

Mini Project 2: Data Aquisition

Rain & Ziling

Table of contents

Introduction	1
Approach	2
Crops	2
Fish	8
Animal Products	14
Minerals	15

```
library(tidyverse)
library(stringr)
library(rvest)
library(polite)
library(sf)
library(maps)
library(viridis)
library(leaflet)
library(htmltools)
library(janitor)
library(httr2)
library(httr)
library(lubridate)
library(tidycensus)
library(purrr)
```

Github Source code [Link](#)

Introduction

For our project, we decided to scrape data from the wiki pages of one of our favorite video games, *Stardew Valley*. *Stardew Valley* is a popular indie farming game that allows players to take on the role of a character who inherits a run-down farm from their grandfather. In the game, players can grow crops, raise animals, fish, mine, and engage in social activities with the towns people.

For our project, we were interested in compiling a list of items from the game that can be farmed or collected. The only way to make money from the game is by selling these items, and the price of the item depends on the quality of the item and the profession(s) of the player. Thus, our dataset includes information on the name, category, subcategory, and the different price points of the item depending on item quality (regular, silver, gold, and iridium) and player's profession.

Approach

All of our data has been accumulated from the [Stardew Valley Wiki](#) page. Since each item in the game has a different page and not all of the pages followed a similar structure, we used a combination of harvesting the data in both table form and anywhere on the webpage using `rvest` with `html_text`. In the end, we were able to create a dataset from the more important item categories: crops, fish, animal products, and minerals.

Crops

Crops was the most difficult item to scrape from the wiki, since not all of the pages are structured the same. However, we tried our best to automate where we could.

We start by getting a list of all the different crops in the game.

```
#check that we are allowed to scrape the wiki
robotstxt::paths_allowed("https://stardewvalleywiki.com/Stardew_Valley_Wiki")

[1] TRUE

session <- bow("https://stardewvalleywiki.com/Stardew_Valley_Wiki", force = TRUE)

crops <- bow("https://stardewvalleywiki.com/Crops", force = TRUE)

result <- scrape(crops) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)

seasonal_crops <- result[[134]][2] #table of the season crops so we can use that list

seasonal_crops <- seasonal_crops |>
  mutate(Crops = strsplit(Crops, " • ", fixed = TRUE)) |>
  unnest(Crops) |>
  mutate(Crops = str_replace_all(Crops, " ", "_")) |>
  distinct(Crops)
```

Create our helper functions for crops:

```

# function for getting the price at a given page and css selector
get_price <- function(page, css_selector) {
  page |>
  html_nodes(css_selector) |>
  html_text()
}

# function for creating a tibble of base prices, no profession, for a given crop page
crop_base_prices <- function(crop, tiller = FALSE) {
  url <- str_c("https://stardewvalleywiki.com/", crop)
  page <- read_html(url)

  qualities <- c("regular", "silver", "gold", "iridium")
  prices <- list()

  for (i in seq_along(qualities)) {
    if (tiller) {
      selector <- str_c("tr:nth-child(10) td+ td tr:nth-child(", i, ") td+ td")
    } else {
      selector <- str_c("tr:nth-child(10) tr td:nth-child(1) tr:nth-child(", i, ") td+ td")
    }
    price <- get_price(page, selector)
    prices[[qualities[i]]] <- parse_number(price)
  }

  tibble(
    item = crop,
    regular_price = prices$regular,
    silver_price = prices$silver,
    gold_price = prices$gold,
    iridium_price = prices$iridium
  )
}

```

Create the tibbles for seasonal crops using the helper functions. Note that items 46 (Tea_Leaves), 44(Sweet Gem Berry), 43(Qi_Fruit), 41(Cactus_Fruit), 36(Grape), 4(Coffee_Bean) have issues when using the functions, so we will scrape the data manually without the functions.

```

# list of all our seasonal crops
seasonal_crops_list <- pull(seasonal_crops) # list of our crops tibble

# List of crops, excluding those with known issues
valid_crops_list <- seasonal_crops_list[-c(46, 44, 43, 41, 36, 4)]

# Base prices without profession
base_crop_prices <- valid_crops_list |>

```

```

purrr::map_dfr(~ crop_base_prices(.x)) |>
mutate(profession = as.character(NA))

# Prices with Tiller profession
tiller_crop_prices <- valid_crops_list |>
  purrr::map_dfr(~ crop_base_prices(.x, tiller = TRUE)) |>
  mutate(profession = "tiller")

# Combine base and tiller crop prices
seasonal_crop_prices <- bind_rows(base_crop_prices, tiller_crop_prices)
seasonal_crop_prices

# A tibble: 80 x 6
  item      regular_price silver_price gold_price iridium_price profession
  <chr>          <dbl>         <dbl>    <dbl>         <dbl> <chr>
1 Blue_Jazz           50           62       75          100 <NA>
2 Carrot              35           43       52           70 <NA>
3 Cauliflower       175          218      262          350 <NA>
4 Garlic             60           75       90          120 <NA>
5 Green_Bean         40           50       60           80 <NA>
6 Kale             110          137      165          220 <NA>
7 Parsnip           35           43       52           70 <NA>
8 Potato            80          100      120          160 <NA>
9 Rhubarb          220          275      330          440 <NA>
10 Strawberry       120          150      180          240 <NA>
# i 70 more rows

```

Do the same for non seasonal crops:

```

# Non-seasonal crops list, excluding problematic items
other_crops <- c("Apple", "Blackberry", "Pomegranate", "Wild_Plum", "Apricot",
  "Cherry", "Spice_Berry", "Peach", "Orange", "Crystal_Fruit",
  "Banana", "Mango", "Fiddlehead_Fern")[-c(10, 7, 4, 2)]

# Base prices without profession
base_other_crops <- other_crops |>
  purrr::map_dfr(~ crop_base_prices(.x)) |>
  mutate(profession = as.character(NA))

# Prices with Tiller profession
tiller_other_crops <- other_crops |>
  purrr::map_dfr(~ crop_base_prices(.x, tiller = TRUE)) |>
  mutate(profession = "tiller")

# Combine base and tiller prices into one table and arrange by item

```

```
nonseasonal_crop_tbl <- bind_rows(base_other_crops, tiller_other_crops) |>
  arrange(item)
```

Finally, create a function for the weird crops that have missing quality or selector path was different

```
#function for the crops that do not have different qualities
```

```
crop_weird_prices <- function(item, selector){
  url <- str_c("https://stardewvalleywiki.com/", item)
  page <- read_html(url)
  regular_price <- get_price(page, selector)

  tibble(item = item,
         regular_price = parse_number(regular_price))
}
```

```
#function for the crops that have different qualities. the Berry is for the fruits that have a we
```

```
crop_weird_prices_w_quality <- function(crop, tiller = FALSE, berry = FALSE ){
  url <- str_c("https://stardewvalleywiki.com/", crop)
  page <- read_html(url)

  qualities <- c("regular", "silver", "gold", "iridium")
  prices <- list()

  for (i in seq_along(qualities)) {
    if (tiller) {
      selector <- str_c("tr:nth-child(11) td+ td tr:nth-child(", i, ") td+ td")
    } else if (berry){
      selector <- str_c("tr:nth-child(9) tr:nth-child(", i, ") td+ td")
    }else {
      selector <- str_c("tr:nth-child(11) tr td:nth-child(1) tr:nth-child(", i, ") td+ td")
    }
    price <- get_price(page, selector)
    prices[[qualities[i]]] <- parse_number(price)
  }

  tibble(
    item = crop,
    regular_price = prices$regular,
    silver_price = prices$silver,
    gold_price = prices$gold,
    iridium_price = prices$iridium
  )
}
```

Now we make all of the tibbles for the weird crops.

```

# Tea Leaves
base_tea_leaves <- crop_weird_prices("Tea_Leaves",
                                     "tr:nth-child(10) tr td:nth-child(1) td+ td")
tiller_tea_leaves <- crop_weird_prices("Tea_Leaves",
                                       "tr:nth-child(10) td+ td td+ td")

tea_leaves <-bind_rows(base_tea_leaves, tiller_tea_leaves)

# Qi_Fruit
base_qi_fruit <-crop_weird_prices("Qi_Fruit",
                                  "tr:nth-child(9) tr td:nth-child(1) td+ td")
tiller_qi_fruit <-crop_weird_prices("Qi_Fruit",
                                    "tr:nth-child(9) td+ td td+ td")

qi_fruit <-bind_rows(base_qi_fruit, tiller_qi_fruit)

# Cactus fruit
cactus_fruit <- crop_weird_prices_w_quality("Cactus_Fruit")
cactus_fruit_tiller <- crop_weird_prices_w_quality("Cactus_Fruit", tiller = TRUE)

cactus_fruit <-bind_rows(cactus_fruit, cactus_fruit_tiller)

# Grape
grape <- crop_weird_prices_w_quality("Grape")
grape_tiller <- crop_weird_prices_w_quality("Grape", tiller = TRUE)

grape <-bind_rows(grape, grape_tiller)

# Coffee_bean
coffee_bean <- crop_weird_prices_w_quality("Coffee_Bean")

# Wild_plum
wild_plum <- crop_weird_prices_w_quality("Wild_Plum", berry = TRUE)

# Spice_berry
spice_berry <- crop_weird_prices_w_quality("Spice_Berry", berry = TRUE)

# Crystal_Fruit
crystal_fruit <- crop_weird_prices_w_quality("Crystal_Fruit", berry = TRUE)

# Finally, blackberry is just weird and likes to be different, so we did not use a function for it
#Blackberry

# Base
url <- str_c("https://stardewvalleywiki.com/", "Blackberry")

```

```

page <- read_html(url)

qualities <- c("regular", "silver", "gold", "iridium")
prices <- list()

# Loop to retrieve and parse prices
for (i in seq_along(qualities)) {
  price <- get_price(page, str_c("tr:nth-child(9) tr td:nth-child(1) tr:nth-child(", i, ") td+ td")
  prices[[qualities[i]]] <- parse_number(price)
}

blackberry <- tibble(
  item = "Blackberry",
  regular_price = prices$regular,
  silver_price = prices$silver,
  gold_price = prices$gold,
  iridium_price = prices$iridium
)

```

Now, we can combine all of the crop tibbles into one:

```

# First chunks of crops
draft_crops <- bind_rows(seasonal_crop_prices,
  nonseasonal_crop_tbl,
  tea_leaves,
  qi_fruit,
  cactus_fruit,
  grape,
  coffee_bean,
  wild_plum,
  blackberry,
  spice_berry,
  crystal_fruit) |>

  arrange(item)

```

Lastly, we can add in the category variable and the subcategory variable. to makes things easier, we decided the subcategory would be the crop's season. Then, we write it to a csv in case the website changes or updates.

```

seasons <- result[[134]] %>%
  select(Season = 1, Crops = 2) |>
  mutate(Crops = strsplit(Crops, " • ", fixed = TRUE)) |>
  unnest(Crops) |>
  mutate(Crops = str_replace_all(Crops, " ", "_"))

crop_prices <- draft_crops |>
  left_join(seasons, join_by(item == Crops))|>

```

```

mutate(category = "crop",
       sub_category = str_c(Season, " Crop"))|>
select(-Season)

write.csv(crop_prices, "crop_prices.csv")

head(crop_prices, n = 10)

# A tibble: 10 x 8
  item regular_price silver_price gold_price iridium_price profession category
  <chr>          <dbl>         <dbl>    <dbl>         <dbl> <chr>      <chr>
1 Amar~          150          187      225          300 <NA>      crop
2 Amar~          165          205      247          330 tiller   crop
3 Anci~          550          687      825         1100 <NA>      crop
4 Anci~          605          755      907         1210 tiller   crop
5 Apple          100          125      150          200 <NA>      crop
6 Apple          110          137      165          220 tiller   crop
7 Apri~           50           62       75          100 <NA>      crop
8 Apri~           55           68       82          110 tiller   crop
9 Arti~          160          200      240          320 <NA>      crop
10 Arti~          176          220      264          352 tiller   crop
# i 1 more variable: sub_category <chr>

```

Fish

Fish was the second most difficult item to scrape from the wiki, since again not all of the pages are structured the same. However, we were able identify 4 different pages in which we could write functions to automate.

We start by getting a list of all the different fish in the game.

```

# Making sure that this url is scrapable
fish <- bow("https://stardewvalleywiki.com/Fish", force = TRUE)

# Scraping table to get a list of all the fish
result <- scrape(fish) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)

# The correct table for the list of fish, and only keeping the names of the fish column
fishes <- result[[225]][2]

# However, it is formatted very poorly so we need to tidy it up
fishes <- fishes |>
  mutate(Fish = strsplit(Fish, " • ", fixed = TRUE)) |>

```



```

unnest(Fish) |>
# splitting the string since " • " was used to separate all fish
mutate(Fish = str_replace_all(Fish, " ", "_")) |>
distinct(Fish) |>
# this is a fish that is in the data set twice but with different spacing
filter(Fish != "_Super_Cucumber")

# This is a tibble with the subcategories of the fish and the fish name for joining later
subcategory <- result[[225]] |>
select(Location = 1, Fish = 2) |>
mutate(Fish = strsplit(Fish, " • ", fixed = TRUE)) |>
unnest(Fish) |>
mutate(Fish = str_replace_all(Fish, " ", "_"))

```

Create our helper functions for fish:

```

# function for getting the price at a given page and css selector
get_price <- function(page, css_selector) {
  page |>
  html_nodes(css_selector) |>
  html_text()
}

```

```

# function for creating a tibble of prices for a given fish

```

```

# this functions output a tibble of our fish
# and the 4 different prices of the fish dependent on quality

```

```

# fish_base_prices takes our fish name,
# and takes a profession if we specify true or false,
# as well as the "nthchild_num" value for where the price is being store on that website

```

```

fish_base_prices <- function(fish, fisher = FALSE, angler = FALSE, nthchild_num) {
  url <- str_c("https://stardewvalleywiki.com/", fish)
  page <- read_html(url)

```

```

  qualities <- c("regular", "silver", "gold", "iridium")
  prices <- list()

```

```

  for (i in seq_along(qualities)) {
    if (fisher) {
      selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(2) tr:nth-child(", i, ")
    } else if (angler) {
      selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(3) tr:nth-child(", i, ")
    }
    else {

```

```

    selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(1) tr:nth-child(", i, ")
  }
  price <- get_price(page, selector)
  prices[[qualities[i]]] <- parse_number(price)
}

tibble(
  item = fish,
  regular_price = prices$regular,
  silver_price = prices$silver,
  gold_price = prices$gold,
  iridium_price = prices$iridium
)
}

```

As well as the function for the fish with a different webpage format.

```

# this functions output a tibble of our fish,
# and the 2 different prices of the fish dependent on quality

```

```

# fish_base_prices takes our fish name,
# and takes a profession if we specify true or false,
# as well as the "nthchild_num" value for where the price is being store on that website

```

```

fish_base_prices2 <- function(fish, fisher = FALSE, angler = FALSE, nthchild_num) {
  url <- str_c("https://stardewvalleywiki.com/", fish)
  page <- read_html(url)

  qualities <- c("regular", "silver", "gold", "iridium")
  prices <- list()

  for (i in seq_along(qualities)) {
    if (fisher) {
      selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(2) tr:nth-child(", i, ")
    } else if (angler) {
      selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(3) tr:nth-child(", i, ")
    }
    else {
      selector <- str_c("tr:nth-child(", nthchild_num,") tr td:nth-child(1) tr:nth-child(", i, ")
    }
    price <- get_price(page, selector)
    prices[[qualities[i]]] <- parse_number(price)
  }

  tibble(
    item = fish,

```

```

    regular_price = prices$regular,
    silver_price = prices$silver,
  )
}

```

Now, we will load in our fishes lists so for the type of webpage format they have and then apply our function to the fishes to find their prices.

```
fishes_list <- pull(fishes) # List of our fishes tibble to view, then dividing up the fish by the
```

```
# Loading in the fish we know that are tr:nth-child(14) in the html (these fishes were found in t
```

```
fishfor14 <- readRDS("fishfor14.RDS")
```

```
fishfor14
```

```

[1] "Mutant_Carp"      "Radioactive_Carp" "Albacore"         "Anchovy"
[5] "Eel"              "Flounder"         "Halibut"          "Herring"
[9] "Octopus"          "Pufferfish"       "Red_Mullet"       "Red_Snapper"
[13] "Sardine"          "Sea_Cucumber"    "Squid"            "Super_Cucumber"
[17] "Tilapia"          "Tuna"            "Bream"            "Catfish"
[21] "Chub"             "Dorado"          "Goby"             "Lingcod"
[25] "Perch"            "Pike"            "Rainbow_Trout"    "Salmon"
[29] "Shad"             "Smallmouth_Bass" "Sunfish"          "Tiger_Trout"
[33] "Walleye"          "Bullhead"        "Carp"             "Largemouth_Bass"
[37] "Midnight_Carp"    "Sturgeon"        "Woodskip"         "Ghostfish"
[41] "Ice_Pip"          "Stonefish"       "Sandfish"         "Slimejack"
[45] "Void_Salmon"      "Blobfish"        "Midnight_Squid"   "Spook_Fish"
[49] "Blue_Discus"      "Lionfish"        "Stingray"

```

```
# Loading in the fish we know that are tr:nth-child(15) in the html, same as above
```

```
fishfor15 <- readRDS("fishfor15.RDS")
```

```
fishfor15
```

```

[1] "Angler"           "Crimsonfish"      "Glacierfish"
[4] "Glacierfish_Jr." "Legend"           "Legend_II"
[7] "Ms._Angler"       "Son_of_Crimsonfish" "Lava_Eel"
[10] "Scorpion_Carp"

```

```
# Loading in the fish we know that are tr:nth-child(10) in the html, same as above
```

```
fishfor10 <- readRDS("fishfor10.RDS")
```

```
fishfor10
```

```
[1] "Clam"    "Cockle" "Mussel" "Oyster"
```

```
# Loading in the fish we know that are tr:nth-child(10) in the html, same as above
```

```
fishleft <- readRDS("fishleft.RDS")
```

```
fishleft
```

```

[1] "Crab"          "Crayfish"      "Lobster"       "Periwinkle"    "Shrimp"
[6] "Snail"

# Creating list of tbl's to store prices so that we can bind into one big tibble
fish_prices <- vector("list", length = 12)

# Base prices without profession for tr:nth-child(14)
fish_prices[[1]] <- fishfor14 |>
  purrr::map_dfr(~ fish_base_prices(.x, nthchild_num = 14)) |>
  mutate(profession = as.character(NA))

# Prices with Fisher profession
fish_prices[[2]] <- fishfor14 |>
  purrr::map_dfr(~ fish_base_prices(.x, fisher = TRUE, nthchild_num = 14)) |>
  mutate(profession = "fisher")

# Prices with Angler profession
fish_prices[[3]] <- fishfor14 |>
  purrr::map_dfr(~ fish_base_prices(.x, angler = TRUE, nthchild_num = 14)) |>
  mutate(profession = "angler")

# Base prices without profession for tr:nth-child(15)
fish_prices[[4]] <- fishfor15 |>
  purrr::map_dfr(~ fish_base_prices(.x, nthchild_num = 15)) |>
  mutate(profession = as.character(NA))

# Prices with Fisher profession
fish_prices[[5]] <- fishfor15 |>
  purrr::map_dfr(~ fish_base_prices(.x, fisher = TRUE, nthchild_num = 15)) |>
  mutate(profession = "fisher")

# Prices with Angler profession
fish_prices[[6]] <- fishfor15 |>
  purrr::map_dfr(~ fish_base_prices(.x, angler = TRUE, nthchild_num = 15)) |>
  mutate(profession = "angler")

# Base prices without profession for tr:nth-child(10)
fish_prices[[7]] <- fishfor10 |>
  purrr::map_dfr(~ fish_base_prices(.x, nthchild_num = 10)) |>
  mutate(profession = as.character(NA))

# Prices with Fisher profession
fish_prices[[8]] <- fishfor10 |>
  purrr::map_dfr(~ fish_base_prices(.x, fisher = TRUE, nthchild_num = 10)) |>
  mutate(profession = "fisher")

```

```

# Prices with Angler profession
fish_prices[[9]] <- fishfor10 |>
  purrr::map_dfr(~ fish_base_prices(.x, angler = TRUE, nthchild_num = 10)) |>
  mutate(profession = "angler")

# Base prices without profession for tr:nth-child(10) but only two qualities
fish_prices[[10]] <- fishleft |>
  purrr::map_dfr(~ fish_base_prices2(.x, nthchild_num = 10)) |>
  mutate(profession = as.character(NA))

# Prices with Fisher profession
fish_prices[[11]] <- fishleft |>
  purrr::map_dfr(~ fish_base_prices2(.x, fisher = TRUE, nthchild_num = 10)) |>
  mutate(profession = "fisher")

# Prices with Angler profession
fish_prices[[12]] <- fishleft |>
  purrr::map_dfr(~ fish_base_prices2(.x, angler = TRUE, nthchild_num = 10)) |>
  mutate(profession = "angler")

```

Finally we will take our fish prices and then create one big tibble.

```

# first tbl in fish prices assigned to our final tibble
tidy_fish_prices <- fish_prices[[1]]

# for loop for iterating each tbl in our fish prices list to our final tibble
for (i in 2:12){
  tidy_fish_prices <- bind_rows(tidy_fish_prices, fish_prices[[i]])
}

# viewing and alphabetizing our tidy fish tbl
# also joining our subcategories and assigning category
(tidy_fish_prices <- tidy_fish_prices |>
  left_join(subcategory, join_by(item == Fish)) |>
  mutate(category = "fish") |>
  rename(sub_category = Location) |>
  arrange(item))

```

A tibble: 318 x 8

	item	regular_price	silver_price	gold_price	iridium_price	profession
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	Albacore	75	93	112	150	<NA>
2	Albacore	93	116	140	187	fisher
3	Albacore	112	139	168	225	angler
4	Anchovy	30	37	45	60	<NA>
5	Anchovy	37	46	56	75	fisher

```

6 Anchovy          45          55          67          90 angler
7 Angler           900         1125         1350         1800 <NA>
8 Angler           900         1125         1350         1800 <NA>
9 Angler          1125         1406         1687         2250 fisher
10 Angler          1125         1406         1687         2250 fisher
# i 308 more rows
# i 2 more variables: sub_category <chr>, category <chr>

# writing our tbl as a csv so that we can join with the other items
write.csv(tidy_fish_prices, "fish_prices.csv")
head(tidy_fish_prices, n = 10)

# A tibble: 10 x 8
  item      regular_price silver_price gold_price iridium_price profession
  <chr>          <dbl>         <dbl>    <dbl>         <dbl> <chr>
1 Albacore         75           93       112           150 <NA>
2 Albacore         93          116       140           187 fisher
3 Albacore        112          139       168           225 angler
4 Anchovy          30           37        45            60 <NA>
5 Anchovy          37           46        56            75 fisher
6 Anchovy          45           55        67            90 angler
7 Angler           900         1125       1350          1800 <NA>
8 Angler           900         1125       1350          1800 <NA>
9 Angler          1125         1406       1687          2250 fisher
10 Angler          1125         1406       1687          2250 fisher
# i 2 more variables: sub_category <chr>, category <chr>

```

Animal Products

Animal products was one of the easier items to scrape since we were able to scrape the data from a table.

```

#first be polite and check that we can scrape it
robotstxt::paths_allowed("https://stardewvalleywiki.com/Animal_Products_Profitability")

[1] TRUE

session <- bow("https://stardewvalleywiki.com/Animal_Products_Profitability", force = TRUE)

#take the second table, because that is the one we are interested in
result_animals <- scrape(session) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)

sd_animal_prices <- result_animals[[2]]

```

From here all we have to do is clean up our tibble.

```
#clean up the sd_animal_prices tibble
tidy_sd_animal_price <- sd_animal_prices |>
  clean_names()|>
  select(item,
         profession,
         quality,
         sell_price)|> #select only the columns we want
  group_by(item, profession)|>
  pivot_wider(names_from = quality,
              values_from = sell_price,
              names_glue = "{quality}_price",
              values_fn = mean)|>
  clean_names()|>
  mutate(category = "animal product",
         profession = ifelse(profession == "-", NA, profession))

#write the final version to a csv
write.csv(tidy_sd_animal_price, "animal_product_prices.csv")
head(tidy_sd_animal_price, n = 10)
```

A tibble: 10 x 7

Groups: item, profession [10]

	item	profession	regular_price	silver_price	gold_price	iridium_price	category
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	Egg	<NA>	50	62	75	100	animal ~
2	Egg	Rancher	60	75	90	120	animal ~
3	Egg	Artisan	50	62	75	100	animal ~
4	Larg~	<NA>	95	118	142	190	animal ~
5	Larg~	Rancher	114	142	171	228	animal ~
6	Larg~	Artisan	95	118	142	190	animal ~
7	Void~	<NA>	65	81	97	130	animal ~
8	Void~	Rancher	78	97	117	156	animal ~
9	Void~	Artisan	65	81	97	130	animal ~
10	Duck~	<NA>	95	118	142	190	animal ~

Minerals

Minerals was one of the easier items to scrape since we were able to scrape the data from a table. However assigning the category and subcategories is what made the process a little more tedious.

```
#first be polite and check that we can scrape it
robotstxt::paths_allowed("https://stardewvalleywiki.com/Minerals")
```

```

[1] TRUE

session <- bow("https://stardewvalleywiki.com/Minerals", force = TRUE)

result_minerals <- scrape(session) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)
#interested in tables 1-4

#This function takes a scraped minerals table and preps it for joining with other datasets
tidy_minerals <- function(data, sub_cat){
  data|>
  clean_names()|>
  mutate(item = name,
          category = "mineral",
          sub_category = sub_cat)|>
  rename(regular_sell_price = sell_price)|>
  pivot_longer(
    cols = c(gemologist_sell_price,
              regular_sell_price),
    names_to = "profession",
    values_to = "sell_price"
  )|>
  select(item,
         profession,
         sell_price,
         category,
         sub_category)|>
  mutate(sell_price = as.numeric(str_extract(sell_price, '(?<=data-sort-value=")\\d+')),
         profession = ifelse(profession == "gemologist_sell_price",
                              "gemologist", NA))
}

#use function for the 1-3 tables using a for loop
minerals_tbl <- vector("list", length = 4)
mineral_sub_cat <- c("foraged mineral",
                    "gem",
                    "geode mineral",
                    "geode")

for (i in 1:3){
  minerals_tbl[[i]] <- tidy_minerals(result_minerals[[i]], mineral_sub_cat[i])
}

```



```

#clean up the variable names so that it is ready for the row bind.
# make sure the category is all mineral, and the sub_category is correct
minerals_tbl[[4]]<- result_minerals[[4]]|>
  clean_names()|>
  mutate(item = name,
         category = "mineral",
         sub_category = "geode",
         sell_price = as.numeric(str_extract(sell_price, '(?<=data-sort-value=")\\d+')),
         profession = NA)|>
  select(item, sell_price, category, sub_category, profession)

tidy_sd_minerals_price <- bind_rows(minerals_tbl)

Write it to a csv in case the website changes or updates.

write.csv(tidy_sd_minerals_price, "minerals_prices.csv")
head(tidy_sd_minerals_price, n = 10)

```

```

# A tibble: 10 x 5
  item           profession sell_price category sub_category
<chr>          <chr>         <dbl> <chr>    <chr>
1 Quartz      gemologist         32 mineral foraged mineral
2 Quartz      <NA>            25 mineral foraged mineral
3 Earth Crystal gemologist         65 mineral foraged mineral
4 Earth Crystal <NA>            50 mineral foraged mineral
5 Frozen Tear  gemologist         97 mineral foraged mineral
6 Frozen Tear  <NA>            75 mineral foraged mineral
7 Fire Quartz  gemologist        130 mineral foraged mineral
8 Fire Quartz  <NA>            100 mineral foraged mineral
9 Emerald      gemologist        325 mineral gem
10 Emerald     <NA>            250 mineral gem

```

Combined Dataset

Lastly, we merged together all of the data sets for each of the 4 categories: crops, fish, animal products, and minerals. This dataset is now ready to be used for different item comparisons depending item category.

```

# binding rows for all of different categories
stardew_items <- bind_rows(crop_prices,
                           tidy_sd_animal_price,
                           tidy_sd_minerals_price,
                           tidy_fish_prices)

write.csv(stardew_items, "stardew_items.csv")

```

```
head(stardew_items, n = 10)
```

```
# A tibble: 10 x 9
```

	item	regular_price	silver_price	gold_price	iridium_price	profession	category
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>
1	Amar~	150	187	225	300	<NA>	crop
2	Amar~	165	205	247	330	tiller	crop
3	Anci~	550	687	825	1100	<NA>	crop
4	Anci~	605	755	907	1210	tiller	crop
5	Apple	100	125	150	200	<NA>	crop
6	Apple	110	137	165	220	tiller	crop
7	Apri~	50	62	75	100	<NA>	crop
8	Apri~	55	68	82	110	tiller	crop
9	Arti~	160	200	240	320	<NA>	crop
10	Arti~	176	220	264	352	tiller	crop

```
# i 2 more variables: sub_category <chr>, sell_price <dbl>
```