Lab 02: Bobabase SOLUTIONS

PSTAT 100, Summer Session A 2025 with Ethan P. Marzban

MEMBER 1 (NetID 1) MEMBER 2 (NetID 2)
MEMBER 3 (NetID 3)

June 26, 2025

Required Packages

```
library(ottr)        # for checking test cases (i.e. autograding)
library(pander)      # for nicer-looking formatting of dataframe outputs
library(tidyverse)   # for graphs, data wrangling, etc.
```

Logistical Details

Logistical Details

- This lab is due by **11:59pm on June 27, 2025**.
- Collaboration is allowed, and encouraged!
  - If you work in groups, list ALL of your group members' names and NetIDs (not Perm Numbers) in the appropriate spaces in the YAML header above.
  - Please delete any "MEMBER X" lines in the YAML header that are not needed.
  - No more than 3 people in a group, please.
- Ensure your Lab properly renders to a `.pdf`; non-`.pdf` submissions will not be graded and will receive a score of 0.
- Ensure all test cases pass (test cases that have passed will display a message stating `"All tests passed!"`)

Lab Overview and Objectives

Welcome to another PSTAT 100 Lab! In this lab, we will cover the following:

- Database basics

- Joining (merging) multiple data frames
- Pivoting (aka widening) data frames

Part I: Introduction to Databases

As you might imagine, a single file isn't always enough to capture all of the data necessary for a particular analysis. Sometimes, we need to consider a **database**: for the purposes of this class, we can think of a database as simply being a collection of dataframes, all related in some way.

In this week's lab we'll don the persona of a financial analyst working for *GauchoBubble*, a new up-and-coming boba shop. We are particularly interested in examining the revenue generated by sales on a particular day at *GauchoBubble*. Available to us is a database called `GauchoBubble`, which contains three files (each containing a single dataframe): `orders.csv`, `inventory.csv`, and `customers.csv`. A brief description of these files (along with associated data dictionaries) is provided below.

`orders.csv`: contains information on the orders placed during a particular day at *GauchoBubble*. Variables included:

- `ORDER_NO`: an identifier for each order placed (orders containing multiple items are split across multiple rows, with only one order in each row)
- `ITEM_NO`: a unique identifier of each item sold at *GauchoBubble*
- `CUST_ID`: a unique customer ID for customers included in *GauchoBubble*'s rewards program. If a particular customer is not in the rewards program, a value of `NA` is recorded.

`inventory.csv`: contains information on the items sold at *GauchoBubble*. Variables included:

- `ITEM_NO`: a unique identifier of each item sold at *GauchoBubble*
- `DESCRIP`: a verbal description of each item sold at *GauchoBubble*
- `PPU`: the price-per-unit of each item sold at *GauchoBubble*

`customers.csv`: contains information on customers enrolled in the *GauchoBubble* rewards program. Variables included:

- `CUST_ID`: a unique identifier for each customer
- `FIRST_NAME`: customer's first name
- `LAST_NAME`: customer's last name
- `STATE`: customer's US state of residence
- `BIRTH_MONTH`: customer's birth month

Here are the first few rows of the three data frames:

ORDERS:

ORDER_NO	ITEM_ID	CUST_ID
1	T00	C25
1	T02	C25
1	T02	C25
2	U00	NA
2	G03	NA
3	T01	NA

INVENTORY:

ITEM_NO	DESCRIP	PPU
T00	Taro; No Topping	5.25
T01	Taro; Boba	5.75
T02	Taro; Lychee	5.75
T03	Taro; Egg Pudding	5.75
J00	Jasmine Milk; No Topping	6.25
J01	Jasmine Milk; Boba	6.75

CUSTOMERS:

CUST_ID	FIRST_NAME	LAST_NAME	STATE	BDAY_MONTH
C1	John	Smith	CA	Aug
C2	Sean	Zhao	WA	May
C3	Abraham	Wald	CA	Sep
C4	Xiao	Li	WA	Nov
C5	Shirley	Li	CA	Jul
C6	Olu	Achebe	OR	Nov

! Question 1

Read in the `orders.csv`, `inventory.csv`, and `customers.csv` files (all located in a `/data` subfolder). Assign these to variables called `orders`, `inventory`, and `customers`, respectively.

Solution:

```
## replace this line with your code
orders <- read.csv("data/orders.csv")
inventory <- read.csv("data/inventory.csv")
customers <- read.csv("data/customers.csv")
```

Answer Check:

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q1.R")})
```

```
All tests passed!
```

Database Terminology

Databases have a rich and complex theory. Much of the modern-day theory surrounding databases is attributed to Edgar F. Codd, a scientist who developed a framework surrounding **relational databases** while working at IBM, in the 60's and 70's. As this is not meant to be a class in database management, we'll only scratch the surface of databases - if you're curious to read more, much of Codd's work is summarized in his book titled "The Relational Model for Database Management".

Here is the gist of things. Again, we can think of a **database** as being a collection of **relations** (i.e. dataframes/tables) comprised of **records** (i.e. rows), where the relations are typically related in some fashion. Each relation in a database has a unique **primary key**, which is the smallest set of variables needed to uniquely determine the rows of the relation For example, the primary key of the `INVENTORY` relation is {`ITEM_NO`}, as each item number corresponds to a unique row in the `INVENTORY` relation.

However, the same cannot be said about the `ORDERS` relation. For example, if I tell you "show me the row of the `ORDERS` relation that has `item_no T02`", you won't be able to do so because there are *multiple* rows with `item_no` value equal to `T02`. As such, we actually need *all three* of `order_no`, `item_id`, and `cust_id`, to uniquely identify the rows of the `ORDERS` relation, and we set the primary key to be the set {`order_no`, `item_no`, `cust_id`} (primary keys consisting of more than one variable are sometimes referred to as **compound keys**).

A **foreign key** is a key (or set of keys) in one relation that corresponds to the primary key of a another relation. Note that it is allowed for foreign keys to point to only one of the keys in a compound primary key. For example, note that the `ITEM_NO` column from the `INVENTORY` table corresponds to the `item_no` column in the `ORDERS` table: hence, we would say that the `INVENTORY:ITEM_NO` key (note the syntax `table_name:column_name`) is a foreign key that points to `ORDERS:item_no`.

Joins

When working with databases, it is sometimes necessary/desired to *combine* one or more dataframes in some way. For example, take the `ORDERS` dataframe: it currently only includes the unique *identifier* of each product, and not the actual *name* of the product. Wouldn't it be nice to include the product names with each of the orders?

I think most of us can do this combination manually fairly easily:

ORDER_NO	ITEM_ID	CUST_ID	DESCRIP
1	T00	C25	Taro; No Topping
1	T02	C25	Taro; Lychee
1	T02	C25	Taro; Lychee
2	U00	NA	Ube; No Topping
2	G03	NA	Original Milk; Egg Pudding
3	T01	NA	Taro; Boba

But, how can we make `R` do this for us? To answer this question, let's break down how we did this combination manually. For example, focusing on the first row:

- We first used the `ORDERS` dataframe to look up the `item_no` of the first order.
- We then found that `item_no` in the `INVENTORY` dataframe.
- Finally, we looked up the `DESCRIP` of the corresponding `item_no`, and added this into the third column of our `ORDERS` dataframe.

This is an example of what is known as a **join** (also known as a **merge**), in which we combine information from multiple tables. The key idea is that we join on/along a foreign key relationship!

There are two main classes of joins: **mutating joins** and **filtering joins**. For now, let's restrict our considerations to mutating joins. As an illustrative example, we'll consider the following two simple tables:

x

ind_x	var_x
one	a
two	b
two	c
three	c
four	d

y

ind_y	var1_y	var2_y
a	cat	piano
b	dog	piano
c	rabbit	violin
e	snake	viola

We can code these tables into `R` as dataframes:

```
x <- data.frame(
  ind_x = c("one", "two", "two", "three", "four"),
  var_x = c("a", "b", "c", "c", "d")
)

y <- data.frame(
  ind_y = c("a", "b", "c", "e"),
  var1_y = c("cat", "dog", "rabbit", "snake"),
  var2y = c("piano", "piano", "violin", "viola")
)
```

Left join

Perhaps the most commonly-used join is that of a **left join**, which is used to add additional information to a table. The syntax of a left join is:

```
left_join(
  x,
  y,
  by = join_by(var_x == ind_y)
)
```

```
  ind_x var_x var1_y  var2y
1   one     a    cat  piano
2   two     b    dog  piano
3   two     c rabbit violin
4 three     c rabbit violin
5  four     d   <NA>   <NA>
```

Note that, by default, `left_join(x, y, ...)` includes all rows of `x`, but not necessarily all rows of `y`. This means that the output of `left_join(x, y, ...)` will (almost) always have the same number of rows as `x`.

Caution

Left joins are **not** symmetric! That is: `left_join(x, y)` will not produce the same output as `left_join(y, x)`, as is indicated below:

```
left_join(
  y,
  x,
  by = join_by(ind_y == var_x)
)
```

```
  ind_y var1_y  var2y ind_x
1     a    cat  piano   one
2     b    dog  piano   two
3     c rabbit violin   two
4     c rabbit violin three
5     e  snake  viola  <NA>
```

Inner Joins, Right Joins, and Full Joins

There are three other kinds of mutating joins: **inner joins**, **right joins**, and **full joins**. All of these are similar to left joins in that they are used to augment existing dataframes with information sourced from another dataframe; the key difference is in which rows are kept post-join.

Right joins keep all rows in `y`:

```
right_join(
  x,
  y,
  by = join_by(var_x == ind_y)
)
```

```
  ind_x var_x var1_y  var2y
1   one     a    cat  piano
2   two     b    dog  piano
3   two     c rabbit violin
4 three     c rabbit violin
5  <NA>     e  snake  viola
```

Inner joins keep only rows present in both `x` and `y`:

```
inner_join(
  x,
  y,
  by = join_by(var_x == ind_y)
)
```

```
  ind_x var_x var1_y  var2y
1   one     a    cat  piano
2   two     b    dog  piano
3   two     c rabbit violin
4 three     c rabbit violin
```

Full joins keep rows present in either `x` or `y`:

```
full_join(
  x,
  y,
  by = join_by(var_x == ind_y)
)
```

```
  ind_x var_x var1_y  var2y
1   one     a    cat  piano
2   two     b    dog  piano
3   two     c rabbit violin
4 three     c rabbit violin
5  four     d   <NA>   <NA>
6  <NA>     e  snake  viola
```

Let's return to our *GauchoBubble* database. Suppose we want to join the `OBJECT` and `INVENTORY` dataframes to include the order information as well as the item descriptions and price-per-units, to obtain a new dataframe with columns `order_no`, `item_no`, `DESCRIP`, and `PPU`, **in that order**. In other words, we want to create a table whose first few rows look like this:

ORDER_NO	ITEM_ID	CUST_ID	DESCRIP	PPU
1	T00	C25	Taro; No Topping	5.25
1	T02	C25	Taro; Lychee	5.75
1	T02	C25	Taro; Lychee	5.75
2	U00	NA	Ube; No Topping	7.00
2	G03	NA	Original Milk; Egg Pudding	5.75
3	T01	NA	Taro; Boba	5.75

! Question 2

A) Which type/s of joins could be used to accomplish this? (If the desired table could be created using several choices of joins, be sure to list them all.)

Solution, part (A):
Replace this line with your answers
Since we want to preserve all rows of the orders relation, it makes sense to use a left join.

B) Now, perform the join. (If you believe there are multiple potential joins we could use, pick one; that is, you do not have to redo the join multiple times). Assign the result to a variable called `orders_items`.

Solution, part (B) :

```
## replace this line with your code
orders_items <- orders %>%
  left_join(
    inventory,
    by = join_by(ITEM_ID == ITEM_NO)
  )
```

Answer Check:

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q2.R")})
```

```
All tests passed!
```

Total Revenue

Something that might be of interest is the total revenue generated by sales at *GauchoBubble* on the day in question.

! Question 3

A) Compute the total revenue generated by each type of item (e.g. display the total amount generated by sales of Taro with No Topping, the total amount generated by sales of Taro with Boba, etc.) Assign the resulting dataframe to a variable called `sales_by_product`. Ensure that the names of your `sales_by_product` dataframe are `ITEM_ID` and `NET_REV`, respectively.

Solution:

```
## replace this line with your code
sales_by_product <- orders_items %>%
  group_by(ITEM_ID) %>%
  summarise(NET_REV = sum(PPU))
```

Answer Check:

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q3.R")})
```

```
All tests passed!
```

B) Display the entirety of your `sales_by_product` dataframe, arranged in descending order of net revenue. Use this to identify which item generated the most revenue, and which item generated the least.

Solution:

```
## replace this line with your code
sales_by_product %>% arrange(desc(NET_REV))
```

```
# A tibble: 16 x 2
   ITEM_ID NET_REV
   <chr>     <dbl>
 1 U00       133
 2 T02        74.8
 3 G03        69
 4 U02        60
 5 T03        57.5
 6 J02        54
 7 U01        52.5
 8 T00        47.2
 9 G01        40.2
10 G00        36.8
11 U03        30
12 G02        28.8
13 J01        20.2
14 J00        18.8
15 T01        17.2
16 J03        13.5
```

We can see that item U00, Ube with No Toppings, generated the most revenue ($133 in total) and item J03, Jasmine with Egg Pudding, generated the least ($13.5)

Answer Check:
There is no autograder for this question; your TA will manually check that your answers are correct.

Customer-Related Considerations

So far we've investigated revenue; let's start investigating customers.

! Question 4

How many customers that arrived in the store (on the day during which the data was collected) had a rewards account? (Recall that the information about customers in the rewards account is found in the `customers` relation.)

Solution:
There are a couple of different ways to go about this. Here is one:

```
## replace this line with your code
orders %>%                    ## join orders and customers
  left_join(
    customers,
    by = join_by(CUST_ID)
  ) %>%
  select(CUST_ID) %>%         ## select the CUST_ID column
  unique() %>%                ## remove duplicates
  na.omit() %>%               ## remove NAs
  nrow()                      ## count the number of rows
```

```
[1] 16
```

Answer Check:
There is no autograder for this question; your TA will manually check that your answers are correct.

! Question 5

Customers from which state generated the most revenue? Justify your answer using the data!

Solution:
Again, there are a couple of different ways to go about this. The key is to note that we need to merge all three relations: the orders relation contains the sales information, the customers relation contains the state information, and the inventory relation contains the price information.

```
## replace this line with your code
orders %>%
  left_join(
    customers,
    by = join_by(CUST_ID)
  ) %>%
  left_join(
    inventory,
    by = join_by(ITEM_ID == ITEM_NO)
  ) %>%
  group_by(
    STATE
  ) %>%
  summarise(
    NET_REV = sum(PPU)
  ) %>% arrange(desc(NET_REV))
```

```
# A tibble: 8 x 2
  STATE NET_REV
  <chr>   <dbl>
1 <NA>    535
2 CA      103
3 WA       28.8
4 AZ       26.2
5 NY       24.2
6 ID       16.8
7 OR       12
8 NM        7.5
```

Answer Check:
There is no autograder for this question; your TA will manually check that your answers are correct.

Part II: Combining Commands

Let's continue investigating the customers in the rewards program. One thing that might be interesting is to construct a sort of **contingency table**, indicating the number of customers with different birthday-month and origin-state pairs. That is, let's work toward constructing a table that looks like:

	AZ	**CA**	**ID**	...
Jan	0	0	0	...
Feb	0	2	0	...
Mar	0	0	0	...
...	...	...	...	

Doing so will require us to review some material from earlier in this week, and will also give us a chance to practice a new dataframe transformation.

! Question 6

What variable type is `BDAY_MONTH`? (Phrase your answer in terms of the variable classification scheme discussed in Lecture 01.) What **data type** (in `R`) should we use to store the information encoded in the `BDAY_MONTH`? (If you aren't familiar with the data types in `R`, please consult the optional Lab00 linked on the course website.)

After answering the above questions, make a copy of the `customers` data frame called `customers2`, and change the data type of the `customers2$BDAY_MONTH` variable to the appropriate data type. **IMPORTANT:** DO NOT CHANGE YOUR ORIGINAL `customers` DATAFRAME, AS DOING SO MAY RESULT IN PROBLEMS WITH THE AUTOGRADER.

Solution: *Replace this line with your answer*
The variable is ordinal, meaning we should use an ordered factor.

```
## replace this line with your code
customers2 <- customers
customers2$BDAY_MONTH <- factor(customers2$BDAY_MONTH,
                                ordered = T,
                                levels = month.abb)
```

Answer Check:

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q6.R")})
```

```
All tests passed!
```

The next step in creating our contingency table is to compute the counts of each month-state pair.

! Question 7

Compute the number of customers in the rewards program with each bday-month and origin-state combination. That is, create a table that displays: the number of CA residents born in Jan, the number of CA residents born in Feb, etc. As a hint, the first few rows of your table should look like this:

STATE	BDAY_MONTH	count
AZ	Oct	1
AZ	NA	1

CA	Feb	2
CA	Apr	1
CA	Jul	2
CA	Aug	2

Assign your answer to a variable called `month_state_counts`, and display the first few rows of the `month_state_counts` dataframe.

Solution:

```
## replace this line with your code
month_state_counts <- customers2 %>%
  group_by(
    STATE, BDAY_MONTH
  ) %>%
  summarise(
    count = n()
  )
```

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

```
head(month_state_counts)
```

```
# A tibble: 6 x 3
# Groups:   STATE [2]
  STATE BDAY_MONTH count
  <chr> <ord>      <int>
1 AZ    Oct            1
2 AZ    <NA>           1
3 CA    Feb            2
4 CA    Apr            1
5 CA    Jul            2
6 CA    Aug            2
```

Answer Check:

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q7.R")})
```

```
All tests passed!
```

Finally, we need to transform our `month_state_counts` dataframe into the contingency table we set out to create. Doing so will require us to **pivot** (aka **widen**) our dataset, which is in essence the

inverse of **melting** (which we discussed in Lecture 02).

Whereas melting takes column names and coerces them into entries in the table, pivoting takes entries in a table and pulls them into column names. For example, consider the following toy dataframe:

```
df2 <- data.frame(
  group = c("A", "A", "B", "B"),
  animal = c("Cat", "Dog", "Cat", "Dog"),
  counts = c(2, 3, 4, 1)
)

df2 %>% pander()
```

group	animal	counts
A	Cat	2
A	Dog	3
B	Cat	4
B	Dog	1

Pivoting the dataframe (using the `pivot_wider()` function from the `tidyverse`) results in:

```
df2 %>% pivot_wider(
  names_from = c(group),
  values_from = counts
) %>% pander()
```

animal	A	B
Cat	2	4
Dog	3	1

! Question 8

Pivot the `month_state_counts` dataframe to create the desired contingency table. Place birth months as rows and states as columns, and arrange the table in ascending order of birth month.

Solution:

```
## replace this line with your code
month_state_counts %>%
  pivot_wider(
    names_from = c(STATE),
    values_from = count
  ) %>% arrange(BDAY_MONTH)

# A tibble: 13 x 10
```

	BDAY_MONTH	AZ	CA	ID	IL	NM	NY	OR	WA	WY
	<ord>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
1	Jan	NA	NA	NA	NA	1	1	NA	NA	NA
2	Feb	NA	2	NA	NA	NA	NA	NA	NA	NA
3	Mar	NA	NA	NA	NA	NA	NA	1	NA	NA
4	Apr	NA	1	NA	1	NA	NA	NA	NA	NA
5	May	NA	NA	NA	NA	NA	NA	NA	1	NA
6	Jun	NA	NA	NA	NA	NA	2	NA	NA	NA
7	Jul	NA	2	1	NA	NA	NA	NA	NA	NA
8	Aug	NA	2	NA	NA	NA	NA	NA	1	NA
9	Sep	NA	4	1	NA	NA	1	NA	1	NA
10	Oct	1	1	NA	NA	NA	NA	NA	NA	NA
11	Nov	NA	NA	NA	NA	NA	NA	1	1	NA
12	Dec	NA	3	NA	NA	NA	1	NA	1	1
13	<NA>	1	NA	NA	NA	NA	NA	NA	NA	NA

Answer Check:
There is no autograder for this question; your TA will manually check that your answers are correct.

Submission Details

Congrats on finishing this PSTAT 100 lab! Please carry out the following steps:

Submission Details

1) Check that all of your tables, plots, and code outputs are rendering correctly in your final `.pdf`.

2) Check that you passed all of the test cases (on questions that have autograders). You'll know that you passed all tests for a particular problem when you get the message "All tests passed!".

3) Submit **ONLY** your `.pdf` to Gradescope. Make sure to match pages to your questions - we'll be lenient on the first few labs, but after a while failure to match pages will result in point penalties.