Selecting a unique subset, we lower the support but only lower the confidence slightly to keep strong rules.

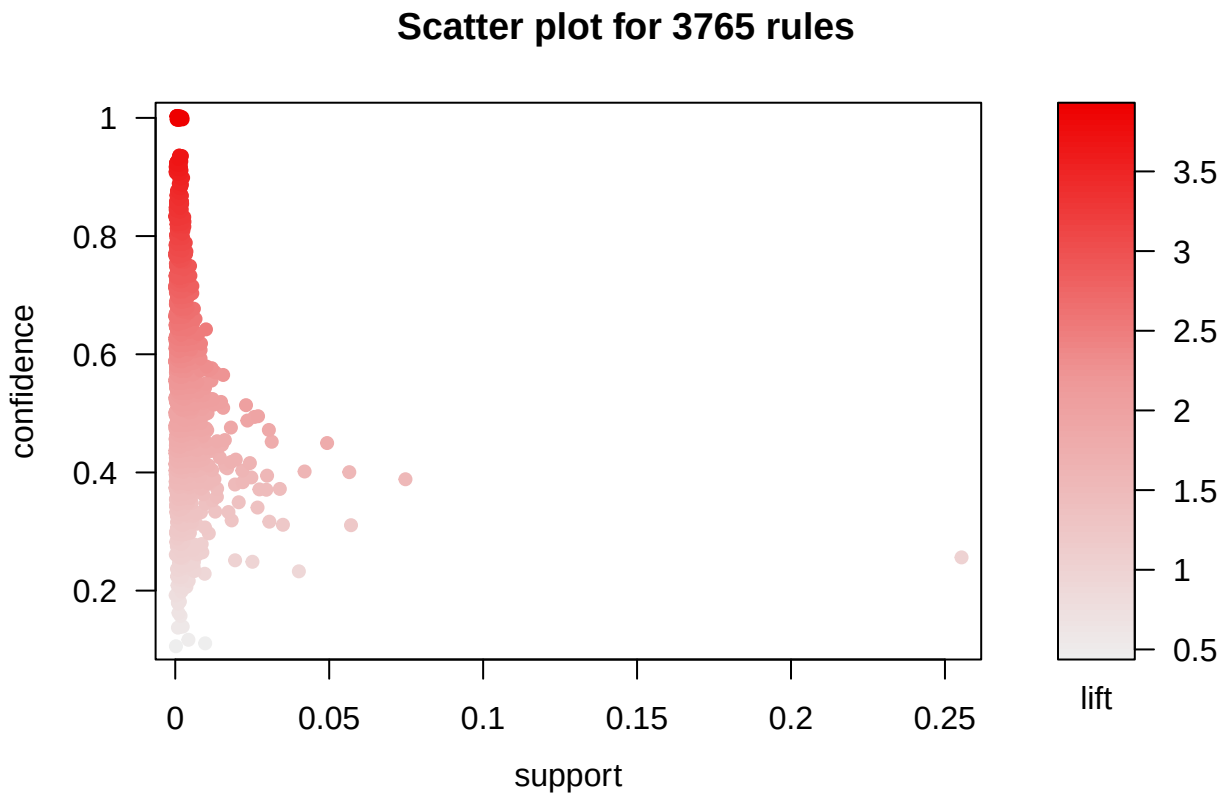
```
milk_rules <- apriori(data=groceries, parameter=list (supp=0.001,conf = 0.08), appearance = list (rhs="
```

```
## Apriori
```

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.08    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [3765 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
plot(milk_rules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



The top ten rules by support.

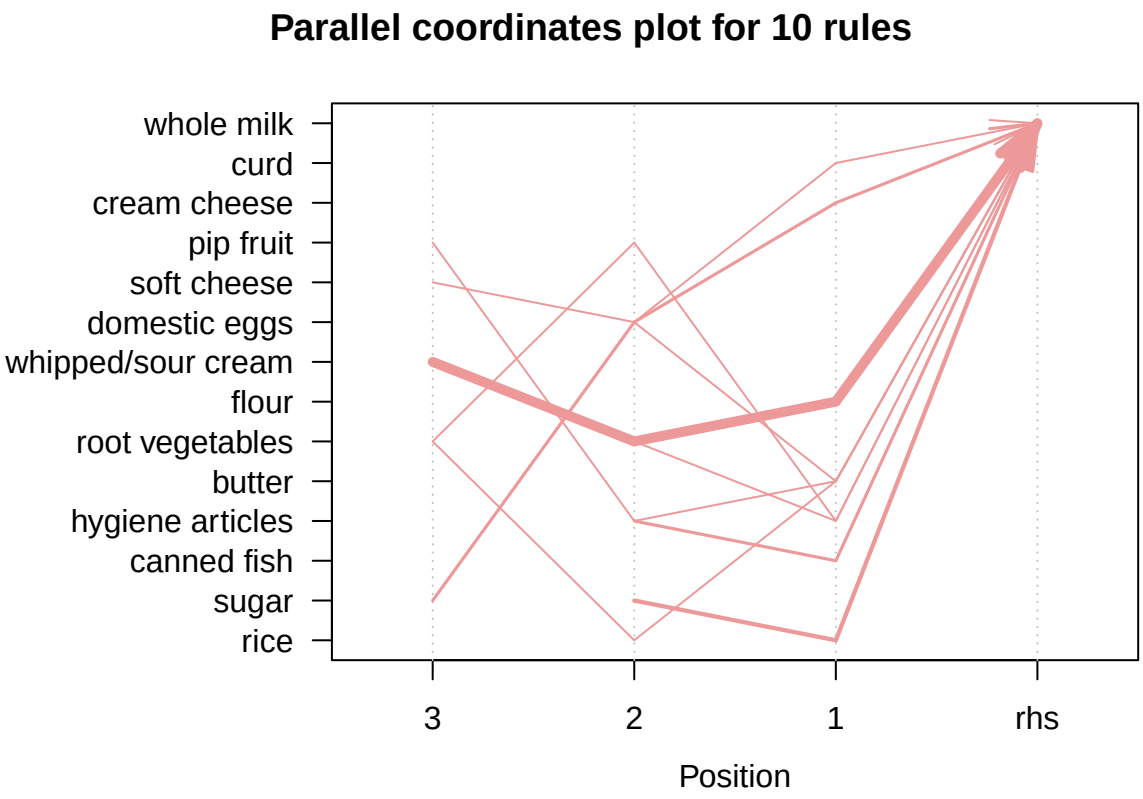
```
##      lhs                    rhs          support    confidence coverage
## [1]  {}                  => {whole milk} 0.25551601 0.2555160  1.00000000
## [2]  {other vegetables}  => {whole milk} 0.07483477 0.3867578  0.19349263
## [3]  {rolls/buns}        => {whole milk} 0.05663447 0.3079049  0.18393493
## [4]  {yogurt}            => {whole milk} 0.05602440 0.4016035  0.13950178
## [5]  {root vegetables}   => {whole milk} 0.04890696 0.4486940  0.10899847
## [6]  {tropical fruit}    => {whole milk} 0.04229792 0.4031008  0.10493137
## [7]  {soda}              => {whole milk} 0.04006101 0.2297376  0.17437722
## [8]  {bottled water}     => {whole milk} 0.03436706 0.3109476  0.11052364
## [9]  {pastry}            => {whole milk} 0.03324860 0.3737143  0.08896797
## [10] {whipped/sour cream} => {whole milk} 0.03223183 0.4496454  0.07168277
##      lift      count
## [1]  1.0000000 2513
## [2]  1.5136341  736
## [3]  1.2050318  557
## [4]  1.5717351  551
## [5]  1.7560310  481
## [6]  1.5775950  416
## [7]  0.8991124  394
## [8]  1.2169396  338
## [9]  1.4625865  327
## [10] 1.7597542  317
```

The top ten rules by confidence.

```
##      lhs                    rhs              support confidence    coverage     lift count
## [1]  {rice,
##       sugar}              => {whole milk} 0.001220132          1 0.001220132 3.913649    12
## [2]  {canned fish,
##       hygiene articles}   => {whole milk} 0.001118454          1 0.001118454 3.913649    11
## [3]  {butter,
##       rice,
##       root vegetables}    => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [4]  {flour,
##       root vegetables,
##       whipped/sour cream} => {whole milk} 0.001728521          1 0.001728521 3.913649    17
## [5]  {butter,
##       domestic eggs,
##       soft cheese}        => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [6]  {butter,
##       hygiene articles,
##       pip fruit}          => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [7]  {hygiene articles,
##       root vegetables,
##       whipped/sour cream} => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [8]  {hygiene articles,
##       pip fruit,
##       root vegetables}    => {whole milk} 0.001016777          1 0.001016777 3.913649    10
## [9]  {cream cheese,
##       domestic eggs,
##       sugar}              => {whole milk} 0.001118454          1 0.001118454 3.913649    11
## [10] {curd,
##       domestic eggs,
##       sugar}              => {whole milk} 0.001016777          1 0.001016777 3.913649    10
```

The

```
subrulesmilk <- head(milk_rules, n = 10, by = "lift")
plot(subrulesmilk, method = "paracoord")
```



Examining the results for whole milk, baskets show trends of dairy products being bought together.

Alcohol Subset

Looking closer at the purchase patterns associated with alcohol purchases such as red wine, beer and liquor, allows the store to understand common trends about their premium items. We lower the support slightly as these are less common items purchased while still keeping the confidence the same to show strong rules.

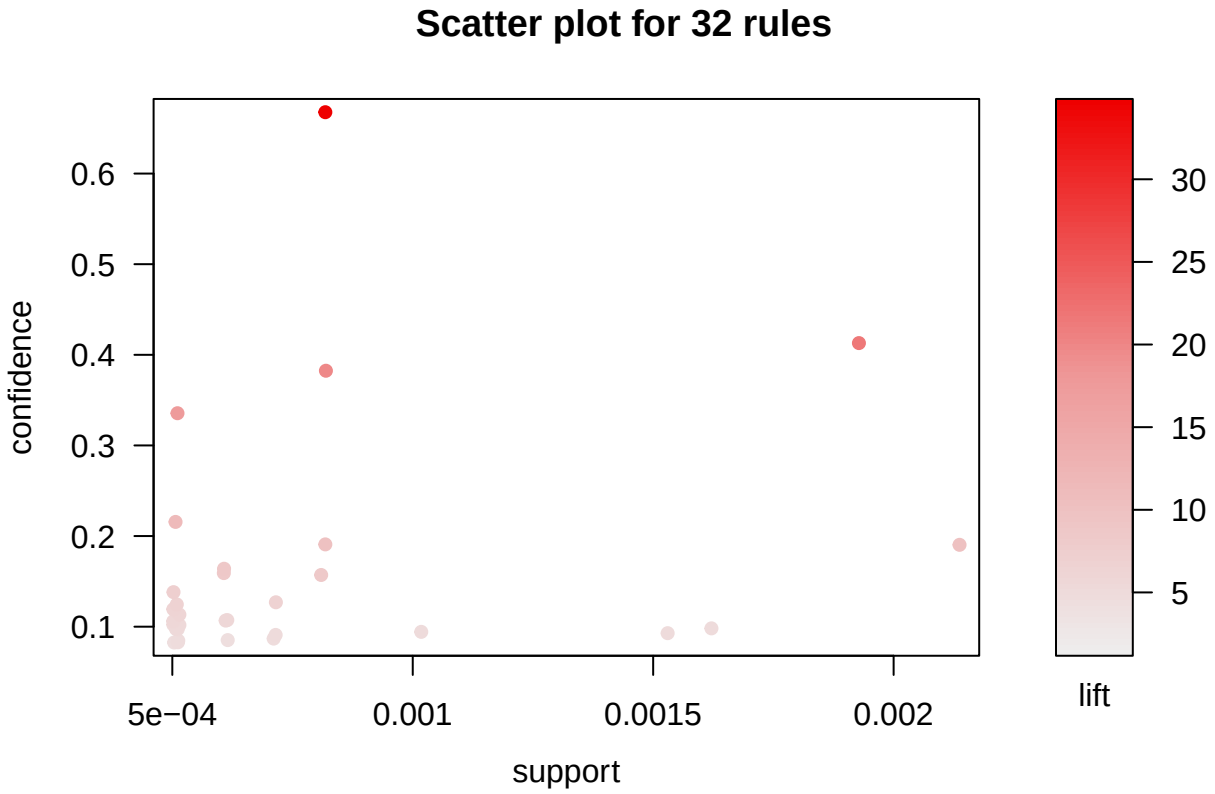
```
redwine_rules <- apriori(data=groceries, parameter=list (supp=0.0005,conf = 0.08), appearance = list (rh
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.08    0.1    1 none FALSE            TRUE       5   5e-04      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
```

```
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 4
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [164 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 done [0.09s].
## writing ... [32 rule(s)] done [0.01s].
## creating S4 object  ... done [0.01s].
```

```
plot(redwine_rules)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Top ten rules for red wine by confidence.

```
##      lhs                          rhs                   support confidence    coverage      lift co
## [1]  {bottled beer,
##       liquor,
##       soda}                    => {red/blush wine} 0.0008134215  0.6666667 0.001220132 34.691358
## [2]  {bottled beer,
##       liquor}                  => {red/blush wine} 0.0019318760  0.4130435 0.004677173 21.493559
```

```
## [3]  {liquor,
##       soda}                    => {red/blush wine} 0.0008134215  0.3809524 0.002135231 19.823633
## [4]  {bottled beer,
##       napkins,
##       soda}                    => {red/blush wine} 0.0005083884  0.3333333 0.001525165 17.345679
## [5]  {ham,
##       soda,
##       whole milk}              => {red/blush wine} 0.0005083884  0.2173913 0.002338587 11.312399
## [6]  {liquor}                  => {red/blush wine} 0.0021352313  0.1926606 0.011082867 10.025484
## [7]  {bottled water,
##       long life bakery product} => {red/blush wine} 0.0008134215  0.1904762 0.004270463  9.911817
## [8]  {fruit/vegetable juice,
##       shopping bags,
##       whole milk}              => {red/blush wine} 0.0006100661  0.1621622 0.003762074  8.438438
## [9]  {bottled water,
##       newspapers,
##       rolls/buns}              => {red/blush wine} 0.0006100661  0.1578947 0.003863752  8.216374
## [10] {bottled beer,
##       napkins}                 => {red/blush wine} 0.0008134215  0.1568627 0.005185562  8.162672
```

Liquor rules. Lowered the support slightly as liquor is a less common purchase than red wine.

```
liquor_rules <- apriori(data=groceries, parameter=list (supp=0.0001,conf = 0.08), appearance = list (rhs
```

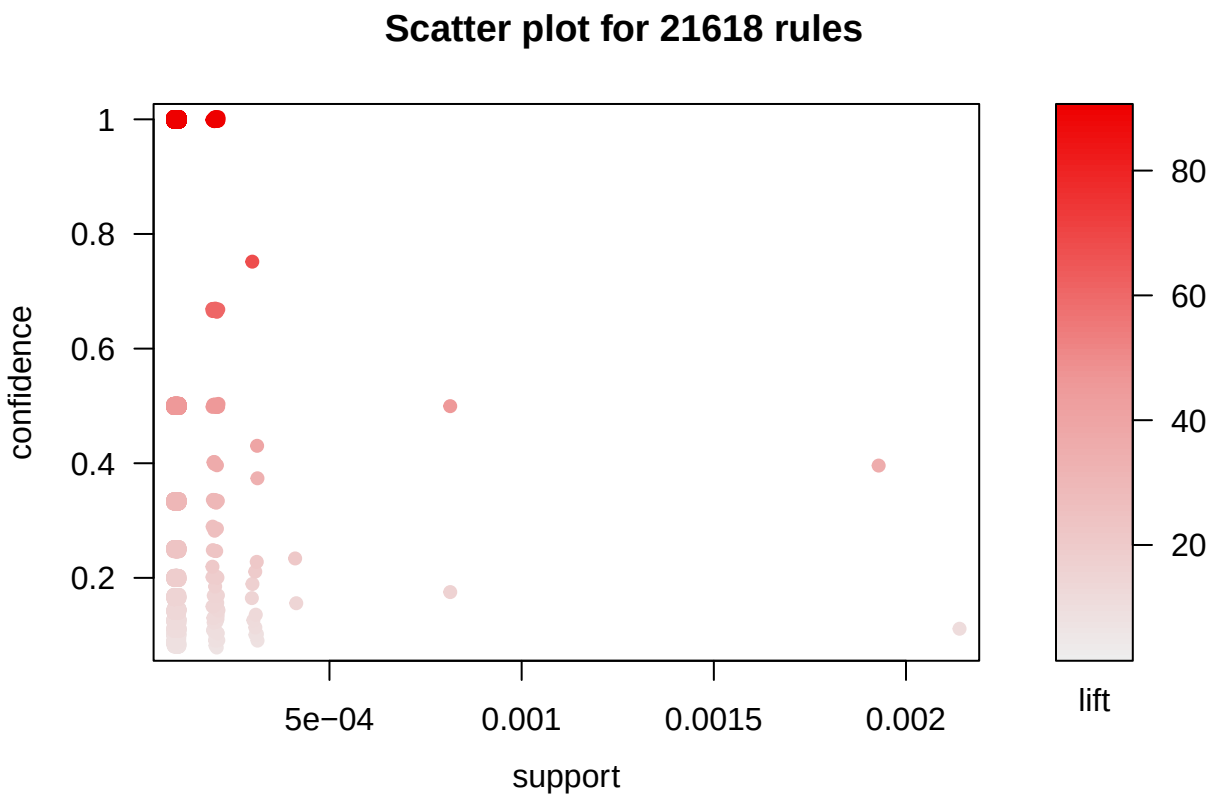
```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.08    0.1    1 none FALSE            TRUE       5   1e-04      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 0
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [169 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7
```

```
## Warning in apriori(data = groceries, parameter = list(supp = 1e-04, conf =
## 0.08), : Mining stopped (time limit reached). Only patterns up to a length of 7
## returned!
```

```
##  done [11.24s].
## writing ... [21618 rule(s)] done [1.80s].
## creating S4 object  ... done [1.25s].
```

```
plot(liquor_rules)
```

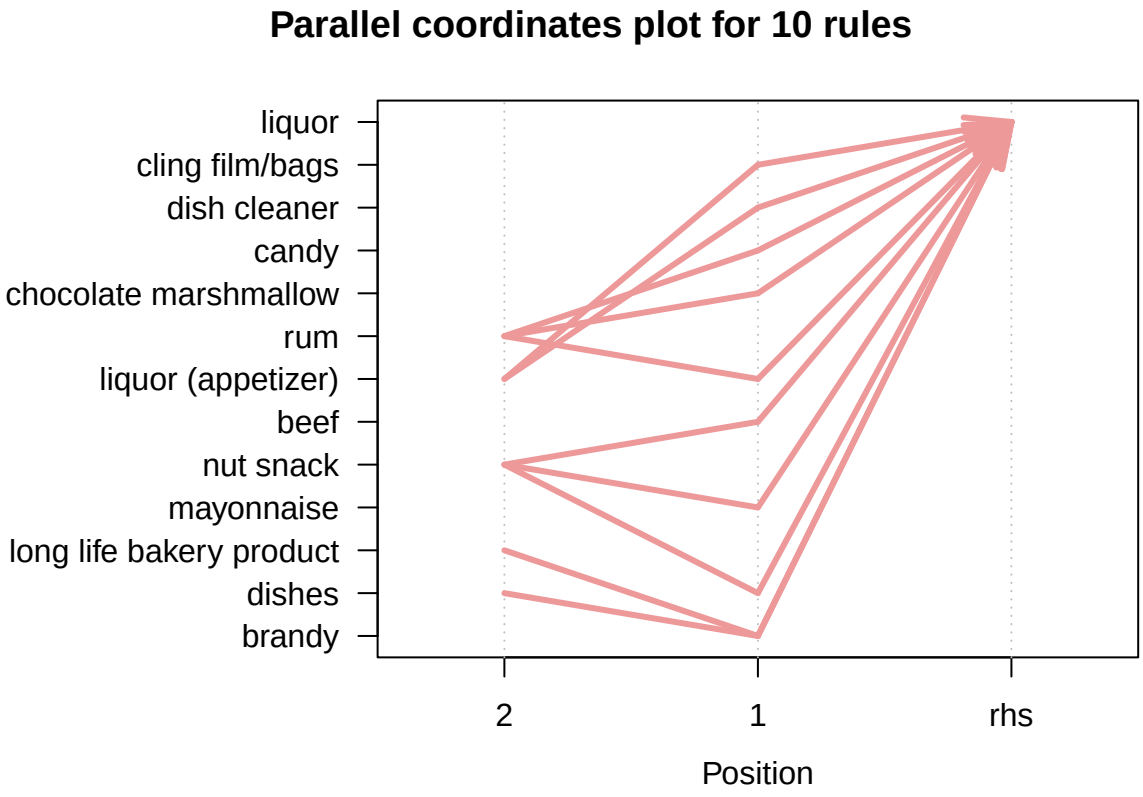
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```




```
##      lhs                                    rhs      support      confidence
## [1]  {brandy,dishes}                     => {liquor} 0.0001016777 1
## [2]  {brandy,long life bakery product}   => {liquor} 0.0001016777 1
## [3]  {mayonnaise,nut snack}              => {liquor} 0.0001016777 1
## [4]  {dishes,nut snack}                  => {liquor} 0.0001016777 1
## [5]  {beef,nut snack}                    => {liquor} 0.0001016777 1
## [6]  {liquor (appetizer),rum}            => {liquor} 0.0001016777 1
## [7]  {chocolate marshmallow,rum}         => {liquor} 0.0001016777 1
## [8]  {candy,rum}                         => {liquor} 0.0001016777 1
## [9]  {dish cleaner,liquor (appetizer)}   => {liquor} 0.0001016777 1
## [10] {cling film/bags,liquor (appetizer)} => {liquor} 0.0001016777 1
##      coverage     lift     count
## [1]  0.0001016777 90.22936 1
## [2]  0.0001016777 90.22936 1
## [3]  0.0001016777 90.22936 1
## [4]  0.0001016777 90.22936 1
## [5]  0.0001016777 90.22936 1
## [6]  0.0001016777 90.22936 1
## [7]  0.0001016777 90.22936 1
## [8]  0.0001016777 90.22936 1
```

```
## [9]  0.0001016777 90.22936 1
## [10] 0.0001016777 90.22936 1
```

```
subrulesliquor <- head(liquor_rules, n = 10, by = "confidence")
plot(subrulesliquor, method = "paracoord")
```



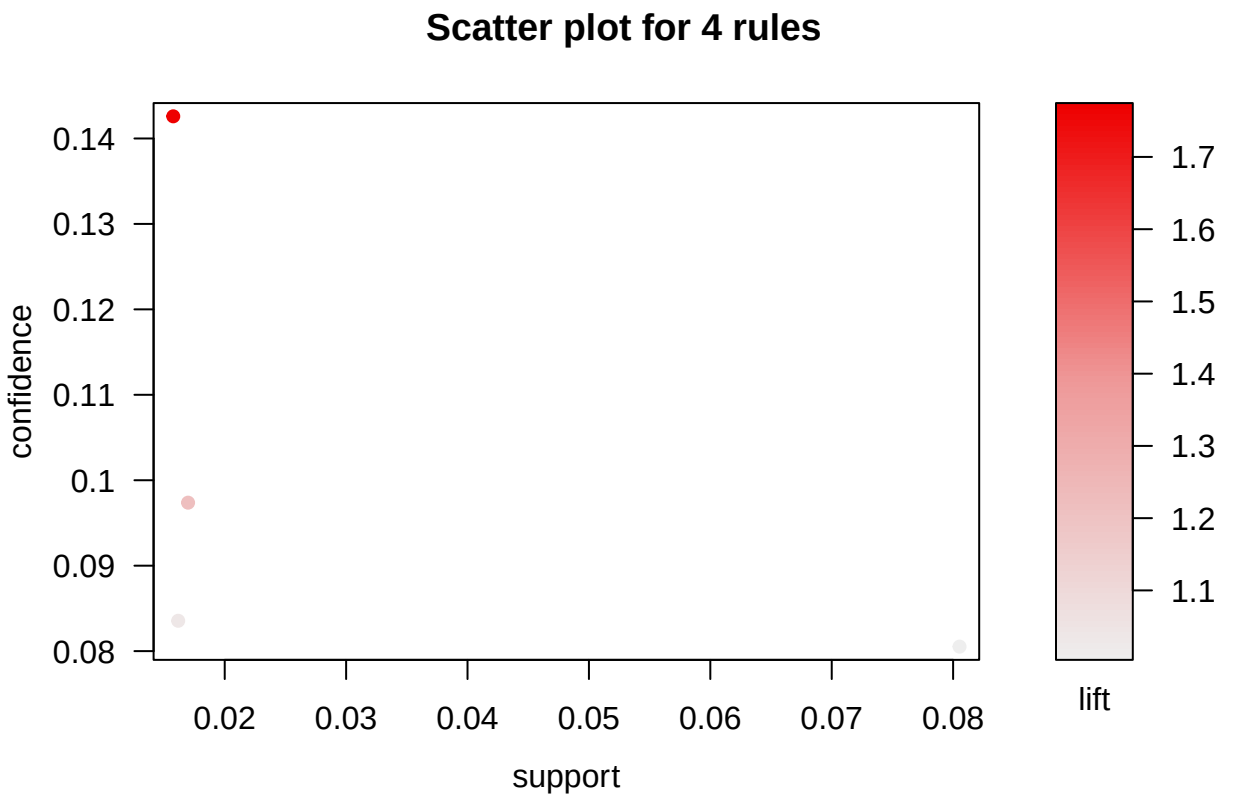
Beer rules.

```
beer_rules <- apriori(data=groceries, parameter=list (supp=0.01,conf = 0.08), appearance = list (rhs="b
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.08    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[1 item(s)] done [0.00s].
```

```
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [4 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
plot(beer_rules)
```



Bottled beer, red wine, and liquor all show similar trends and rules. People tend to purchase alcohol items together and along with these items purchases that are typical for parties such as dishes, candy, or cling/film bags.

These insights into consumer purchases can help the grocery store in a variety of ways such as promotions, coupons or item placement. Items, such as liquor that they want to sell more they can promote with other items that are more commonly purchased. They can also place party items, such as solo cups or soda mixers nearby to these items as they know people tend to purchase these items together. For the more common items like whole milk, it would be helpful to tie these into promotions with less commonly purchased items as they know many people will be coming into buy milk regardless.