Applied Predictive Analytics for Business Homework\_1

Team 3 - Priyanshu Porwal and Yash Saraiya

2/10/2021

library(dplyr)  
library(ggplot2)

## Question 1.

Print the top 10 items (item codes) returned by total amount refunded for the entire period of time covered by the transaction data set. Provide your list sorted by total amount refunded from highest to lowest. Note that transactions that correspond to returns have negative values for numpurchases and saleamount fields. Of course, positive values in these fields denote sales. Hence, there should be two columns of output, “item code” and “total refund.”

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 - Subsetting the dataframe for returns; Aggregating the return dataframe by item code

return<- trans[which(trans$numpurchases<0 & trans$saleamount<0),]  
#return  
return\_agg<- aggregate(return[,8],list(return$itemcode),sum,na.rm=TRUE)  
#return\_agg

Step 4 - Renaming the columns of the dataframe ; Sorting the return\_agg dataframe by total\_refund value ; Selecting the top 10 value from the return\_agg dataframe

names(return\_agg)[1]="item code"  
names(return\_agg)[2]="total refund"  
#return\_agg  
return\_agg<- return\_agg[order(return\_agg$`total refund`),]  
print(return\_agg[c(1:10),])

## item code total refund  
## 7602 36225560 -3595.9  
## 578 10435396815 -2550.0  
## 5038 21033438 -1379.9  
## 1943 1284487 -1350.0  
## 10756 8348528 -1239.8  
## 6417 26196669 -1200.0  
## 5470 2246335 -1198.0  
## 2797 1497953 -1195.0  
## 1678 12259121 -1185.0  
## 7798 37945275 -1095.0

## Question 2.

Do a similar analysis as in question 1, but now for customers (custcode) with the highest return rates (total number of items returned/total number of items sold) with a minimum of 2 returns.

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

category <- read.csv("category.csv") #variable category  
load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 - Subsetting the dataframe to calculate the return count for each custcode

return<- trans[which(trans$numpurchases<0 & trans$saleamount<0),]  
#return  
items\_returned\_per\_customer<- aggregate(return[,7],list(return$custcode),length)  
#items\_returned\_per\_customer  
names(items\_returned\_per\_customer)[1]="custcode"  
names(items\_returned\_per\_customer)[2]="return count"  
head(items\_returned\_per\_customer)

## custcode return count  
## 1 912074 11  
## 2 912079 2  
## 3 912138 5  
## 4 912214 2  
## 5 912238 5  
## 6 912279 2

Step 4 - Subsetting a dataframe to calculate the total numbers of item sold for each custcode

sold<- trans[which(trans$numpurchases>0 & trans$saleamount>0),]  
#sold  
items\_sold\_per\_customer<-aggregate(sold[,7],list(sold$custcode),length)  
#items\_sold\_per\_customer  
names(items\_sold\_per\_customer)[1]="custcode"  
names(items\_sold\_per\_customer)[2]="sale count"  
head(items\_sold\_per\_customer)

## custcode sale count  
## 1 912027 8  
## 2 912074 44  
## 3 912079 122  
## 4 912092 128  
## 5 912138 109  
## 6 912143 1

Step 5 - Creating a dataframe containing custcode, return count, sale count and return rate for custcode having minimum of 2 returns

x<-merge(items\_returned\_per\_customer,items\_sold\_per\_customer)  
x<- x[which(x$`return count`>1),]  
x$return\_rate<-x$`return count`/x$`sale count`  
head(x)

## custcode return count sale count return\_rate  
## 1 912074 11 44 0.25000000  
## 2 912079 2 122 0.01639344  
## 3 912138 5 109 0.04587156  
## 4 912214 2 3 0.66666667  
## 5 912238 5 34 0.14705882  
## 6 912279 2 112 0.01785714

Step 6 - Selecting top 10 results highest return rate

highest\_return\_rate<- x[order(-x$return\_rate),]  
print(highest\_return\_rate[1:10,])

## custcode return count sale count return\_rate  
## 821 961495 2 2 1.0000000  
## 851 963595 2 2 1.0000000  
## 354 933179 5 6 0.8333333  
## 43 914514 6 8 0.7500000  
## 138 920627 3 4 0.7500000  
## 546 945243 9 12 0.7500000  
## 753 957743 8 11 0.7272727  
## 64 915938 17 25 0.6800000  
## 4 912214 2 3 0.6666667  
## 206 924843 2 3 0.6666667

## Question 3.

Do customers that make a high number of returns have a longer customer relationship than customers with a low number of returns? What is the length of relationship for each group of customers? Use the mean number of returns to distinguish between high and low. Calculate the length of relationship as the difference (in days) between a customer’s very first transaction and very last transaction. When evaluating the difference, you may find it helpful to use ‘as.numeric’.

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

category <- read.csv("category.csv") #variable category  
load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 Grouping transaction data by customer to get the grouped view and summarizing the sale date difference as length of relationship. This is done by subtracting last transaction and first transaction. The length of relationship came out to be in days here.

updated\_trans <- trans %>% group\_by(custcode) %>% summarize(first\_transaction = min(saledate),   
 last\_transaction = max(saledate),   
 lenOfRelationship = last\_transaction - first\_transaction)  
print(head(updated\_trans))

## # A tibble: 6 x 4  
## custcode first\_transaction last\_transaction lenOfRelationship  
## <dbl> <date> <date> <drtn>   
## 1 912027 2000-08-13 2003-11-11 1185 days   
## 2 912074 2000-01-20 2004-10-24 1739 days   
## 3 912079 1999-07-22 2004-05-17 1761 days   
## 4 912092 1999-08-21 2006-08-12 2548 days   
## 5 912138 1999-08-14 2004-07-07 1789 days   
## 6 912143 1999-10-08 1999-10-08 0 days

Step 4 Merged the dataframe with length of relationship of customer with dataframe containing return count. We had it in last question as items\_returned\_per\_customer so merged both using custcode. The new dataframe has now both length of relationship and customer return count

merged\_trans = merge(updated\_trans, items\_returned\_per\_customer, by ="custcode")

Step 5 For analysis we converted lenOfRelationship to numeric using as.numeric

merged\_trans$lenOfRelationship <- as.numeric(merged\_trans$lenOfRelationship)

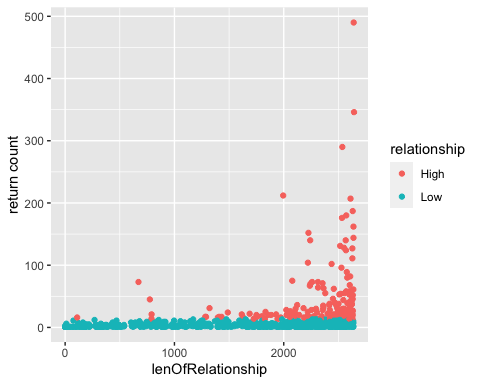
Step 6 Created a new column showing if relationship is longer or not using mean of return count as reference using ifelse

merged\_trans$relationship <- ifelse(merged\_trans$`return count` >= mean(merged\_trans$`return count`), "High", "Low")  
print(head(merged\_trans))

## custcode first\_transaction last\_transaction lenOfRelationship return count  
## 1 912074 2000-01-20 2004-10-24 1739 11  
## 2 912079 1999-07-22 2004-05-17 1761 2  
## 3 912138 1999-08-14 2004-07-07 1789 5  
## 4 912214 2005-08-03 2005-12-20 139 2  
## 5 912238 1999-07-20 2005-06-22 2164 5  
## 6 912279 2000-04-25 2006-06-23 2250 2  
## relationship  
## 1 Low  
## 2 Low  
## 3 Low  
## 4 Low  
## 5 Low  
## 6 Low

Step7 To answer the question “Do customers that make a high number of returns have a longer customer relationship than customers with a low number of returns?” We plotted a scatter-plot to show return count v/s length of relationship of customer, indicating each point as higher or lower relationship. We observed that it is indeed true that high number of returns have a longer customer relationship and customers with low number of returns have various lengths of relationship.

ggplot(merged\_trans, aes(x=lenOfRelationship, y=`return count`, color=relationship)) +  
 geom\_point()



## Question 4.

Provide total sales by income code and age code.Put the results in a matrix in which the columns denote the age and the rows denote income. Suppress (hide or otherwise ignore) codes that correspond to value of NA. If a specific income code and age code combination is NA, then you should report a zero, not NA.

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

category <- read.csv("category.csv") #variable category  
load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 - Merging customer and transaction tables to create a summary of transactions with respect to each customer for further processing. Also, subsetting data from the resultant dataframe and keeping only those columns required for processing.

merged <- merge(customer, trans)  
merged <- merged[ , -c(4:12)]

Step 4 - Based on which rows have both income code & age code showing as NA, assigning a value of 0 to these rows. Similarly, for rows with only either of the codes missing, we remove these rows by setting the sales amount as NA before finally subsetting using sales amount where value is numeric & not NA.

merged[which(is.na(merged$inccode) & is.na(merged$agecode)), 4] <- 0  
merged[which(is.na(merged$inccode) | is.na(merged$agecode)), 4] <- NA  
merged <- merged[-which(is.na(merged$saleamount)), ]

Step 5 - Aggregating merged dataframe with both age code and income code by using the sum function to create a new dataframe for representing age and income-based sales. Also, adding the missing 3 rows for age group 1 to the sales dataframe by using rbind() function. Lastly, renaming the three columns to appropriate column names using names() command.

sales <- aggregate(merged[,c(4)], by=list(merged$agecode, merged$inccode), sum)  
row1 <- c(1,1,0)  
row2 <- c(1,3,0)  
row3 <- c(1,5,0)  
sales <- rbind(row1, row2, row3, sales)  
names(sales)[1] <- "agecode"  
names(sales)[2] <- "inccode"  
names(sales)[3] <- "totalsales"

Step 6 - After adding the missing rows, we see that the sorting order has changed. To insert into a matrix, the dataframe needs to be sorted. So the below mentioned order() command is used to sort using income code, so that the matrix insertion can happen by default.

sales <- sales[order(sales$inccode), ]

Step 7 - Finally, inserting the dataframe into the result matrix with 6 rows for age code and 5 columns of income code. The matrix is filled with total sales amount for each of these groups. And then lastly, print the result matrix.

result <- matrix(sales$totalsales, nrow = 6, ncol = 5)  
print(result)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 0.0 157.00 0.00 506.00 0.00  
## [2,] 244478.5 48223.64 9393.72 7393.45 23093.91  
## [3,] 222451.2 98819.04 80408.73 87435.67 93580.72  
## [4,] 267604.6 233252.05 136447.60 93405.52 239148.39  
## [5,] 228688.8 254469.20 180584.79 103377.41 211382.11  
## [6,] 314238.0 123152.07 86665.10 85907.62 97861.88

## Question 5.

Determine at an aggregate level (total$refunded/total $sales), the return rate by product category. Be sure to identify the product categories by name and sort by return rate from highest to lowest. Do not specify sub-categories.

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

category <- read.csv("category.csv") #variable category  
load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 - Calculate two dataframes that gives list of all refunds and all sales.

refunds <- trans[which(trans$numpurchases < 0), ]  
sales <- trans[which(trans$numpurchases > 0 ), ]

Step 4 - Create two new combined dataframes now, that combines refunds & sales with category dataframe.

merged\_sales <- merge(sales, category)  
merged\_refunds <- merge(refunds, category)

Step 5 - With this new merged dataframes of sales and refunds, now aggregate using category code. Rename columns with logical names.

agg\_sales <- aggregate(merged\_sales[, c(8)], by=list(merged\_sales$catcode), sum)  
names(agg\_sales)[1] <- "catcode"  
names(agg\_sales)[2] <- "totalsales"  
agg\_refunds <- aggregate(merged\_refunds[, c(8)], by=list(merged\_refunds$catcode), sum)  
names(agg\_refunds)[1] <- "catcode"  
names(agg\_refunds)[2] <- "totalrefunds"

Step 6 - Now merge the two dataframes using category codes, followed by another merge with category dataframes to subset category names. Also subset the resultant dataframe with the required columns.

agg\_sales\_refunds <- merge(agg\_sales, agg\_refunds)  
result <- merge(agg\_sales\_refunds, category)  
result <- result[, c(2,3,6)]

Step 7 - Now, create the returns rate. Formula for return rate is given below: Return Rate = Total Dollars Refunded / Total Dollars in Sales. Note that absolute value is taken to compute the ratio (Return Rate), which is why abs() function is used to determine the absolute value.

result$returnrate <- abs(result$totalrefunds) / result$totalsales

Step 8 - Sort the table with return rate and remove duplicates.Lastly, print the dataframe to show product categories ordered from highest to lowest return rate.

result <- result[order(-result$returnrate), ]  
result <- result[!duplicated(result), ]  
print(result)

## totalsales totalrefunds catname returnrate  
## 1 215448.20 -47650.70 Accessories 0.22117010  
## 12 114484.02 -24029.24 Apparel 0.20989165  
## 86 139498.29 -27958.65 Leased 0.20042289  
## 87 139498.29 -27958.65 Jewelry 0.20042289  
## 38 2351566.35 -424298.63 Apparel Women 0.18043234  
## 104 725135.51 -130756.12 Shoes Women 0.18031957  
## 18 144893.76 -21482.47 Apparel Children 0.14826360  
## 100 127589.95 -16953.74 Shoes Men 0.13287677  
## 97 43112.90 -5195.84 Shoes Children 0.12051706  
## 28 311256.83 -31578.42 Apparel Men 0.10145454  
## 6 86982.92 -7947.57 Accessories Men 0.09136932  
## 7 20008.68 -1415.96 Accessories Women 0.07076729  
## 93 357838.36 -21648.24 Service 0.06049726  
## 79 117107.25 -6774.37 Cosmetics 0.05784757  
## 83 22109.51 -1105.07 Home Goods 0.04998166

## Question 6.

In the transaction data set, for all the returns, determine the time-to-return, measured as the number of elapsed days between the sale of an item and its return (return date - purchase date). If for a given return, you identify multiple purchases to which it may belong, choose the purchase date closest to the return date.

Note that it is possible to have zero day returns (items returned on the same day of purchase). Determine the fraction of the total number of returns that are completed in 0 days, 1 day, 2 days, ., up to 25 days, as well as the remaining fraction of returns that are completed after 25 days.

Step 1 - Initializing the current file path & clearing variables from the environment of execution.

library(rstudioapi)  
current\_path=rstudioapi::getActiveDocumentContext()$path  
setwd(dirname(current\_path))

Step 2 - Loading the required input files for processing

category <- read.csv("category.csv") #variable category  
load("transaction.rdata") #variable trans  
load("customer.rdata") #variable customer  
head(trans)

## custcode itemcode channel subcatcode saledate price numpurchases saleamount  
## 2 912027 1071714 2 300 2000-08-13 14.5 1 14.5  
## 5 912027 1170201 2 300 2000-08-13 10.5 1 10.5  
## 8 912027 748263 2 125 2000-08-13 10.0 1 10.0  
## 1 912027 10131552 2 300 2003-11-11 10.0 1 10.0  
## 3 912027 11669952 2 300 2003-11-11 16.5 1 16.5  
## 4 912027 11670498 2 300 2003-11-11 16.5 1 16.5

Step 3 - Calculate two dataframes that gives list of all refunds and all sales.

refunds <- trans[which(trans$numpurchases < 0 & trans$saleamount < 0), ]  
sales <- trans[which(trans$numpurchases > 0 & trans$saleamount > 0), ]

Step 4 - Sorting the dataframes using itemcode and custcode to make it easier to perform code analysis, and to test code. Also, rename the new saledate of refund dataframe to refunddate

refunds <- refunds[order(refunds$itemcode, refunds$custcode),]  
sales <- sales[order(sales$itemcode, sales$custcode),]  
names(refunds)[5] <- "refunddate"

Step 5 - Unselect columns that are not essential for further analysis.

refunds <- refunds[,c(1,2,5)]  
sales <- sales[,c(1,2,5)]

Step 6 - Merge the two dataframes of sales and refunds to form a single table. Merge using both custcode and itemcode, and ensure to only keep records that match on both dataframes to analyze time-to-return values.

merged <- merge(sales, refunds, by=c("custcode","itemcode"), all = FALSE)

Step 7 - Compute new column elapseddays by subtracting refunddate and saledate.

merged$elapseddays <- as.Date(merged$refunddate) - as.Date(merged$saledate)

Step 8 - Data cleaning; there are a few rows with refund date showing as before the product’s sale date. This is an inconsistency that is unlikely to appear. Thus, eliminate these records by assuming they are wrongly entered data.

merged <- (merged[!merged$elapseddays < 0, ])

Step 9 - Loop through elapsed days to compute new elapsed days by only considering the smallest elapsed days.

for (row in 2:nrow(merged)) {  
 cust <- merged[row, "custcode"]  
 item <- merged[row, "itemcode"]  
   
 #After sorting, when custcode and itemcode match consistently, is when they are to be changed to their  
 #smallest elapsed days, i.e. selecting the smallest difference between purchase date & return date.  
 if(cust == merged[row-1, "custcode"] && item == merged[row-1, "itemcode"]) {  
 elapsed = min(merged[row, "elapseddays"], merged[row-1, "elapseddays"])  
 merged[row, "elapseddays"] <- elapsed  
 merged[row-1, "elapseddays"] <- elapsed  
 }  
}

Step 10 - Remove total duplicates and sort the dataframe by elapsed days finally

merged <- merged[!duplicated(merged[, c(1,2,5)]), ]  
merged <- merged[order(merged$elapseddays), ]

Step 11 - Before running the main fraction calculation loop, initialize an empty dataframe to store these values, along with the totalrows present in the merged dataframe.

totalrows <- nrow(merged)  
returnsfraction <- data.frame(days = numeric(0), fraction = numeric(0))

Step 12 - Loop through the merged dataframe and calculate the ratio of number of 0 days, 1 days, .. and so on. Store these values in a new dataframe called returnsfraction using rbind() function.

for (num in 0:max(merged$elapseddays)){  
 count <- nrow(merged[merged$elapseddays == num, ])  
 if(!is.null(count)){  
 if(count > 0){  
 ratio <- count / totalrows  
 returnsfraction <- rbind(returnsfraction, c(num, ratio))  
 }  
 }  
}

Step 13 - Rename columns to display logical names for the fractions dataframe before running a print command.

colnames(returnsfraction)[1] <- "fraction\_of\_X\_days"  
colnames(returnsfraction)[2] <- "fraction"  
print(returnsfraction)

## fraction\_of\_X\_days fraction  
## 1 0 1.337404e-01  
## 2 1 6.287090e-02  
## 3 2 5.237809e-02  
## 4 3 4.592758e-02  
## 5 4 3.311258e-02  
## 6 5 3.362862e-02  
## 7 6 3.629483e-02  
## 8 7 4.824976e-02  
## 9 8 2.907027e-02  
## 10 9 2.141567e-02  
## 11 10 2.072762e-02  
## 12 11 1.969554e-02  
## 13 12 1.763138e-02  
## 14 13 2.029758e-02  
## 15 14 2.373785e-02  
## 16 15 1.926550e-02  
## 17 16 1.634127e-02  
## 18 17 1.315903e-02  
## 19 18 1.272899e-02  
## 20 19 1.272899e-02  
## 21 20 1.169691e-02  
## 22 21 1.608325e-02  
## 23 22 1.040681e-02  
## 24 23 8.342651e-03  
## 25 24 9.718758e-03  
## 26 25 8.600671e-03  
## 27 26 9.460738e-03  
## 28 27 1.023480e-02  
## 29 28 1.075084e-02  
## 30 29 8.428657e-03  
## 31 30 6.794530e-03  
## 32 31 7.654597e-03  
## 33 32 6.536510e-03  
## 34 33 5.418423e-03  
## 35 34 7.482584e-03  
## 36 35 6.192483e-03  
## 37 36 7.396577e-03  
## 38 37 4.988389e-03  
## 39 38 4.300335e-03  
## 40 39 3.354262e-03  
## 41 40 3.096242e-03  
## 42 41 3.612282e-03  
## 43 42 5.418423e-03  
## 44 43 4.730369e-03  
## 45 44 4.300335e-03  
## 46 45 3.182248e-03  
## 47 46 3.354262e-03  
## 48 47 2.838221e-03  
## 49 48 3.182248e-03  
## 50 49 5.332416e-03  
## 51 50 1.892148e-03  
## 52 51 2.666208e-03  
## 53 52 2.924228e-03  
## 54 53 2.322181e-03  
## 55 54 1.118087e-03  
## 56 55 3.010235e-03  
## 57 56 3.440268e-03  
## 58 57 3.096242e-03  
## 59 58 1.204094e-03  
## 60 59 1.376107e-03  
## 61 60 1.634127e-03  
## 62 61 1.290101e-03  
## 63 62 2.752215e-03  
## 64 63 2.064161e-03  
## 65 64 2.408188e-03  
## 66 65 1.806141e-03  
## 67 66 1.634127e-03  
## 68 67 1.376107e-03  
## 69 68 1.806141e-03  
## 70 69 1.032081e-03  
## 71 70 1.720134e-03  
## 72 71 1.634127e-03  
## 73 72 1.204094e-03  
## 74 73 1.978154e-03  
## 75 74 1.118087e-03  
## 76 75 1.892148e-03  
## 77 76 2.666208e-03  
## 78 77 1.548121e-03  
## 79 78 1.032081e-03  
## 80 79 1.376107e-03  
## 81 80 1.720134e-03  
## 82 81 6.020470e-04  
## 83 82 7.740604e-04  
## 84 83 1.376107e-03  
## 85 84 1.290101e-03  
## 86 85 1.032081e-03  
## 87 86 1.032081e-03  
## 88 87 7.740604e-04  
## 89 88 4.300335e-04  
## 90 89 6.020470e-04  
## 91 90 1.032081e-03  
## 92 91 1.204094e-03  
## 93 92 6.020470e-04  
## 94 93 6.020470e-04  
## 95 94 8.600671e-04  
## 96 95 1.204094e-03  
## 97 96 1.720134e-04  
## 98 97 9.460738e-04  
## 99 98 1.118087e-03  
## 100 99 6.880537e-04  
## 101 100 7.740604e-04  
## 102 101 1.032081e-03  
## 103 102 3.440268e-04  
## 104 103 8.600671e-04  
## 105 104 7.740604e-04  
## 106 105 1.290101e-03  
## 107 106 3.440268e-04  
## 108 107 1.720134e-04  
## 109 109 6.020470e-04  
## 110 110 7.740604e-04  
## 111 111 1.290101e-03  
## 112 112 4.300335e-04  
## 113 113 6.020470e-04  
## 114 114 5.160403e-04  
## 115 115 5.160403e-04  
## 116 116 4.300335e-04  
## 117 117 1.720134e-04  
## 118 118 2.580201e-04  
## 119 119 9.460738e-04  
## 120 120 6.020470e-04  
## 121 121 1.290101e-03  
## 122 122 4.300335e-04  
## 123 123 3.440268e-04  
## 124 125 3.440268e-04  
## 125 126 8.600671e-05  
## 126 127 5.160403e-04  
## 127 128 3.440268e-04  
## 128 129 1.720134e-04  
## 129 130 3.440268e-04  
## 130 131 4.300335e-04  
## 131 133 5.160403e-04  
## 132 134 6.020470e-04  
## 133 135 2.580201e-04  
## 134 136 5.160403e-04  
## 135 137 5.160403e-04  
## 136 138 6.020470e-04  
## 137 139 3.440268e-04  
## 138 140 1.720134e-04  
## 139 142 2.580201e-04  
## 140 143 1.720134e-04  
## 141 144 2.580201e-04  
## 142 145 1.720134e-04  
## 143 146 1.720134e-04  
## 144 147 4.300335e-04  
## 145 148 2.580201e-04  
## 146 150 8.600671e-05  
## 147 151 4.300335e-04  
## 148 152 8.600671e-05  
## 149 153 8.600671e-05  
## 150 154 6.020470e-04  
## 151 155 8.600671e-05  
## 152 156 1.720134e-04  
## 153 157 8.600671e-05  
## 154 158 4.300335e-04  
## 155 159 8.600671e-05  
## 156 160 4.300335e-04  
## 157 161 1.720134e-04  
## 158 163 1.720134e-04  
## 159 164 2.580201e-04  
## 160 165 1.720134e-04  
## 161 167 1.720134e-04  
## 162 168 5.160403e-04  
## 163 169 1.720134e-04  
## 164 170 8.600671e-05  
## 165 171 2.580201e-04  
## 166 172 4.300335e-04  
## 167 173 1.720134e-04  
## 168 174 8.600671e-05  
## 169 175 1.720134e-04  
## 170 177 3.440268e-04  
## 171 180 8.600671e-05  
## 172 181 2.580201e-04  
## 173 182 6.880537e-04  
## 174 186 3.440268e-04  
## 175 187 2.580201e-04  
## 176 188 1.720134e-04  
## 177 189 2.580201e-04  
## 178 191 8.600671e-05  
## 179 192 1.720134e-04  
## 180 194 1.720134e-04  
## 181 195 1.720134e-04  
## 182 196 3.440268e-04  
## 183 197 8.600671e-05  
## 184 198 1.720134e-04  
## 185 199 1.720134e-04  
## 186 200 3.440268e-04  
## 187 201 8.600671e-05  
## 188 203 1.720134e-04  
## 189 204 6.020470e-04  
## 190 205 8.600671e-05  
## 191 206 2.580201e-04  
## 192 210 4.300335e-04  
## 193 212 8.600671e-05  
## 194 214 3.440268e-04  
## 195 215 8.600671e-05  
## 196 216 1.720134e-04  
## 197 217 8.600671e-05  
## 198 218 2.580201e-04  
## 199 219 1.720134e-04  
## 200 220 1.720134e-04  
## 201 221 8.600671e-05  
## 202 222 8.600671e-05  
## 203 223 8.600671e-05  
## 204 224 8.600671e-05  
## 205 228 8.600671e-05  
## 206 229 8.600671e-05  
## 207 231 1.720134e-04  
## 208 232 8.600671e-05  
## 209 236 8.600671e-05  
## 210 238 8.600671e-05  
## 211 243 1.720134e-04  
## 212 244 8.600671e-05  
## 213 245 4.300335e-04  
## 214 246 8.600671e-05  
## 215 247 8.600671e-05  
## 216 249 1.720134e-04  
## 217 252 8.600671e-05  
## 218 253 8.600671e-05  
## 219 256 8.600671e-05  
## 220 258 8.600671e-05  
## 221 259 2.580201e-04  
## 222 260 1.720134e-04  
## 223 261 1.720134e-04  
## 224 262 8.600671e-05  
## 225 263 8.600671e-05  
## 226 264 8.600671e-05  
## 227 268 2.580201e-04  
## 228 273 8.600671e-05  
## 229 277 1.720134e-04  
## 230 280 8.600671e-05  
## 231 283 8.600671e-05  
## 232 286 1.720134e-04  
## 233 287 8.600671e-05  
## 234 290 8.600671e-05  
## 235 293 8.600671e-05  
## 236 294 8.600671e-05  
## 237 295 8.600671e-05  
## 238 296 8.600671e-05  
## 239 304 8.600671e-05  
## 240 308 2.580201e-04  
## 241 309 8.600671e-05  
## 242 318 8.600671e-05  
## 243 322 8.600671e-05  
## 244 327 8.600671e-05  
## 245 329 8.600671e-05  
## 246 332 8.600671e-05  
## 247 334 8.600671e-05  
## 248 335 8.600671e-05  
## 249 336 8.600671e-05  
## 250 343 1.720134e-04  
## 251 344 8.600671e-05  
## 252 349 8.600671e-05  
## 253 352 8.600671e-05  
## 254 354 8.600671e-05  
## 255 357 8.600671e-05  
## 256 359 1.720134e-04  
## 257 361 8.600671e-05  
## 258 363 4.300335e-04  
## 259 373 8.600671e-05  
## 260 384 8.600671e-05  
## 261 388 8.600671e-05  
## 262 389 8.600671e-05  
## 263 393 2.580201e-04  
## 264 395 8.600671e-05  
## 265 399 8.600671e-05  
## 266 404 8.600671e-05  
## 267 405 1.720134e-04  
## 268 406 8.600671e-05  
## 269 414 8.600671e-05  
## 270 416 8.600671e-05  
## 271 418 8.600671e-05  
## 272 423 8.600671e-05  
## 273 424 8.600671e-05  
## 274 427 8.600671e-05  
## 275 439 8.600671e-05  
## 276 444 8.600671e-05  
## 277 446 8.600671e-05  
## 278 448 1.720134e-04  
## 279 449 8.600671e-05  
## 280 450 8.600671e-05  
## 281 456 2.580201e-04  
## 282 467 8.600671e-05  
## 283 470 8.600671e-05  
## 284 475 8.600671e-05  
## 285 477 8.600671e-05  
## 286 480 8.600671e-05  
## 287 487 8.600671e-05  
## 288 488 8.600671e-05  
## 289 498 8.600671e-05  
## 290 502 1.720134e-04  
## 291 510 1.720134e-04  
## 292 512 8.600671e-05  
## 293 516 8.600671e-05  
## 294 526 8.600671e-05  
## 295 532 8.600671e-05  
## 296 537 4.300335e-04  
## 297 552 1.720134e-04  
## 298 569 8.600671e-05  
## 299 571 8.600671e-05  
## 300 583 8.600671e-05  
## 301 625 8.600671e-05  
## 302 637 1.720134e-04  
## 303 660 8.600671e-05  
## 304 661 8.600671e-05  
## 305 686 8.600671e-05  
## 306 692 8.600671e-05  
## 307 759 8.600671e-05  
## 308 824 8.600671e-05  
## 309 833 8.600671e-05  
## 310 922 8.600671e-05  
## 311 924 8.600671e-05  
## 312 945 8.600671e-05  
## 313 963 8.600671e-05  
## 314 1352 8.600671e-05  
## 315 1649 1.720134e-04  
## 316 1760 1.032081e-03