Web Scraping

Kevin DelRosso

December 9, 2015

Setting up the workspace

```
rm(list = ls())
setwd("~/Desktop/STA_141/assignment_6/")

require(scales) # for changing alpha level in plots
require(png)
require(grid)
require(magrittr) # for updating row names (while returning object)
require(lattice)
require(ggplot2)
require(XML)
require(RCurl)

save_plot = function(file_name, ht = 600, wd = 1000) {
    png(file_name, width = wd, height = ht, pointsize = 16)
    file_name
}
```

Part 1

The basic strategy was to use the *getNodeSet* function using the div tag with class 'question-summary' since it appears that every post contains this general tag. We then looped over each of these nodes and extracted all the specified information. We found that some users were not registered with stack overflow, thus they did not have a reputation score, user id, and their user name was contained in a slightly different piece of html. For this reason we first attempted to extract users / reputation using one method, and if no results were found attempted to extract just the user using a second method.

We attempted to use general xpath queries with the *starts-with* and *contains* functions, rather than exact matching to make the code more robust. We also ran the code over many pages to find strange cases and improve the code further. We checked the first several pages of results manually, and then checked the final output length (i.e. number of pages * 50 = number of results). Finally we found a small number of NA values in the final data frame indicating the web scraping was successful.

```
##########
# Part 1 #
##########

fix_numbers =
    # INPUT: character vector containing numbers
    # OUTPUT: numeric vector
    # DOC: fix 1,500 and 25.1k type numbers
function( char_vect )
{
    char_vect = gsub( ",", "", char_vect )

    # create multiplier vector, 1 and 1000 for k values
    mask_k = grepl( "k$", char_vect )
```

```
    mult = rep_len( 1, length( char_vect ) )
    mult[ mask_k ] = 1000

    char_vect = gsub( "k$", "", char_vect )

    as.numeric( char_vect ) * mult
}

# extract user id from, for example, /users/4788367/kukushkin
extract_userid = function( id ) gsub( "^/users/([[:digit:]]+)/.*$", "\\1", id )

get_user_reputation =
    # INPUT: html node containing 'user-details' div tag
    # OUTPUT: vector containing "user", "reputation", and "id"
function( node )
{
    # get user name
    xp = ".//div[@class = 'user-details']/a"
    user = xpathSApply( node, xp, xmlValue )

    # get repulation score
    rep_xp = ".//div[@class = 'user-details']//span[@class = 'reputation-score']"
    reputation = xpathSApply( node, rep_xp, xmlValue )
    reputation = fix_numbers( reputation )

    # get id number
    id = xpathSApply( node, xp, xmlGetAttr, 'href' )
    id = extract_userid( id )

    # in case we have guest poster, then the above will return character(0) and we'll
    # extract just the text, setting reputation and id to NA
    if( length(user) == 0 ) {
        user = xpathSApply( node, ".//div[@class = 'user-details']", xmlValue, trim = TRUE )
        reputation = NA
        id = NA
    }

    setNames( c( user, reputation, id ), c( "user", "reputation", "id" ) )
}

get_tags =
    # INPUT: html node
    # OUTPUT: all tag with '; ' separator
    # DOC: example output 'r; ggplot2; plot'
function( node )
{
    xp = ".//div[ contains(@class, 'tags') ]//a[@rel = 'tag']"
    tags = xpathSApply( node, xp, xmlValue )

    # just in case we don't find any (shouldn't be the case)
    if( length(tags) == 0 ) return( NA )

    paste( tags, collapse = "; " )
```

```
}

# when it was posted
get_date_time = function( doc ) {
    xp = "//div[@class = 'user-action-time']/span"
    as.POSIXct( xpathSApply( doc, xp, xmlGetAttr, 'title' ) )
}

# the title of the post
get_title = function( doc ) {
    xpathSApply( doc, "//a[@class = 'question-hyperlink']", xmlValue )
}

# the current number of views for the post
get_views = function( doc ) {
    views = xpathSApply( doc, "//div[ starts-with(@class, 'views ') ]", xmlGetAttr, 'title' )
    fix_numbers( gsub( " views", "", views ) )
}

# the current number of answers for the post
get_num_answers = function( doc ) {
    xp = "//div[ starts-with(@class, 'status ') and contains(@class, 'answered') ]/strong"
    as.numeric( xpathSApply( doc, xp, xmlValue ) )
}

# the vote "score" for the post
get_score = function( doc ) {
    score = xpathSApply( doc, "//span[@class = 'vote-count-post ']/strong", xmlValue )
    as.numeric( score )
}

# the URL for the page with the post, answers and comments
get_post_url = function( doc, base_url ) {
    post_url = xpathSApply( doc, "//a[@class = 'question-hyperlink']", xmlGetAttr, "href" )
    unname( getRelativeURL( post_url, base_url ) )
}

# the id (a number) uniquely identifying the post
get_unique_id = function( doc ) {
    id = xpathSApply( doc, "//div[@class = 'question-summary']", xmlGetAttr, "id" )
    as.numeric( gsub( "question-summary-", "", id ) )
}

# get the current page to add to df
get_current_page = function( doc ) {
    current_page = xpathSApply( doc, "//span[@class = 'page-numbers current']", xmlValue )
    as.numeric( current_page )
}


create_stackoverflow_df =
    # INPUT: a parsed html document and the url for the page
    # OUTPUT: a data frame with info for part 1 extracted
```

```
    # DOC: scapes the content from a single page
function( doc, base_url )
{
    posts = getNodeSet( doc, "//div[@class = 'question-summary']" )

    # user and reputation as a data.frame, convert column types to numeric
    user_rep = as.data.frame( t( sapply( posts, get_user_reputation ) ),
                              stringsAsFactors = FALSE )
    to_fix = c( "reputation", "id" )
    user_rep[ ,to_fix ] = sapply( user_rep[ ,to_fix ], as.numeric )

    tags = sapply( posts, get_tags )
    date_time = get_date_time( doc )
    title = get_title( doc )
    views = get_views( doc )
    num_answers = get_num_answers( doc )
    score = get_score( doc )
    post_url = get_post_url( doc, base_url )
    id = get_unique_id( doc )
    current_page = get_current_page( doc )

    data.frame( user = user_rep$user, date = date_time, title,
                reputation = user_rep$reputation, views, num_answers,
                score, post_url, id, tags, current_page,
                stringsAsFactors = FALSE )
}


reached_limit =
    # INPUT:
    #   - results: list of data frames
    #   - limit / limit_type from scrape_stackoverflow()
    #
    # OUTPUT: TRUE if we've reached the limit, FALSE otherwise
function( results, limit, limit_type )
{
    to_break = FALSE

    pages = length(results)
    if( pages == 0 ) return( to_break )

    posts = sum( sapply( results, nrow ) )

    # update to_break to TRUE is we've reached the limit
    if( limit_type == "pages" ) {
        if( pages >= limit ) {
            to_break = TRUE
        }
    } else {
        if( posts >= limit ) {
            to_break = TRUE
        }
    }
```

```
    to_break
}


get_filename =
    # INPUT: inputs from get_html_doc()
    # OUTPUT: a filename with path
    # DOC: different filenames whether or not post_id is specified
function( tag, page, post_id )
{
    filename =
    if( is.null( post_id ) ){
        sprintf( "~/Desktop/STA_141/assignment_6/saved_html/%s_%d.html", tag, page )
    } else {
        sprintf( "~/Desktop/STA_141/assignment_6/saved_pages/%s.html", post_id )
    }

    filename
}

get_html_doc =
    # INPUT: a url, tag, page number, and post_id
    # OUTPUT: a parsed html object
    # DOC:
    #   - load from disk if available, otherwise load url and save the html as
    #   tag_page.html ( if post_id = NULL ) or post_id.html otherwise
function( url, tag = NULL, page = NULL, post_id = NULL )
{
    already_saved = FALSE

    filename = get_filename( tag, page, post_id )

    # get html from file (if exists), or via url
    html =
    if( file.exists( filename ) ) {
        already_saved = TRUE
        filename
    } else {
        # to fix unicode error per piazza
        rawToChar( getURLContent( url, binary = TRUE, followlocation = TRUE ) )
    }

    doc = htmlParse( html )

    # in case we get redirected, update filename
    if( !is.null( post_id ) ) {
        question_id = get_question_id( doc )

        if( question_id != post_id )
            filename = get_filename( tag, page, question_id )
    }

    # if a new page, save to disk
```

```
    if( !already_saved ) saveXML( doc, filename )

    doc
}


scrape_stackoverflow =
    # INPUT:
    #   - a tag to search stackoverflow
    #   - either a page or posts limit
    #   - use limit_type = "pages" or "posts" to specify
    # OUTPUT: data frame with the specified number of results
    # DOC: calls create_stackoverflow_df() repeated to scrape all pages (up to limit)
function( tag = "r", limit = NULL, limit_type = "pages", print_progress = FALSE )
{
    # checking inputs, returns only the first page of results if not specified correctly
    if( !limit_type %in% c( "pages", "posts" ) ) {
        warn_message = "Please specify limit_type as either 'pages' or 'posts',
        only returning first page of results"
        warning( warn_message )
        limit = 1
        limit_type = "pages"
    }

    url = "http://stackoverflow.com/questions/tagged/%s?page=1&sort=newest&pagesize=50"
    url = sprintf( url, tag )

    # if limit is not specified, we'll process all the pages
    if( is.null( limit ) ) {
        limit = Inf
        limit_type = "pages"
    }

    results = list()
    page_count = 1
    while( TRUE ) {
        # keeping track of where we are
        if( print_progress ) print( url )

        # scrape the current page
        doc = get_html_doc( url, tag, page_count )
        results[[ page_count ]] = create_stackoverflow_df( doc, url )

        # get the next page url
        next_url = xpathSApply( doc, "//a[@rel = 'next']", xmlGetAttr, "href" )

        # there is no next page, i.e. we've reached the end
        if( length(url) == 0 ) break

        url = unname( getRelativeURL( next_url, url ) )

        # we've reached the maximum number of pages / posts
        if( reached_limit( results, limit, limit_type ) ) break
```

```
        page_count = page_count + 1
    }

    df = do.call( rbind, results )

    # return the exact specified limit for posts
    if( limit_type == "posts" ) df = head( df, limit )

    df
}

post_summaries = scrape_stackoverflow( 'r', 1, "pages", print_progress = FALSE )

if( FALSE ) save( post_summaries, file = "part_1_df.RData" )
```

Part 2

The overall strategy was to extract nodes for the question and each of its answers. Each may or may not contain comments, but if they do we'll loop over each comment and extract information from each as well. We found div tags with class 'question' and class starting-with 'answer' to identify the overall nodes, and used the div tag with id starting-with 'comment' to identify the comment nodes. The remaining information was extracted in a similar fashion to part 1, with the notable exception that we used the *saveXML* function to keep the content and html tags for each question/answer.

We had to overcome several issues with unusual pages. Several pages returned a 'Page Not Found' and an error, so we used the *try* function to get around this issue while issuing a warning message. We also had to add some code to handle the case when a question has been asked but has no answers. We tested the code on 250 pages without error, however it's very likely that there are other strange cases which we'll run into if we scrapes all the pages. For this reason, we saved all the html pages we scraped in anticipation of these potential unknown errors.

```
##########
# Part 2 #
##########

# extract post_id from url
get_post_id = function( url ) gsub( "^.*?/questions/([[:digit:]]+)/.*$", "\\1", url )

get_date =
    # INPUT: parsed html doc and an xpath query
    # OUTPUT: date as POSIXct class
    # DOC: date is either inside of <a> and <span>, or just inside <span>
function( doc, xpath )
{
    post_date = xpathApply( doc, xpath, xmlChildren )

    date = sapply( post_date, function(node) {
        # date is in either of these locations
        option_a = node[['a']][['span']]
        option_b = node[['span']]

        tmp =
        if( !is.null( option_a ) ) {
            option_a
```

```
        } else if( !is.null( option_b ) ) {
            option_b
        } else {
            return( NA )
        }

        xmlGetAttr( tmp, 'title' )

        } )

    as.POSIXct( date )
}

fix_entry_type =
    # converts asked -> question and answered -> answer
function( char_vect )
{
    char_vect = gsub( "asked", "question", char_vect )
    gsub( "answered", "answer", char_vect )
}

extract_from_post =
    # INPUT:
    #   - parsed html doc for a question or answer node
    #   - id of the post
    #   - id of the parent
    # OUTPUT: data frame of results for the post and it's comments
    # DOC: we filter out user who edit questions and answers
function( doc, id, parent_id = NA )
{
    # entry type
    xp_sig = ".//td[ starts-with(@class, 'post-signature') ]"
    xp_time = paste0( xp_sig, "//div[@class = 'user-action-time']" )
    asked_answered = xpathSApply( doc, xp_time, xmlValue, trim = TRUE )
    asked_answered = gsub( "^(asked|answered).*$", "\\1", asked_answered )

    # edited is also an option, we only want 'asked' or 'answered
    filter = asked_answered %in% c( "asked", "answered" )
    asked_answered = fix_entry_type( asked_answered[ filter ] )

    date = get_date( doc, xp_time )[ filter ]

    # get the user, repulation, and userid using part 1 function
    user_nodes = getNodeSet( doc, xp_sig )
    user_rep_id = lapply( user_nodes, get_user_reputation )[ filter ][[1]]

    # score / votes for the entry
    xp = ".//div[@class = 'vote']/span[@itemprop = 'upvoteCount']"
    votes = as.numeric( xpathSApply( doc, xp, xmlValue ) )

    # HTML content as string still containg html tags
    html_content = lapply( getNodeSet( doc, ".//div[@class = 'post-text']" ), saveXML )[[1]]
```

```
    qa_df = data.frame( entry_type = asked_answered,
                        user = user_rep_id[ "user" ],
                        userid = user_rep_id[ "id" ],
                        date = date,
                        reputation = user_rep_id[ "reputation" ],
                        votes, html_content, parent = parent_id, id,
                        stringsAsFactors = FALSE )

    comments_df = get_comments( doc, id )

    rbind( qa_df, comments_df )
}

get_comments =
    # INPUT: doc and parent_id from extract_from_post()
    # OUTPUT: data frame of results for comments
    # DOC: may contain many comments, loop over them and extract info for each
function( doc, parent_id )
{
    # get node for each comment
    xp = ".//div[ starts-with(@id, 'comment') ]//div[@class = 'comment-body']"
    comments = getNodeSet( doc, xp )

    # loop over each node and extract info, NA if doesn't exist
    comment_info = lapply( comments, function( node ) {

        html_content = xpathSApply( node, ".//span[@class = 'comment-copy']", xmlValue )

        xp = ".//a[ starts-with(@class, 'comment-user') ]"
        user = xpathSApply( node, xp, xmlValue )
        reputation = xpathSApply( node, xp, xmlGetAttr, 'title' )

        userid = xpathSApply( node, xp, xmlGetAttr, 'href' )
        userid = extract_userid( userid )

        # in case user doesn't have reputation as we've seen before
        if( length( user ) == 0 ) {
            user = NA
            reputation = NA
            userid = NA
        } else {
            reputation = gsub( " reputation", "", reputation )
        }

        xp = ".//span[ starts-with(@class, 'relativetime') ]"
        date = as.POSIXct( xpathSApply( node, xp, xmlGetAttr, 'title' ) )

        # first NA is for score since comments don't get votes
        # second NA overall post id
        data.frame( entry_type = "comment", user, userid, date, reputation,
                    votes = NA, html_content, parent = parent_id, id = NA,
                    stringsAsFactors = FALSE )
    } )
```

```
    do.call( rbind, comment_info )
}

# extract the question id from the question div tag
get_question_id = function( doc ) {
    xp = "//div[@id = 'mainbar']//div[@class = 'question']"
    as.numeric( xpathSApply( doc, xp, xmlGetAttr, 'data-questionid' ) )
}

scrape_posts =
    # INPUT: a url and optional argument to short text
    # OUTPUT: data frame of results
    # DOC: gets info for all question, answers, and comments
function( url, shorten_content = FALSE )
{
    doc = try( get_html_doc( url, post_id = get_post_id( url ) ) )

    # if we get an error loading the page (i.e. 'Page Not Found') issue warning with url
    # and return NULL
    if( class(doc)[[1]] == "try-error" ) {
        warm_message = paste( doc, url, sep = ": " )
        warning( warm_message )
        return( NULL )
    }

    # get question and comments df
    xp = "//div[@id = 'mainbar']//div[@class = 'question']"
    question_node = getNodeSet( doc, xp )[[1]]
    question_id = get_question_id( doc )
    question_df = extract_from_post( question_node, question_id )

    # get answers and comments df
    xp = "//div[@id = 'mainbar']//div[ starts-with(@class, 'answer') ]"
    answer_nodes = getNodeSet( doc, xp )
    answer_ids = as.numeric( xpathSApply( doc, xp, xmlGetAttr, 'data-answerid' ) )

    # in case page doesn't have any answers
    answers_df =
    if( length( answer_ids ) == 0 ) {
        NULL
    } else {
        answers = mapply( extract_from_post, answer_nodes, answer_ids, question_id,
                          SIMPLIFY = FALSE )
        do.call( rbind, answers )
    }

    df = rbind( question_df, answers_df )
    df$url = url
    rownames( df ) = NULL

    # for display purposes only, limits html_content character length to 20
    if( shorten_content ) {
        df$html_content = substring( df$html_content, 1, 20 )
```

```
        df$url = substring( df$url, 1, 20 )
    }

    df
}

# loop over urls from first three pages of summary results
summary_df = scrape_stackoverflow( 'r', 5, "pages", print_progress = FALSE )

df_QA = lapply( summary_df$post_url, function(url) scrape_posts( url, TRUE ) )
df_QA = do.call( rbind, df_QA )
```

Part 3

We'll use the provided data for part 3.

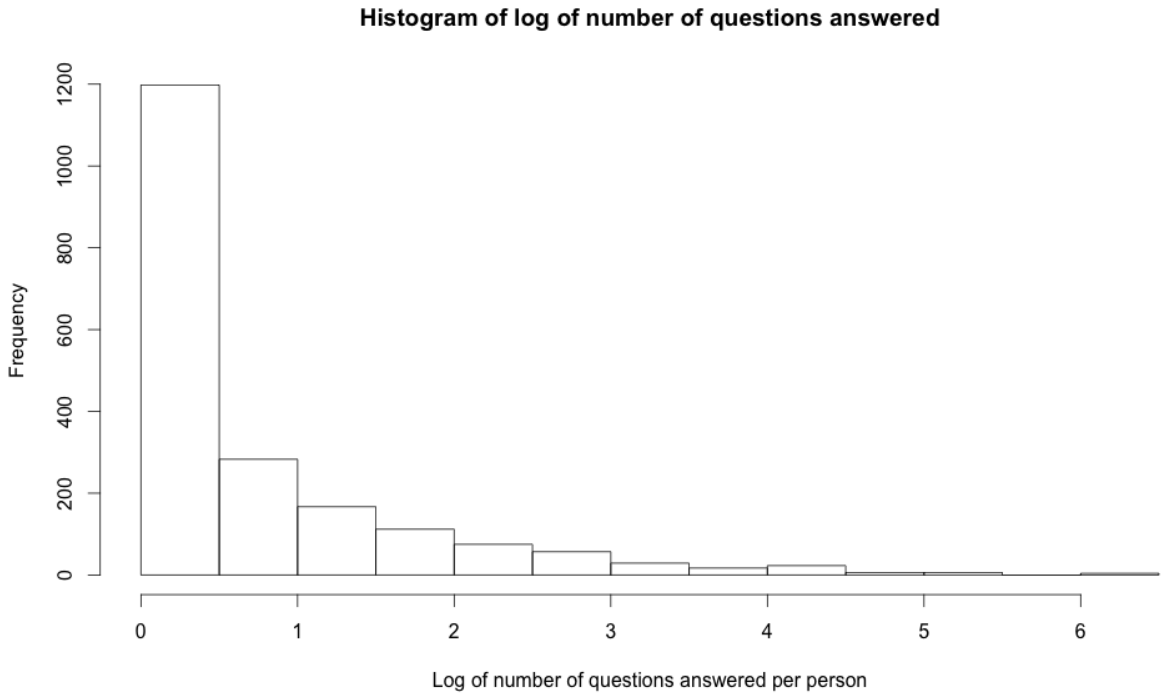
```
##########
# Part 3 #
##########

load( "./rQAs.rda" )

get_urls =
    # INPUT: unique urls from rQA rownames
    # OUTPUT: actual url
function( urls )
{
    base = "http://stackoverflow.com"
    urls = gsub( base, "", urls )
    urls = sapply( strsplit( urls, "\\." ), '[', 1 )
    paste0( base, urls )
}

fix_QA_df =
    # INPUT: data frame
    # OUTPUT: data frame with a url column and rownames reset
function( df, shorten_content = FALSE )
{
    df$url = get_urls( rownames( df ) )
    rownames( df ) = NULL

    if( shorten_content ) {
        df$text = substring( df$text, 1, 20 )
        df$url = substring( df$url, 1, 20 )
    }

    df
}

stack_df = fix_QA_df( rQAs )
```

1. What is the distribution of the number of questions each person answered?

The distribution of the number of questions each person answered is very right skewed since most people have only answered a few questions, while several people have answered hundreds. For this reason we chose

to plot a histogram using a log scale to show more details. In addition, we found that $\frac{1701}{1977} \approx 86\%$ answered at most 5 questions, while only 16 people answered more than 100 questions. The maximum answers for a single user was 642.

```
answered_df = subset( stack_df, stack_df$type == "answer" )

image = save_plot("./images/problem_3_1.png")

hist( log( table( answered_df$user ) ), xlab = "Log of number of questions answered per person",
      main = "Histogram of log of number of questions answered" )

invisible( dev.off() )
grid.raster( readPNG(image) )
```




```
max( table( answered_df$user ) )
```

```
## [1] 642
```

```
table( cut( table( answered_df$user ), breaks = c(0, 5, 10, 15, 20, 100, Inf) ) )
```

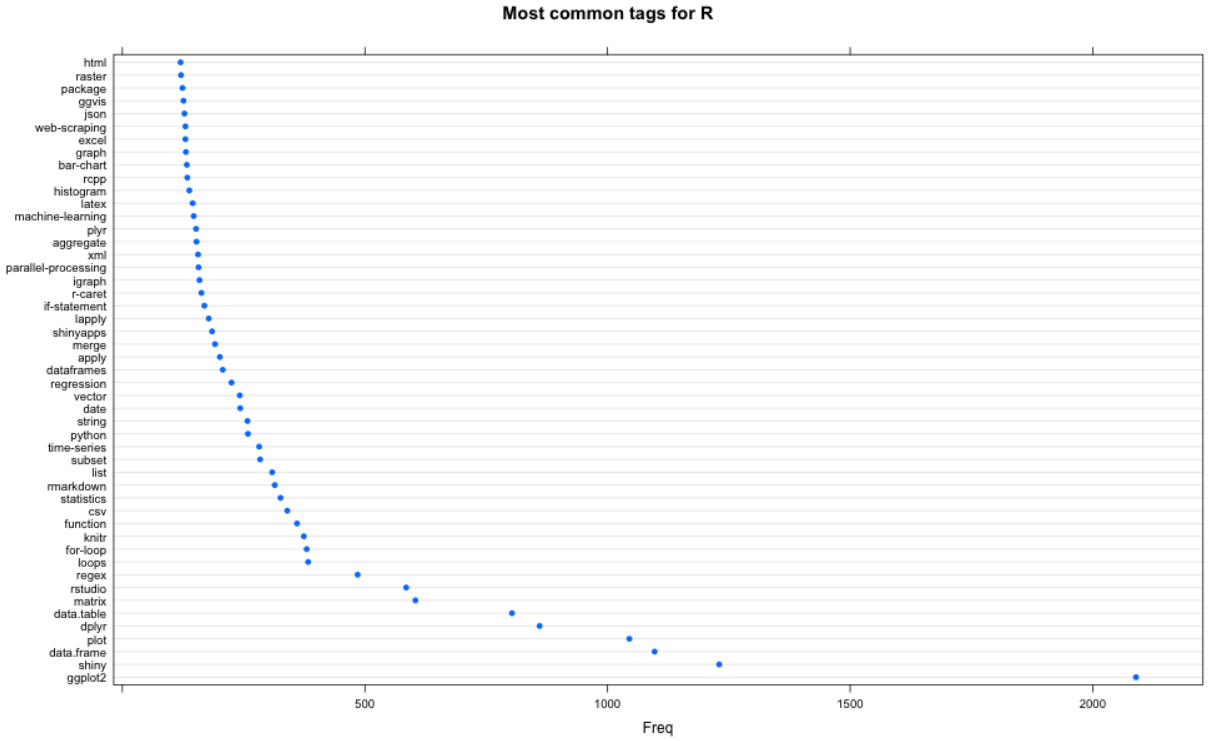
```
##
##     (0,5]    (5,10]   (10,15]   (15,20]  (20,100] (100,Inf]
##      1701       111        49        31        69        16
```

2. What are the most common tags?

We'll use the data frame we created in part 1, since it contains a column of tags. We chose to remove the 'r' tag since it was used to find these results, hence R will be a tag in each of the 25,000 posts. We found the

next 50 most common tags for R, which are summarized in the dot plot. The top-five tags are *ggplot2*, *shiny*, *data.frame*, *plot*, and *dplyr*.

```
# load the data frame from part 1
load( file = "part_1_df.RData" )

# split the tags by '; ' and unlist into a single vector
common_tags = table( unlist( strsplit( post_summaries$tags, "; " ) ) )

# r will always be the first one (since we searched for it), thus we'll remove it
common_tags = head( sort( common_tags, decreasing = TRUE ), 50 )[-1]

image = save_plot("./images/problem_3_2.png")

dotplot( common_tags, main = "Most common tags for R" )

invisible( dev.off() )
grid.raster( readPNG(image) )
```



3. How many questions are about ggplot?

We considered the question about ggplot if something related to ggplot is written in the question. We searched the post title (in the url) and text for *ggplot* and some common ggplot functions. We chose to only include functions whose name is most likely to be unique to ggplot: 'geom_', 'gg', and 'qplot'. We found 1018 posts about ggplot.

```
search_url_text =
    # INPUT:
    #   - df: data frame with columns for url (for title) and text
    #   - rx: a regular expression for finding matches
```

```
    #   - html_text: logical, whether or not to extract only text or
    #   use all html from text column
    # OUTPUT: data frame filtered to only include rows with a match
    # DOC: if either the url or text finds a match the row is returned
function( df, rx, html_text = FALSE )
{
    text = df$text
    # convert html to text contained within
    if( html_text ) {
        text = lapply( text, function(t) {
            html = htmlParse( t, asText = TRUE )
            txt = xpathSApply( html, "//p", xmlValue )
            link = xpathSApply( html, "//a", xmlValue )
            code = xpathSApply( html, "//code", xmlValue )
            unlist( c( txt, link, code ) )
        } )
    }

    # search url and text for term in the rx
    url_tf = grepl( rx, df$url, ignore.case = TRUE )
    text_tf = grepl( rx, text, ignore.case = TRUE )

    # keep TRUE results in either url or text
    mask = url_tf | text_tf

    df[ mask, ]
}

stack_df = fix_QA_df( rQAs )

# subset to only included questions
question_df = subset( stack_df, stack_df$type == "question" )

# get most common ggplot functions (also those function names only used by ggplot)
ggplot_functions = ls("package:ggplot2")

# ggplot functions and terms to search for
rx = "^(geom_|gg|qplot)"
ggplot_functions = ggplot_functions[ grepl( rx, ggplot_functions ) ]

# search url and text for ggplot and ggplot_functions
gg_rx = paste( ggplot_functions, collapse = "|" )
gg_df = search_url_text( question_df, gg_rx )

dim( gg_df )
```

```
## [1] 1018   11
```

4. How many questions involve XML, HTML or Web Scraping?

The initial strategy was to search post titles and text for terms related to the question: *xml*, *html*, *scraping*, and *xpath*. However, since the text actually contains html content we were finding too many false positives (i.e. a web page with .html extension that isn't actually a question about html). Thus, we added a function to extract just the text and code from the text column and search that.

We considered the question involving XML, HTML or Web Scraping if something related to those terms is written in the question, answer, or comments. We found all such matches and then found the unique count by question id to avoid over counting. We found 798 questions involving XML, HTML or Web Scraping.

```
search_terms = c( "xml", "html", "scraping", "scrape", "xpath" )

web_rx = paste( search_terms, collapse = "|" )
web_df = search_url_text( stack_df, web_rx, html_text = TRUE )

length( unique( web_df$qid ) )
```

```
## [1] 798
```

5. What are the names of the R functions referenced in the titles of the posts?

We'll use the data frame we created in part 1, since it contains a column for the title. The strategy used is a combination of the *exists* function and looking for words followed by parenthesis, i.e. function(). We removed common words from the workspace and then loaded all available packages to have the greatest chance of obtaining a match. We decided to exclude single characters from the search, in hopes of avoiding false positives. We found *library*, *function*, *UseMethod*, and *c* occurred the most frequently. We also found over a thousand unique functions from the 25,000 titles searched.

```
get_title_words =
    # INPUT: char vector of post titles
    # OUTPUT: char vector of function names
function( titles ) {
    # all unique words in all titles, remove single character words
    all_words = unique( unlist( strsplit( titles, "[[:space:]]" ) ) )
    all_words = all_words[ nchar(all_words) > 1 ]

    # looking for functions like rbind(), or rbind(list)
    mask = grepl( "[[:alpha:]][(]", all_words ) | grepl( "[(][)]$", all_words )
    funct_words = all_words[ mask ]

    # keep everything until the first '('
    funct_words = gsub( "(.*?)[(].*", "\\1", funct_words )

    # search for remaining words using exists
    all_words = all_words[ !mask ]
    are_functions = sapply( all_words, exists )

    # combine results and remove empty strings and 'R'
    funct_words = c( funct_words, all_words[ are_functions ] )

    funct_words[ !(funct_words %in% c( "", "R" )) ]
}


# clean-up workspace and load all packages I have available
# found here: http://www.r-bloggers.com/loading-all-installed-r-packages/
rm(list = c("rx", "answered", "image") )
invisible( lapply( .packages(all.available = TRUE), function(lib) {
    require( lib, character.only = TRUE)
    }) )
```

```
title_functions = get_title_words( post_summaries$title )

# markdown gives a different answer than normal R, so saving results
if( FALSE ) save( title_functions, file = "title_functions.RData" )

load( "title_functions.RData" )

head( sort( table( title_functions ), decreasing = TRUE ), 7 )
```

```
## title_functions
##   library  function UseMethod         c      plot       sum      eval
##        17         7         7         6         6         6         5
```

```
length( unique( title_functions ) )
```

```
## [1] 1856
```

6. What are the names of the R functions referenced in the accepted answers and comments of the posts?

We used two methods to find the accepted answers and comments (in order to avoid scraping thousands of pages). First we found questions which had more than three answers, increasing the chances of one of the answers being accepted. We then used the url to scrape those pages and extract the id of the accepted answer if it exists. We also looked at the scores and made the assumption that a score greater than 40 should be considered an accepted answer. This gave us more than 4,000 answers and comments to parse for R functions.

To get more reliable results than problem 5, we chose to only look for R functions within html code tags. With this strategy we'll likely miss some functions, but we can be very confident in the functions we do extract. To handle nested functions, we split on '(' and used regular expressions to extract the text before the original '(' location.

We found a total of 863 functions, fewer than problem 5 which is expected due to fewer false positives. In particular, nine functions: *library*, *function*, *list*, *lapply*, *unlist*, *dim*, *getURL*, *readHTMLTable*, and *c* were used with much greater frequency than the rest. These functions were very similar to the top functions found in problem 5. Finally we created a dot plot showing the frequency of the top 35 most used functions.

```
get_accepted_id =
    # INPUT: url
    # OUTPUT: id of accepted answer
    # DOC: return NULL if no accepted answer or Page Not Found
function( url )
{
    doc = try( get_html_doc( url, post_id = get_post_id( url ) ) )

    # some url return 'Page Not Found', return NA
    if( class(doc)[[1]] == "try-error" ) {
        return( NULL )
    }

    xp = "//div[@class = 'answer accepted-answer']"
    accepted_id = xpathSApply( doc, xp, xmlGetAttr, 'data-answerid' )

    # no accepted answer
```

```
    if( length( accepted_id ) == 0 ) return( NULL )

    accepted_id
}

get_accepted_df =
    # INPUT: data frame with only answers the entire data frame
    # OUTPUT: data frome with only accepted answers and comments
    # DOC: we'll also include ids with a score above min_score
function( df, min_score = 40 )
{
    answered_df = subset( df, df$type == "answer" )

    # find number of answers per question
    by_id = split( answered_df, answered_df$qid )
    num_answers = sapply( by_id, nrow )

    # scrape questions with more than 3 answers for an accepted answer
    ids_to_scrape = names( num_answers[ num_answers > 3 ] )
    accepted_df = df[ df$qid %in% ids_to_scrape, ]
    urls = unique( accepted_df$url )
    scraped_ids = unlist( sapply( urls, get_accepted_id ) )

    # include ids of answers with score greater than min_score
    more_ids = answered_df[ answered_df$score > min_score, ]$id

    # filter by the union of these two sets of ids
    accepted_ids = union( scraped_ids, more_ids )
    mask = (df$parent %in% accepted_ids) | (df$id %in% accepted_ids)

    df[ mask, ]
}

find_functions =
    # INPUT: text containing an html code tag
    # OUTPUT: character vector of functions found
function( text )
{
    doc = htmlParse( text )

    # get each line of code
    text = xpathSApply( doc, "//code", xmlValue )
    text = unlist( strsplit( text, "\n" ) )

    # remove comment lines
    text = text[ !grepl( "^ ?#", text ) ]

    # for dealing with nested function
    text = gsub( "(", ";;;(", text, fixed = TRUE )
    text = unlist( strsplit( text, "(", fixed = TRUE ) )

    # extract function with letters and [._-] only
    rx = "(^|.*[[:space:]])([[:alpha:]._-]+);;;$"
```

```
    mask = grepl( rx, text )

    gsub( rx, "\\2", text )[ mask ]
}

accepted_df = get_accepted_df( stack_df )
table( accepted_df$type )
```

```
##
##  answer comment
##    1357    3225
```

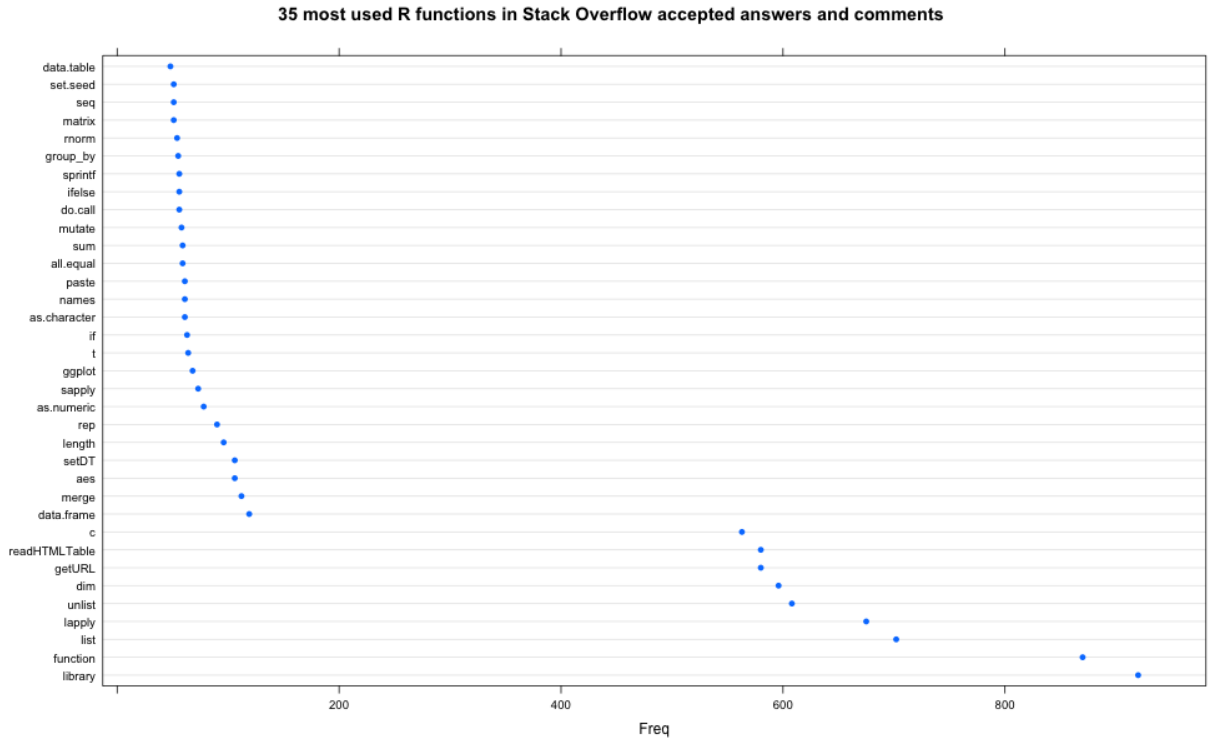
```
# look for functions inside code blocks, remove those without
text = accepted_df$text
text = text[ grepl( "<code>", text, fixed = TRUE ) ]

accepted_functions = unlist( lapply( text, find_functions ) )

results = sort( table( accepted_functions ), decreasing = TRUE )

head( results, 9 )
```

```
## accepted_functions
##       library      function          list        lapply        unlist
##           920           870           702           675           608
##           dim        getURL readHTMLTable             c
##           596           580           580           563
```

```
length( unique( accepted_functions ) )
```

```
## [1] 863
```

```
image = save_plot("./images/problem_3_6.png")

dotplot( head( results, 35 ),
         main = "35 most used R functions in Stack Overflow accepted answers and comments")

invisible( dev.off() )
grid.raster( readPNG(image) )
```


35 most used R functions in Stack Overflow accepted answers and comments
data.table
set.seed
seq
matrix
rnorm
group_by
sprintf
ifelse
do.call
mutate
sum
all.equal
paste
names
as.character
if
t
ggplot
sapply
as.numeric
rep
length
setDT
aes
merge
data.frame
c
readHTMLTable
getURL
dim
unlist
lapply
list
function
library
200
400
600
800
Freq