Setting up a WooCommerce image gallery for your shop is a grueling process if you use the online forms. Thankfully, you can import goods and setup an image gallery using a simple CSV file. Now, if you have a few products and a few images for each product, preparing the CSV for bulk import using your favorite spreadsheet program is trivial. But, if you have lots of products and lots of images per product, manually creating the CSV via spreadsheet becomes a considerable task. However, if we know the CSV structure, we leverage R to help us automate this task in a more scaleable way.

**Objectives:**

1. [Understand what’s in a WooCommerce CSV file](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#WooCommerce_CSV)
2. [No Images – Importing the CSV to WooCommerce](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#CSV_No_Images)
   1. [WooCommerce Import Process](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#WC_Import_PROC)
3. [One Image – Importing the CSV to WooCommerce](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#CSV_ONE_IMAGE)
   1. File and Folder Structure
   2. [FTP Upload Process](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#FTP_Upload)
4. [Image Gallery – Importing the CSV to WooCommerce](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#Image_Gallery)
   1. File and Folder Structure
   2. [The Final CSV](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#The_Final_CSV)
5. What R needs to generate the CSV?
   1. [Developing an R Process](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#developing_R_PROC)
   2. [The Final R Function](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#the_function)

**What’s in a WooCommerce CSV Import File**

The WooCommerce CSV import file allows you boot up an e-commerce shop in a matter of minutes. One of the coolest aspects of the CSV import process, though, is that you can also setup a WooCommerce image gallery. All you need is a unique identifier for your products such as a SKU; some meta data about your products like descriptions and prices; and some images in a structured file system. Put all these elements together and you could have a proof of concept shop up and running in no time. If you want to take it to the next level though, you will still need to fill in “other” important details in your CSV file, setup a merchant account, get licenses, etc.

While my main purpose in this article is to describe how you might setup an WooCommerce image gallery using R, you might be interested in what “other” types of data could go into your WooCommerce CSV file. To give you a sense of what you have to work with, check out the listing in the drop-down-box below. Also, to help get you started on setting up your own CSV file, I've included a blank template below with only the column headings.

[[Download Empty WooCommerce CSV]](https://cdn.r-bar.net/PID855/empty-wc-product.csv)

**Simple WooCommerce CSV File: No Images**

To get familiar with the WooCommerce import process, we are going to use three donation style products valued at 0.01$, 0.10$, and 1.00$. To keep things simple, we will only be using a few columns: SKU, Name, Description, Regular price, and Images. Below is the table with our data. To download a copy of the CSV, click the link below the table.

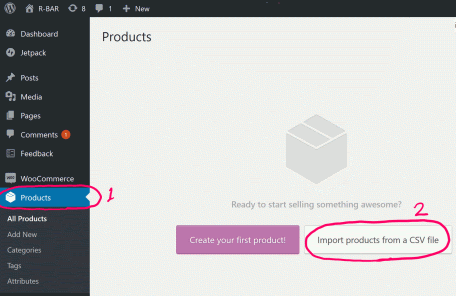
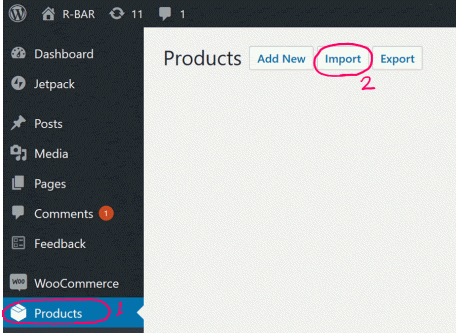
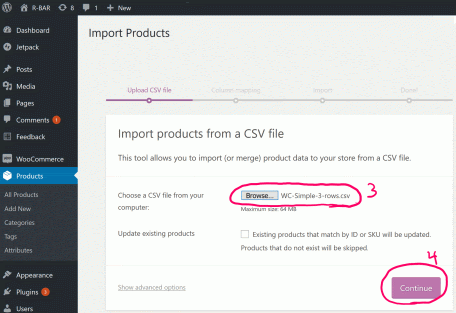
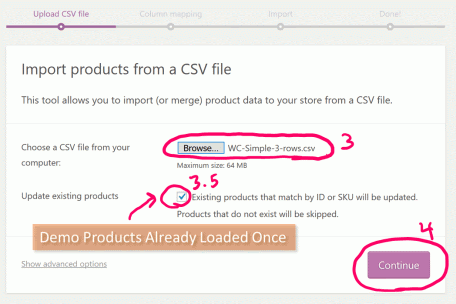
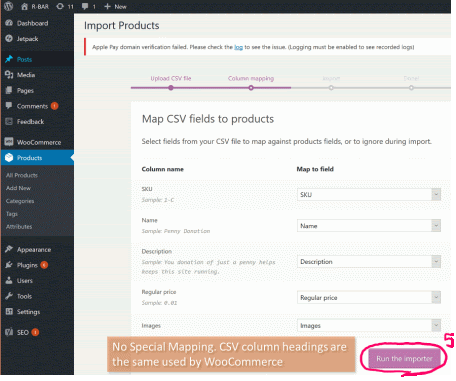
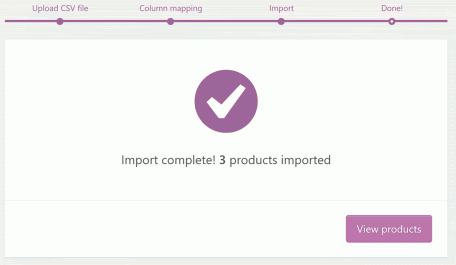
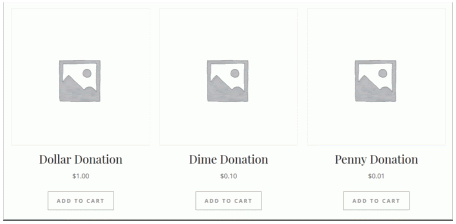
| **SKU** | **Name** | **Description** | **Regular price** | **Images** |
| --- | --- | --- | --- | --- |
| 1-C | Penny Donation | You donation of just a penny helps keeps this site running. | 0.01 |  |
| 10-C | Dime Donation | You donation of just a dime helps keeps this site running. | 0.1 |  |
| 100-C | Dollar Donation | You donation of just a dollar helps keeps this site running. | 1 |  |

[[csv file for table]](https://cdn.r-bar.net/PID855/WC-Simple-3-rows.csv)

**WooCommerce Import Process**

**Note:** This post is a minimalist example of setting up a WooCommerce shop from **0 products** using an imported CSV. I have no idea what effects this will have on an existing store setup. Before trying something new with your store, back it up.

Starting from a fresh WooCommerce install, we would continue with the following steps:

1. Make and save the CSV file to a place you will easily find it again. For now a folder on your desktop is likely a great place to put it. You’ll need it for later steps in this post. Here are some ready made CSVs. The first CSV, we’ll being using throughout this post.
   1. [Simple CSV for our examples](https://cdn.r-bar.net/PID855/WC-Simple-3-rows.csv)
   2. [Expanded CSV](https://cdn.r-bar.net/PID855/empty-wc-product.csv)
2. Sign into your WordPress dashboard
3. Click Products in your WordPress menu panel (Assumes fresh WooCommerce Install).
4. Find the Import Button (**2 options**)
   1. If you have 0 products, and this your first time through this process, click import as shown. If you are updating the demo products with images described in later sections, see the next bullet.  
      
   2. If you already have the demo products in your store, the location of the import button moves to the top of the screen. Press the button as shown and continue to the next step.  
      
5. Setup the Import Conditions (**2 options**)
   1. If you have 0 products, Upload your CSV file as shown below. If you are updating the demo products with images described in later sections, see the next bullet.  
      
   2. If you already have the demo products in your store, click the check box asking about *“Existing products that match by ID or SKU will be updated. Products that do not exist will be skipped.”*
6. Column Mapping – Here is where you will line up (map) the columns in your csv with the columns used by WooCommerce. Because we are using standard WooCommerce names there won’t be anything to do here but verify the columns are mapping correctly as shown below.  
   
7. Import by pressing the “Run the Importer” button at the bottom of the mapping list.
8. Seeing the “Import Complete” message below is a good sign the import worked with no errors noted.  
   
9. Check that it worked by visiting your website’s store front. You should see something like this in your shop:  
   

If you were able to import the data successfully, continue to the next section. There you'll see how to import a single image.

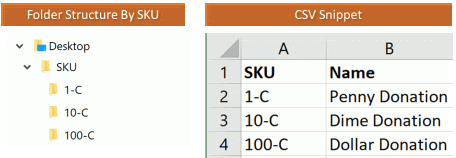
**Simple WooCommerce CSV File: Single Image**

So, now we are ready to associate an image with the products. When it comes to adding an image into a WooCommerce shop, a little planning goes a long way. And, that planning will pay dividends when we automate the process with R. The general idea of what we are going to do is line up the SKU in our csv with a file system folder titled with that SKU containing the images. Let's see how this might work.

First, let's look at our CSV from above, reshown here:

| **SKU** | **Name** | **Description** | **Regular price** | **Images** |
| --- | --- | --- | --- | --- |
| 1-C | Penny Donation | You donation of just a penny helps keeps this site running. | 0.01 |  |
| 10-C | Dime Donation | You donation of just a dime helps keeps this site running. | 0.1 |  |
| 100-C | Dollar Donation | You donation of just a dollar helps keeps this site running. | 1 |  |

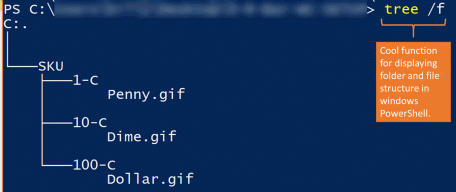
If we assume that our SKUs are unique in our CSV, we can use this as a way to organize our image files. To get us started, make a folder called SKU (preferably on your desktop for now) and enter the folder. Next, make a folder for each SKU listed in the CSV. Warning, making folders by hand like this is error prone due to typos. Later I'll show you how to do this with R. Right now, the goal is to see how all the pieces fit together. To that end, here is a visual of the folder structure and some of the CSV side by side.



To go any further, we need some images. Below you will find a table with SKU, the image file name, and the image. Save each of the images to their respective SKU folder on your desktop.

|  |  |  |
| --- | --- | --- |
| **SKU** | **Image Name** | **Image** |
| 100-C | Dollar.gif | Dollar |
| 10-C | Dime.gif | Dime |
| 1-C | Penny.gif | Penny |

After you've saved the images. Your file and folder structure should look like the tree below. Aside, learned how to make the tree in powershell when writing this article. The tree command is a nice feature.



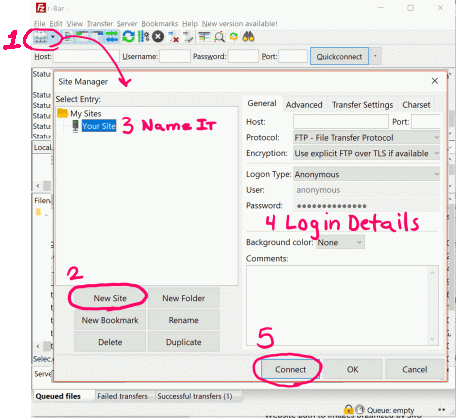
With the basic folder structure in place, there are three more things to do, i) Upload the images and folder structure to our website, ii) Update our WooCommerce CSV with html links to our images, and iii) import the CSV to WooCommerce.

**Upload Images using FTP**

The next thing we need to do is upload the file structure we just made to our WordPress website. To do this we are going to use FTP. Before going any further, make sure you know the following login details to your FTP service:

* The host, port, and FTP protocol used by your website hosting company.
* Your username
* Your password

In the FTP example that follows, I'm using [Filezilla](https://filezilla-project.org/download.php?type=client) to upload the SKU folder we just created. In FileZilla, I recommend making a profile with your login details. To make a profile, click on the “open site manager” icon, fill in the login information, and click connect. When you want to login later, the information will be saved.

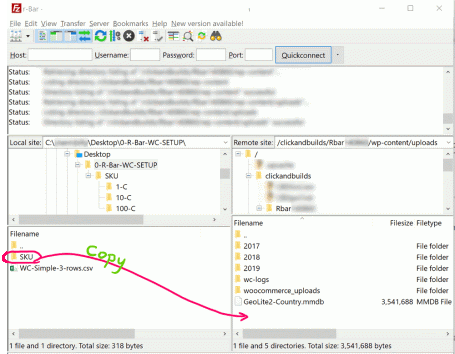


Once you are signed in, you'll notice that the left panel shows your local file system and the right panel shows the remote file system. The right panel is where your website lives. Be careful what you do here. Moving or deleting files and folders without knowing what your doing can break your website. Before proceeding, make sure you have a recovery plan (i.e., a backup). Mistakes happen even when you know what your doing. Be prepared.

Anyway, you need to locate the following folder in the remote file system:

ftp://[your site path]/wp-content/uploads/

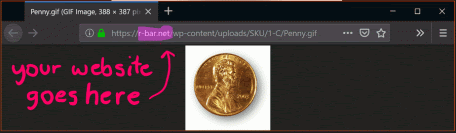
When you find the folder, you'll likely see sub-folders with date labels such as 2018, 2019, etc. This is a place where we can upload our image file structure. Before we can upload the images, we need to locate the folder we want to upload using FileZilla's right panel. If you made the folder structure on your desktop, this should be straight forward.



The last step here is to drag the SKU folder from your local file system to the wp-content/uploads/ folder in the remote file system. When you are done, you should have a full copy, with images, of the SKU folder in the remote file system.

To make sure that everything is working correctly, try to access one of the images via HTTP. To do this, point your web browser at a location like the following:

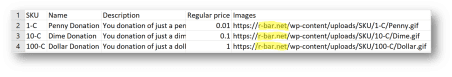
http://[your website]/wp-content/uploads/SKU/1-C/Penny.gif



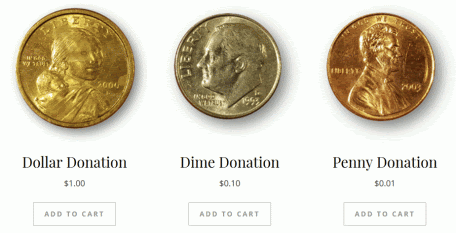
Make sure to use the name of “your website” after http:// in the URL above. If you are able to see the image using your web browser, then you are ready to move on to the next step.

**Adding the Image to the CSV File**

Great, so you have the images in a location on your website where you can access them using http. To add the image to our WooCommerce bulk import CSV, we are going to need the full URL of each image for each product. Since we only have three products and one image for each product, the resulting CSV file will look like this.



If you are working along, be sure to update the http://[your website] to reflect your actual web address. Save the updated file and repeat the [WooCommerce Import Process](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#WC_Import_PROC) as described above. After pressing the import button and letting WooCommerce do its import things, go check your shop. There should now be three products with images associated with them. Example shown below.



Great, we got our images showing up in the shop. Now let's tackle the image gallery.

**Simple WooCommerce CSV File: Image Gallery**

This is the last section where we'll being making the CSV by hand. This however is the most important section to our cause. Why?, you ask. Because, this will tell us what structure our CSV has to be in to make it work. With the final form of the CSV in hand, we will know how to target the R script to automate the process.

So if you've made it to this section, I'm assuming you understand

1. How to organize your images in folders by SKU.
2. How to upload the SKU folder and images to your website via FTP.
3. How to add a single image to your WooCommerce CSV import file.

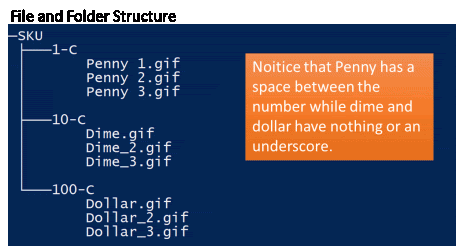
If you don't understand these concepts, read through those sections before continuing to this next section. Once we are through this, we'll have all the background we need to start building out the R code. This is much more scaleable, and will allow us to process lots of products and lots of images.

**Additional Images for the Gallery**

For each SKU in our simple 3 row CSV, here are a few additional images which we are going to use to generate an image gallery.

|  |  |  |  |
| --- | --- | --- | --- |
| **SKU** | **Image 1** | **Image 2** | **Image 3** |
| 100-C | Dollar Dollar.gif | Dollar Img 2 Dollar\_2.gif | Dollar Img 3 Dollar\_3.gif |
| 10-C | Dime Dime.gif | Dime Img 2 Dime\_2.gif | Dime Img 3 Dime\_3.gif |
| 1-C | Penny Penny 1.gif | Penny IMG 2 Penny 2.gif | Penny IMG 3 Penny 3.gif |

Download the images above and place each image in the appropriate SKU folder that you've been working in so far. When you are done, the file structure should look like this:



Upload the revised SKU folder to your website as described in the section on [Uploading your images with FTP](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#FTP_Upload). Do a spot check to make sure the images are accessible with your browser. Specifically try to view the files for the penny. For me, the URL would be something like

https://r-bar.net/wp-content/uploads/SKU/1-C/Penny 2.gif

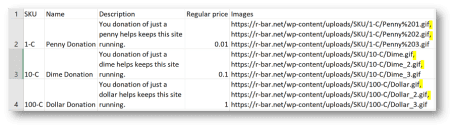
Now is a good time to mention that file names with spaces like “Penny 2.gif” are not HTML friendly. For this link to work correctly in WooCommerce, you need to encode the “ ” with %20 like so:

https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%202.gif

Great! Let's setup the CSV file.

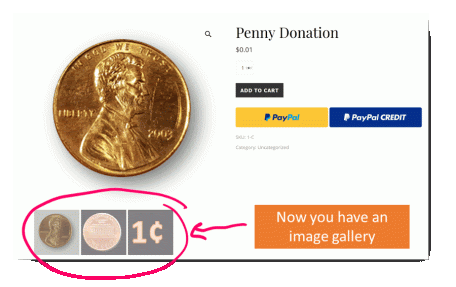
**Specifying the Image Gallery in the CSV**

If you gotten to this point, setting up the CSV for the image gallery is simple. All you do is list the images in the “Images” column. Each image should be separated by a comma: “, ”. Please note in some versions of WooCommerce, the separator between the images was a pipe symbol: “|”. So if one separator doesn't work, try the other. Below is a simple table illustrating how to list out the images in the CSV.



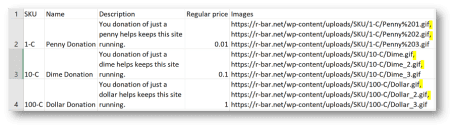
Once you have the image URLs listed out appropriately, follow the [WooCommerce import procedure](https://r-bar.net/woocommerce-image-gallery-tutorial-automate-r/#WC_Import_PROC) previously described.

If everything is loaded and there were no errors, navigate over to your shop with your browser. On the front page you will see the three images we saw when we loaded a single image. However, if we click on one of those products to see more details, an image gallery is now visible. Here is an example of what mine looks like for the penny:



**Generating a WooCommerce Image Gallery with R**

The previous section demonstrated the basics of putting together a WooCommerce image gallery via CSV bulk upload. We learned how to organize our images in folders by SKU number, how to upload those images via FTP, and how to specify those images in the CSV file. Now we're going to use all the information we've learned and automate building the WooCommerce image gallery CSV with R. The method, in this section is general. As long as the file structure is good, in principle you could do this on as many products as you like with varying image gallery size. For this example, though, we're going to keep with the 3 row test case we developed above. In which case, what we want R to do is build this table:



Before we dive into generating the table. I said, above that making the folder structure manually was error prone as well as tedious. Let's quickly see how to take care of this in R.

**Using R to Generate SKU Folders from File**

Here is a simple function to read the SKUs from your CSV and generate the folder structure in your target location. The function takes the following input arguments

**CSV\_File:** The WooCommerce Import CSV with a SKU column  
**Target\_Path:** The complete path to where you want to put the image folders

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # Make\_SKU\_Folder\_Function  Make\_SKU\_Folder <- **function**(CSV\_File, Target\_Path){  # Load File with SKU; NO IMAGES  WC\_CSV <- **read.csv**(**file**=CSV\_File)  #Call to dir.create to make all the SKU folders  **lapply**(WC\_CSV$SKU,  **function**(x){  **dir.create**(paste0(Target\_Path,x), recursive = **T**)  }  )  } |

**Example**  
Make\_SKU\_Folder(  
   CSV\_File = “WC-Simple-3-rows.csv”,  
   Target\_Path = “c:/Users/Mary/Desktop/WC-SETUP/SKU/”  
)

This should make the folder creation process quite a bit faster and accurate. Unfortunately, unless the file names are structured in a systematic way, you will still have the tedious task of copying the files to each individual SKU folder. The function above at least saves you from typos in the folder names.

**WooCommerce CSV R Function**

To automate the task of writing the WooCommerce image gallery to the CSV using R, we need the following items:

1. The path to the local folder structure with all of our images organized by SKU
2. A CSV containing the product SKU and any other additional information about the products.
3. The remote path where the images could be found with a web browser.
4. A separator used to delimit images in the image gallery (e.g., “,” or “|”, etc)

**Get The Local File and Folder List**

The first thing we are going to do have R determine the file and folder list. In order for this to work, R needs to know the local path to your WooCommerce images organized by SKU. In the example below, I've moved my SKU folder to C:/SKU. All of my images were gif, so I'll be searching those out specifically in case I have any PDFs or other file types that may also be in the folders.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | ###Get all the local images  Local\_PATH <- "C:/SKU/"  File\_Type <- "gif"  SKU\_Images <-  **data.frame**(  SKU\_Images=**list.files**(path=Local\_PATH,  pattern = File\_Type,  recursive = **T**  ),  stringsAsFactors = **F**  )    #Output  SKU\_Images |

Running this code, will give you a listing of the folder and file structure deeper than c:/SKU as shown in the table below:

| **SKU\_Images** |
| --- |
| 1-C/Penny 1.gif |
| 1-C/Penny 2.gif |
| 1-C/Penny 3.gif |
| 10-C/Dime.gif |
| 10-C/Dime\_2.gif |
| 10-C/Dime\_3.gif |
| 100-C/Dollar.gif |
| 100-C/Dollar\_2.gif |
| 100-C/Dollar\_3.gif |

**Remove Spaces in the File Name**

Sometimes image file names have spaces. Spaces are not HTML friendly characters. We need to convert or encode spaces to “%20”. For example, the file “Penny 1.gif” would need to be converted to “Penny%201.gif”. The code to do that on our image list is:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | ###If there is a space in the image title replace it with "%20"  SKU\_Images$SKU\_Images <-  **gsub**(pattern = " ", replacement = "%20", x = SKU\_Images$SKU\_Images)    #Output  SKU\_Images | |
| **SKU\_Images** | |
| 1-C/Penny%201.gif | |
| 1-C/Penny%202.gif | |
| 1-C/Penny%203.gif | |
| 10-C/Dime.gif | |
| 10-C/Dime\_2.gif | |
| 10-C/Dime\_3.gif | |
| 100-C/Dollar.gif | |
| 100-C/Dollar\_2.gif | |
| 100-C/Dollar\_3.gif | |

Now when we look at our image list, all the spaces have been replaced by “%20”. Awesome!

**Split into SKU and Image Columns**

The next thing we need to do is transform our image list into a data frame with two columns: “SKU” and “Images”. We can do that with tidyr's separate command as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | ##Split the SKU from the image  SKU\_Images <-  tidyr::separate(**data** = SKU\_Images, **col** = SKU\_Images,  into = **c**("SKU","Images"),  sep = "/" , extra="merge")    #Output  SKU\_Images | | |
| **SKU** | | **Images** |
| 1-C | | Penny%201.gif |
| 1-C | | Penny%202.gif |
| 1-C | | Penny%203.gif |
| 10-C | | Dime.gif |
| 10-C | | Dime\_2.gif |
| 10-C | | Dime\_3.gif |
| 100-C | | Dollar.gif |
| 100-C | | Dollar\_2.gif |
| 100-C | | Dollar\_3.gif |

**Inserting the Remote URL**

So, we have a data frame of SKU and Images. But, remember for the WooCommerce image gallery, we need to provide the full image URL to our wp-content/upload/ folder in the CSV. In order to take care of this, we need to provide R the “base URL”. That is the portion of the URL without the SKU number and the image name. To do this, we simply store the base URL as a string and paste it together with our images as shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8 | ###define the base url where your images can be found.  Base\_URL <- "https://r-bar.net/wp-content/uploads/SKU/"    ###Make the URL for each image  SKU\_Images$Images <- paste0(Base\_URL, SKU\_Images$SKU, "/", SKU\_Images$Images)    #Output  SKU\_Images | | |
| **SKU** | | **Images** |
| 1-C | | https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%201.gif |
| 1-C | | https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%202.gif |
| 1-C | | https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%203.gif |
| 10-C | | https://r-bar.net/wp-content/uploads/SKU/10-C/Dime.gif |
| 10-C | | https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_2.gif |
| 10-C | | https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_3.gif |
| 100-C | | https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar.gif |
| 100-C | | https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_2.gif |
| 100-C | | https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_3.gif |

If you've already uploaded your images to your web-server, you should copy-paste one of the URLs you've generated with R into your browser and make sure it works. If it does, you are doing great and we are almost done with the explanation of the parts of the script we need to make our function.

**One SKU many Images**

Right now, each of our image URLs is in a different line in our data frame and our SKUs appear multiple times. What we need is for each unique SKU to appear once and all the images to be listed beside that SKU in the Images column. We can achieve this in R using the aggregate function:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ###grouping by SKU, collapse the images by sku into a single comma separated list  SKU\_Images\_Final <-  **as.data.frame**(  **aggregate**(**list**(Images = SKU\_Images$Images),  **list**(SKU = SKU\_Images$SKU),  FUN = **paste**, collapse = ", "  )  )    SKU\_Images\_Final | |
| **SKU** | | **Images** |
| 1-C | | https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%201.gif, https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%202.gif, https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%203.gif |
| 10-C | | https://r-bar.net/wp-content/uploads/SKU/10-C/Dime.gif, https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_2.gif, https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_3.gif |
| 100-C | | https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar.gif, https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_2.gif, https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_3.gif |

Now our images are in a format acceptable to the WooCommerce Importer.

**Making the WooCommerce CSV**

Next, we need to combine the data frame we've been working on with our product database. Also, we don't want R to alter the column names in product database, so when we load it with the read.csv command we'll set the check.names argument to false. To combine the tables, we'll do a simple data merge. Our product database will be the left table and our data frame of images will be the right table. We want to keep the rows from our product database even if there are no associated images, so we'll be doing a left merge on “SKU” as shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | ### Load the Product Database (CSV File)  Product\_DB <- **read.csv**(**file**="WC-Simple-3-rows.csv", stringsAsFactors = **F**, check.names = **F**)    #Remove existing images column if there is one  Product\_DB$Images <- NULL    ###Merge your images back with the table.  WooCommerce\_TBL <-  **merge**(Product\_DB, SKU\_Images\_Final, **by** = "SKU", all.x = **T**)    #Output  WooCommerce\_TBL | | | | |
| **SKU** | | **Name** | **Description** | **Regular price** | **Images** |
| 1-C | | Penny Donation | You donation of just a penny helps keeps this site running. | 0.01 | https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%201.gif, https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%202.gif, https://r-bar.net/wp-content/uploads/SKU/1-C/Penny%203.gif |
| 10-C | | Dime Donation | You donation of just a dime helps keeps this site running. | 0.10 | https://r-bar.net/wp-content/uploads/SKU/10-C/Dime.gif, https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_2.gif, https://r-bar.net/wp-content/uploads/SKU/10-C/Dime\_3.gif |
| 100-C | | Dollar Donation | You donation of just a dollar helps keeps this site running. | 1.00 | https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar.gif, https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_2.gif, https://r-bar.net/wp-content/uploads/SKU/100-C/Dollar\_3.gif |

**Save the WooCommerce Table**

The last step, is to write the table to CSV so we can import it to WooCommerce. Below is the code to do just that. One thing that is slightly special about writing this CSV is the use of na=“” in the write.csv command. If you have no images for a product, the script above will place an NA in that row. WooCommerce will give an error if it sees “NA” in the Images column. To address this issue, we use na=“” in the write.csv command.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ### Write the New CSV File)  TheTime <- **format.Date**(**Sys.time**(), "%Y%m%d%H%M%S")  **write.csv**(WooCommerce\_TBL,  **file** = paste0(TheTime, "-WooCommerceTBL", ".csv"),  **quote** = **F**, **row.names** = **F**, na = ""  ) |

Great! We have all the pieces we need. Let's make a function.

**WooCommerce Image Gallery CSV | A Simple R Function**

So, from all the discussion above, we know the form of the target table, and we know the R code to get us there. In this section, we wrap all that up into a single R function. So here it is. The function takes the following input arguments:

InputCSV:

string, The path and file name of the WooCommerce Import CSV with a SKU column

LocalImagePath:

string, The complete path to the image folders

RemoteImagePath:

string, The HTTP path where the images can be found on your website

ImgType:

string, Describe the image file type (e.g., gif, jpg, tif, png …)

sep:

string, Separator for images in gallery, default is “, ” have seen “|” in other articles

OutputName:

string, suffix for optional CSV name

writeCSV:

boolean, if True writes a csv with OutputName suffix

**The Make\_WC\_IMG\_Gallery Function:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | Make\_WC\_IMG\_Gallery <-  **function**(InputCSV, LocalImagePath, RemoteImagePath, ImgType="gif", sep=", ",  OutputName = "-WooCommerceTBL", writeCSV = **T**){  Local\_PATH <- LocalImagePath  File\_Type <- ImgType  SKU\_Images <- **data.frame**(  SKU\_Images=**list.files**(path=Local\_PATH,  pattern = File\_Type,  recursive = **T**  ), stringsAsFactors = **F**)    SKU\_Images$SKU\_Images <-  **gsub**(pattern = " ", replacement = "%20", x = SKU\_Images$SKU\_Images)    ##Split the SKU from the image  SKU\_Images <-  tidyr::separate(**data** = SKU\_Images, **col** = SKU\_Images,  into = **c**("SKU","Images"),  sep = "/" , extra="merge")      ### Define the base url where your images can be found.  Base\_URL <- RemoteImagePath    ### Make the URL for each image  SKU\_Images$Images <- paste0(Base\_URL, SKU\_Images$SKU, "/", SKU\_Images$Images)    ### Grouping by SKU, collapse the images by sku into a single comma separated list  SKU\_Images\_Final <-  **as.data.frame**(  **aggregate**(**list**(Images = SKU\_Images$Images),  **list**(SKU = SKU\_Images$SKU),  FUN = **paste**, collapse = sep  )  )    ### Load the Product Database (CSV File)  Product\_DB <- **read.csv**(**file**=InputCSV, stringsAsFactors = **F**,  check.names = **F**)    ### Remove existing images column if there is one  Product\_DB$Images <- NULL    ###Merge your images back with the table.  WooCommerce\_TBL <-  **merge**(Product\_DB, SKU\_Images\_Final, **by** = "SKU", all.x = **T**)    ### Option to Write the New CSV File  **if**(writeCSV){  TheTime <- **format.Date**(**Sys.time**(), "%Y%m%d%H%M%S")  **write.csv**(WooCommerce\_TBL, **file** = paste0(TheTime, OutputName, ".csv"),  **quote** = **F**, **row.names** = **F**, na = "")  }    **return**(WooCommerce\_TBL)  } |

**Example**

**Make\_WC\_IMG\_Gallery(  
   InputCSV = “c:/0-R-Bar-WC-SETUP/WC-Simple-3-rows.csv”,  
   LocalImagePath = “C:/SKU/”,  
   RemoteImagePath = “https://r-bar.net/wp-content/uploads/SKU/”,  
   writeCSV = F  
)**

**Summary**

Wow, we covered a lot in this article. You learned how to manually setup a simple WooCommerce CSV with an image gallery. As well, we covered techniques to setup the image file structure and upload it to our website via FTP. Next, we used our experience of manually creating the WooCommerce image gallery to guide the development of scaleable process with R. If you want to see the fruits of this script working in action, head over to [r-bar.net](https://r-bar.net/shop/) to see the result. Best of luck setting up your own shop. I hope the info in this post saves you some time.