トランザクションデータ処理に使用した S 言語プログラム

【使用したトランザクション型ショッパー行動データ】

ショッパー行動データは変数 main として格納.

main の各列に格納された情報は以下のとおりである.

[1] ショッパーID
[2] 商品名
[3] ブランド名
[4] 表示価格
[5] 接触時間
[6] 棚前滞在時間
[7] 商品接触までの時間

【アリゴリズムの特徴】

トランザクションデータであるため，1ショッパーのレコード数が異なる.

したがってすべての処理において，preID として一つ上のレコードのショッパーID を格納しておき，

現在処理しているレコードの ID と preID を比較することで，

処理するショッパーが一つ上のレコードのショッパーと同じか，

新たな次のショッパーかを識別することで処理している.

【前処理：価格データの設定】

```
price_pre<-main[,4]            #価格帯の元データを price_pre に格納
price<-replace(price_pre,price_pre<690,"low")                    #低価格帯を price に記録
price<-replace(price,price_pre>690&price_pre<1260,"middle")      #中価格帯を price に記録
price<-replace(price,price_pre>=1260,"high")                     #高価格帯を price に記録
```

【Data1:接触ブランド数】

```
ID<-main[,1]        #ショッパーID を ID に格納
preID<-ID[1]        #1つ上のレコードのショッパーID を示す preID を1番目のショッパーにセット
brand<-main[,3]     #ブランド名を brand に格納
brand_count<-NULL           #接触ブランド数リストを初期化
x<-1
b_list<-NULL        #処理用ブランドリスト初期化
```

```
for(i in 1:length(brand)){
if(ID[i]==preID){              #処理 ID が1つ上のレコードのショッパーID と同じ
                               #＝一つ上のレコードと同じショッパー＝処理対象ショッパーが同じ
    b_list[brand[i]]<-1        #接触したブランドにフラグを立てる
  }else{                       #処理 ID が前ループの ID と異なる＝処理対象ショッパーが変更
    preID<-ID[i]               #preID を現在のショッパーID に再設定
    count[x]<-sum(na.omit(b_list))
                               #処理用ブランドリストに立ったフラグの数を接触ブランド数リストへ格納
    x<-x+1                     #接触ブランド数リストを1つ進める
    b_list<-NULL               #処理用ブランドリストに立ったフラグを全部初期化
    b_list[brand[i]]<-1        #現在処理しているレコードのブランドにフラグを立てる
  }
}
```

【Data2:価格帯のばらつき】

```
#価格帯別の商品数／接触時間リストの作成
time<-main[5]                  #商品接触データを time へ格納
preID<-ID[1]
dat<-c(0,0,0,0,0,0)            #処理用リストを初期化
                               #dat[低価格帯商品数，中価格帯商品数，高価格帯商品数，
                               #     低価格帯接触時間，中価格帯接触時間，高価格帯接触時間]
list<-dat                      #結果格納用リスト
for(i in 1:nrow(main)){
 if(preID==ID[i]){
   dat[price[i]]<-dat[price[i]]+1                #商品の価格帯に応じて商品数を加算
   dat[price[i]+3]<-dat[price[i]+3]+time[i]      #商品の価格帯に応じて接触時間を加算
 }else{
   preID<-ID[i]
   list<-rbind(list,dat)       #リストをつなげる
   dat<-c(0,0,0,0,0,0)         #処理用リストを初期化
   dat[price[i]]<-dat[price[i]]+1                #商品の価格帯に応じて商品数を加算
    dat[price[i]+3]<-dat[price[i]+3]+time[i]     #商品の価格帯に応じて接触時間を加算
 }
}
list<-rbind(list,dat)          #商品数／接触時間リストの完成
```

```
#作成したリストを用いて，価格帯のばらつきを計算
price_sd <- apply(list[,1:3],1,sd)

【Data3：最も接触した価格帯】
max_price<-apply(list[,1:3],1,max)     #最も接触した価格帯の商品数
max_time<-apply(list[,4:6],1,max)      #最も長時間接触した価格帯の接触時間

m_list<-cbind(list,max_price,max_time)#最も接触した価格帯リスト作成処理用リスト

func<-function(list){                                  #最も接触した価格帯計算用関数
          list2<-matrix(c(1,0,0,2,0,0,3,0,0),ncol=3)       #処理用行列
          for(i in 1:3){
           if(list[i]==list[7]) list2[2,i]<-1                #最も接触した商品の価格帯格納
           if(list[i+3]==list[8]) list2[3,i]<-1              #最も長時間接触した価格帯格納
          }
          if(sum(list2[2,])==1))        return(list2[1,list2[2,]==1]) #商品数から決定
          else if(sum(list2[3,])==1)    return(list2[1,list2[3,]==1]) #接触時間から決定
          else                          return(round(runif(1)))       #ランダムに決定
}

max_result<-apply(m_list,1,func)       #最も接触した価格帯リストを func を用いて作成
```