|  |  |  |  |
| --- | --- | --- | --- |
| 数据清理 | SQL | Pandas | R |
| 删除重复行 | 需要使用到row\_number（）或者是join+delete | 直接df.drop\_duplicates() | data <- data[!duplicated(data), ]或者是df\_unique <- unique(df)  再或者是df\_unique <- df %>% distinct()，都可以的 |
| 处理姓名，字符清理（首尾空格+首尾字符串） | 前后的空格需要用到trim(),前后的的一些字符串例如-/.等等需要用到regexp\_replace()。比如说REGEXP\_REPLACE(Last\_Name, '^[/\_.]+|[/\_.]+$', '')，因为这里主要会用到^：表示匹配字符串的开头，$：表示匹配字符串的结尾 | Pandas中去除前后的空格是str.strip()。str.replace(r"[字符]", "", regex=True)（去掉中间字符） | str\_trim即是清除首尾的空格 |
| 中间字符清理  【全局替换】 | 同样还是regexp\_replace（） | 同样还是str.replace，记得regex=True，启动正则表达式 | str\_replace\_all |
| 拆分列（比如拆分地址之类的） | Sql需要使用使用substring\_index，可能会涉及到多层嵌套，里面的count可能会是1，2，也有可能是是-1，表示从左往右，或者是从右往左。 | 直接 df['Address'].str.split(',', expand=True)，清晰简单 | Tidyr里面的separate()  separate(data, col, into, sep = " ", remove = TRUE, extra = "warn", fill = "warn") |
| 统一yes\no | 使用case when 进行update | 依旧是使用str.replace | Dplyr里面的case when,其实相当于sql里面的case when |
| 处理没有信息的值（无论是真正的null值还是以字符串储存的，都需要统一起来） | SQL 需要 CASE WHEN 'NA' THEN NULL 来 UPDATE,此时这里的null就是sql 里面真正的null的 | 直接 df.replace([...], np.nan, inplace=True)，然后 df.fillna('') 统一，这里的np.nan就和前面的sql里面的null是一样的，表示缺失值 | data[data %in% c("Na", "None", "N/a", "NA", " ")] <- NA |
| 删除没有用的列 | ALTER TABLE Customer\_Call\_List\_2  DROP COLUMN Not\_Useful\_Column; | df = df.drop(columns = "Not\_Useful\_Column") | 直接使用索引data <- data[c(-8,-9,-10)]；或者是df$B <- NULL；再或者是df\_new <- subset(df, select = -B) |