helpers.R

kored

2020-04-14

```
library(plyr)
library(data.table)
library(log4r)
```

```
## Warning: package 'log4r' was built under R version 3.6.3
```

```
##
## Attaching package: 'log4r'
```

```
## The following object is masked from 'package:base':
##
##     debug
```

```
library(tidyr)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.6.3
```

```
library(grid)

logger <- create.logger()
logfile(logger) <- 'debug.log'
level(logger) <- 'DEBUG'

# Purpose: Retrieve the necessary data
# Input: @meas - temp, previously - min_temp, max_temp, mean_temp
#       @month: all months of the year
#       @year_to_start:
# Output: output_df_all
getData <- function(meas, month, year_to_start){
  if(file.exists(paste('../RData/',meas,month, year_to_start,'.RData'))){
    load(paste('../RData/',meas,month, year_to_start,'.RData'), .GlobalEnv)
    debug(logger, paste("Rdata exists"))
  } else {
    debug(logger, paste("RData does not exists"))
    # ugly...
    min_input_df_all <- load_cleaned_data(year_to_start, month, 'min_temp') #data matrix X
    output_df_all <- regression(min_input_df_all) #reg results
    max_input_df_all <- load_cleaned_data(year_to_start, month, 'max_temp') #data matrix X
    output_df_all <- rbind(output_df_all, regression(max_input_df_all)) #reg results
    mean_input_df_all <- load_cleaned_data(year_to_start, month, 'mean_temp') #data matrix X
    output_df_all <- rbind(output_df_all,regression(mean_input_df_all)) #reg results
    input_df_all <- rbind(min_input_df_all, max_input_df_all, mean_input_df_all)
    save(output_df_all,input_df_all, file = paste('../RData/',meas,month, year_to_start,'.RData'))
    load(paste('../RData/',meas,month, year_to_start,'.RData'), .GlobalEnv)
```

```
  }
  return(output_df_all)
}

# Find data files
find_meas_data <- function(meas){
  # debug(logger, paste("|IM HERE 2|"))
  if(meas == 'min_temp'){
    txt_files_ls = list.files(path="../Data/Homog_monthly_min_temp_cleaned", pattern="*.txt", full.names
    names = list.files(path="../Data/Homog_monthly_min_temp_cleaned", pattern="*.txt")
  }
  else if(meas == 'max_temp'){
    txt_files_ls = list.files(path="../Data/Homog_monthly_max_temp_cleaned", pattern="*.txt", full.names
    names = list.files(path="../Data/Homog_monthly_max_temp_cleaned", pattern="*.txt")
  }
  else if(meas == 'mean_temp'){
    txt_files_ls = list.files(path="../Data/Homog_monthly_mean_temp_cleaned", pattern="*.txt", full.name
    names = list.files(path="../Data/Homog_monthly_mean_temp_cleaned", pattern="*.txt")
  }
  # else if(meas == 'precip'){
  #   debug(logger, paste("|IM HERE 3|"))
  #   txt_files_ls = list.files(path="../Data/Adj_monthly_total_prec_cleaned", pattern="*.txt", full.na
  #   names = list.files(path="../Data/Adj_monthly_total_prec_cleaned", pattern="*.txt")
  # }
  path_names <- list(txt_files_ls, names)
  # debug(logger, paste('|FIND TEMP DATA|'))
  return(path_names)
}

# Purpose: Load data from cleaning step
load_cleaned_data <- function(year_to_start = 1980, month = 'Feb', meas){
  # debug(logger, paste("|IM HERE 1|"))
  path_names <- find_meas_data(meas)
  txt_files_ls <- path_names[[1]]
  names <- path_names[[2]]
  ns = matrix(unlist(strsplit(names,'_',)),ncol = 3,byrow = TRUE)

  # build input data frame.
  input_df <- data.frame()
  debug(logger, paste('|BEFORE FOR LOOP|'))

  for (i in 1:length(txt_files_ls)){
      nom_city <- ns[i,2]
      nom_prov <- unlist(strsplit(ns[i,3],'.txt'))
      txt_files_df <- read.table(file = txt_files_ls[i], header = TRUE, sep = " ",dec = ".", colClasses

      years_greater<-txt_files_df[as.numeric(as.character(txt_files_df$Year))>=year_to_start,]

      y_temp <- suppressWarnings(as.numeric(as.character(unlist(years_greater[,month]))))

      x_year <- suppressWarnings(as.numeric(as.character(unlist(years_greater[,'Year']))))
      # debug(logger, paste('|START YEAR|', year_to_start, '|'))
      temp_df <- data.frame(y_temp, x_year, "city" = nom_city, "prov" = nom_prov, 'meas_name' = meas)
```

```
      # debug(logger, paste('|LOAD CLEANED DATA|', 6, '|'))
      input_df <- rbind(input_df, temp_df)
  }
  return(input_df)
}

############################################################
# Purpose: Perform Regression
# Input: years, temp, city, prov
# Output: statistical data
############################################################
regression <- function(input_df){
  city_prov_vector <- unique(input_df[,c("city", 'prov')])
  city_vector <- city_prov_vector[, 'city']
  prov_vector <- city_prov_vector[, 'prov']
  meas <- unique(input_df$meas_name)

  output_df <- data.frame()
  for (i in 1:length(city_vector)){
    index <- which(input_df[, "city"] == city_vector[i])
    # if(numVar == 1)
    fit <- lm(y_temp[index]~x_year[index], data = input_df)
    # else if(numVar == 2)
      # fit <- lm(y_meas.x[index]~y_meas.y[index], data = input_df)

    b <- data.frame("intercept" = fit$coefficients[1], "slope" = fit$coefficients[2])
    R_2 <- data.frame("r.squared" = as.numeric(unlist(summary(fit)$r.squared)))
    # CIs <- ci(fit, 0.95, alpha=1-0.95, na.rm = TRUE)
    critical_value <- qt((1-0.95)/2, (nrow(fit$model)-1))
    standard_error <- summary(fit)$coef[,2][2]
    margin_error <- critical_value*standard_error
    estimate <- summary(fit)$coef[,1][2]
    CI_lower <-  estimate + margin_error
    CI_upper <- estimate - margin_error
    variance <- (standard_error)^2


    curr_results_df <- data.frame("city"=city_vector[i],'prov' = prov_vector[i],
                                  b,"r.squared"=R_2,CI_lower, CI_upper,variance,
                                  "n"=nrow(fit$model), 'meas_name' = meas,  row.names = NULL)
    output_df <- rbind(output_df,curr_results_df)
  }
  return(output_df)
}

# Interaction Model - Confirm Regression Results
# city<- data.table(city_vector, stringsAsFactors = TRUE)
# fit_2 <- lm(y_temp~ city-1 + city*x_year , data = input_df)

# Draw plots
# df_consts <- data.frame(year_to_start <- '1980',
# plot_type <- 'regression line',
# location <- 'TORONTO,ON',
```

```
# region <- 'Province',
# stat<- 'Slopes',
# meas<-'min_max_temp',
# month <-'Feb',
# city <- 'TOROMTO',
# prov <- 'ON',
# city_lab <-'Enable')



#########################################################################
# Purpose: Modularize plotting - Mediator for other plot function
# Input: meas, month, dataframe containing more variables
# output: grid drawn on UI, and grob object
#########################################################################
setup_plots <- function(meas, month, df_consts){
  year_to_start <- df_consts$year_to_start
  plot_type <- df_consts$plot_type
  location <- df_consts$location
  region <- df_consts$region
  stat<- df_consts$statistic
  city <- df_consts$city
  prov <- df_consts$prov
  city_lab <- df_consts$prov
  # print(city)
  # print(prov)
  # debug(logger, paste('-----------df_consts ----------',df_consts ))

  output_df_all <- getData('temp', month, year_to_start)
  if(region == 'Province'){
    index <- which(output_df_all[, "prov"] == location)
    output_df_all <- output_df_all[index,]
  }
  else if(region == 'City'){
    # city <- strsplit(location, ',')[[1]][1]
    # prov <- strsplit(location, ',')[[1]][2]
    index <- which(input_df_all$prov==prov
                                    & input_df_all$city == city)
    output_df_all <- input_df_all[index,] # chnage name.. .
  }

  p<-add_plot_data(meas, output_df_all) # returns a list plot(s)
  p<-add_plot_type(p, df_consts) #constructs plot(s)
  # print(p)
  p<- create_grid(p,month, df_consts)
  suppressMessages( grid.draw(p))
  invisible(p)
}


add_plot_data <- function(meas, output_df_all){

  if(meas == 'min_max_temp'){
```

```
    min_output_df_all <- output_df_all[which(output_df_all$meas_name=='min_temp'),]
    max_output_df_all <- output_df_all[which(output_df_all$meas_name=='max_temp'),]
    p1<-ggplot(min_output_df_all)
    p2<-ggplot(max_output_df_all)

    return(list(p1,p2))
  }
  else if(meas == 'mean_temp'){
    mean_output_df_all <- output_df_all[which(output_df_all$meas_name=='mean_temp'),]
    p<-ggplot(mean_output_df_all)
    return(list(p))
  }
}



##################################################
# Purpose: setup plot types, such as histogram or boxplot
##################################################
add_plot_type<- function(curr_plots, df_consts){
  plot_type <- df_consts$plot_type
  region <- df_consts$region
  stat <- df_consts$stat
  city <- df_consts$city
  prov <- df_consts$prov
  city_lab <- df_consts$city_lab

  stat_lab <-bquote(.(stat)*' ('*degree *'C)')

  for(i in 1: length(curr_plots)){
    dat <- curr_plots[[i]]$data
    index <- which(dat$prov==prov& dat$city == city)
    dat_city <- dat[index,]
    if(plot_type == 'histogram'){
      aes <- aes(x = slope)
      # aes_vline<- aes(xintercept=mean(slope))
      x_city <- dat_city$slope
      x_dat <- mean(dat$slope)
      if(strsplit(stat, ' ')[[1]][1] == 'R-squared'){
        stat_lab <-bquote(.(stat)*' (%)')
        aes <- aes(x = r.squared)
        # aes_vline<-aes(xintercept=mean(r.squared))
        x_city <- dat_city$r.squared
        x_dat <- mean(dat$r.squared)
      }
      else if(strsplit(stat, ' ')[[1]][1] == 'CI_lower'){
        aes <- aes(x = CI_lower)
        # aes_vline<-aes(xintercept=mean(CI_lower))
        x_city <- dat_city$CI_lower
        x_dat <- mean(dat$CI_lower)
      }
      else if(strsplit(stat, ' ')[[1]][1] == 'CI_upper'){
        aes <- aes(x = CI_upper);
```

```
      # aes_vline<-aes(xintercept=mean(CI_upper))
      x_city <- dat_city$CI_upper
      x_dat <- mean(dat$CI_upper)
    }

    curr_plots[[i]] <- curr_plots[[i]]+ aes +
      geom_histogram(aes(y=..density..), colour="black", fill="white")+
      geom_density(alpha=.05, fill="#FF6666") +
      # geom_vline(aes_vline,color="blue", linetype="dashed", size=1)+
      labs(y='Frequency', x = stat_lab)

    if(city_lab == 'Enable')
      curr_plots[[i]] <- curr_plots[[i]] +
      geom_point(x = x_city ,y = 5, colour = 'purple')+
      annotate("text", x = x_dat, y = 10  , vjust = 1, hjust = 1,
               label = str_replace(city," ","_"), parse = TRUE, colour= 'purple')

  }
  else if(plot_type == 'boxplot'){
    aes <- aes(x=prov, y=slope) #For slope..

    if(strsplit(stat, ' ')[[1]][1] == 'R-squared')
      aes <- aes(x=prov, y=r.squared);stat_lab <-bquote(.(stat)*' (%)')

    curr_plots[[i]] <- curr_plots[[i]] + aes +
      geom_boxplot() +
      stat_summary(fun.y=mean, geom="point", shape=23, size=4)+
      stat_boxplot(geom = 'errorbar')+
      labs(y=stat_lab, x = 'Province')
  }
  else if(plot_type == 'regression line'){
    dat <- curr_plots[[i]]$data
    aes <- suppressMessages(aes(x = x_year, y = y_temp))
    fit <- suppressMessages(lm(y_temp~x_year, data = dat))
    R_2 <- as.numeric(unlist(summary(fit)$r.squared))
    # print(R_2)
    # critical_value <- qt((1-0.95)/2, (nrow(fit$model)-1))
    # standard_error <- summary(fit)$coef[,2][2]
    # margin_error <- critical_value*standard_error
    # estimate <- summary(fit)$coef[,1][2]
    # CI_lower <-  estimate + margin_error
    # CI_upper <- estimate - margin_error
    curr_plots[[i]] <- curr_plots[[i]]+ aes +
      geom_point(size = 1)+
      stat_smooth(method = 'lm', se = FALSE)+
      labs(y=stat_lab, x = 'Years')+
      scale_x_continuous(breaks  = seq(1980,2020, by = 5))

    curr_plots[[i]] <- curr_plots[[i]] +
      ggpubr::stat_regline_equation(label.x.npc = 'left', label.y.npc='bottom', colour= 'purple')+
      annotate("text", x = 1985, y = -Inf  , vjust = -0.5,
               label = paste('R^2 == ', signif(R_2,2)), parse = TRUE, colour= 'purple')
```

```
    }
  }
  return(curr_plots)
}

########################################
# Purpose: adding plot labels
# Input: curr_plot, title, meas, stat
# Output:
########################################
add_plot_labels <- function(curr_plot, title_meas, stat){
  title  = bquote(.(title_meas)*' - '~.(stat))
  if (stat == "Temperatures vs Years")
    title  = bquote(.(title_meas) *' vs Years')
  curr_plot <- curr_plot +
    ggtitle(title)+
    theme(plot.title = element_text(hjust = 0.5, size = 10),
          axis.title.x = element_text(size = 9),
          axis.title.y = element_text(size = 9))
}

#############################################################################
# Purpose: creating grid for displaying both min and max plots on same panel
#############################################################################
create_grid <-function(curr_plot, month, df_consts){
  year_to_start <- toString(df_consts$year_to_start)
  location <- df_consts$location
  stat <- df_consts$stat
  region <- df_consts$region
  print(location)
  month <- toString(month)
  year_end <- switch(region, 'City' = max(curr_plot[[1]]$data$x_year), {'2017'})
  subt<- bquote(italic(.(location)*' -'~.(month) *' - ('* .(year_to_start)*' - '* .(year_end) *')'))

  # if its Min_max_temp
  if (length(curr_plot) == 2){
    curr_plot[[1]] <- add_plot_labels(curr_plot[[1]], 'Minimum Temperatures', stat)
    curr_plot[[2]]<- add_plot_labels(curr_plot[[2]], 'Maximum Temperatures', stat)
    title = bquote('Min. vs Max. Temperatures - '~.(stat))

    if(stat == 'Temperatures vs Years')
      title = bquote('Min. vs Max. Temperatures')

    p<- arrangeGrob(
      top = textGrob(title, gp=gpar(fontface="bold")),
      sub = textGrob(subt, gp = gpar(col = 'red', fontface='italic',
                                     fontsize = 11 )),
      curr_plot[[1]],
      curr_plot[[2]],
      bottom = textGrob(
        "Source: Environment Canada Temperature Data - 2017",
        gp = gpar(fontface = 3, fontsize = 9),
        hjust = 1,
```

```
        x = 1
      ),
      ncol = 1,
      heights=c(0.05, 0.5, 0.55)
    )
  } # add caption...
  else if(length(curr_plot) == 1){
    title = bquote('Mean Temperatures - '~.(stat))
    if (stat == "Temperatures vs Years")
      title = 'Mean Temperatures vs Years'

    p<- curr_plot[[1]]+
      ggtitle(title)+
      labs(subtitle = subt)+
      theme(plot.title = element_text(hjust = 0.5),
            plot.subtitle = element_text(hjust = 0.5, color = 'red' ))
  }
  return(p)
}

#####################################################
# Purpose: get names of city for a given province
# input: prov
# output: vector with names of city
#####################################################
get_city_vector <- function(prov){
  require(plyr)
  if(file.exists(paste('../RData/','constant_values','.RData'))){
    load(paste('../RData/','constant_values','.RData'), .GlobalEnv)
    city_vector <- city_prov_vector[which(city_prov_vector$prov==prov), ]
    city_vector <- select(city_vector, city)
    # city_vector <- data.frame(city_vector[, 'city'])
    city_vector$city <- as.character(city_vector$city)
    city_v <- sort(city_vector$city)
    return(city_v)
  }
}

get_city_stats<- function(city, month, year_to_start){
  output_df_all <- getData('temp', month, year_to_start)
  mean_output_df_all <- output_df_all[which(output_df_all$meas_name=='mean_temp'),]
  mean_stats <- mean_output_df_all[which(mean_output_df_all$city==city),]
  return(mean_stats)
}

#####################################################
#####################################################
# The following functions are not needed for the app
# clean_data
# check_start_year_cutoff
# get_prov_vector(
# map
#####################################################
```

```
#########################################################
# # Purpose: Data cleaning step
# # Input: @var, @dir
# # Output: No return; writes to file
# # Not needed for app...
# clean_data <- function(var, dir){
#   for (i in 1:length(var)){
#     df = read.delim(var[i], skip = 0, header = FALSE, as.is=TRUE, dec=".", sep = ",", na.strings=c("
#     stationNum_city_prov <- paste(select(df, V1)[1,1], trimws(select(df, V2)[1,1]), province <- selec
#     #forward slash for precipatation files - "7025250_MONTREAL/PIERRE ELLIOTT T_QC"
#     stationNum_city_prov<- str_replace_all(stationNum_city_prov, "/",'-')
#     seq(from = 3, to = 35, by = 2)
#
#     df <- select(df, -seq(from = 3, to = 35, by = 2))
#     data <- slice(df, 5:n())
#     (hdr <- slice(df, 3))
#     is.na(hdr)
#
#     df <- plyr::rename(data, hdr)
#     #filter out -9999.9 - default values
#     df <- data.frame(lapply(df, function(x){
#       gsub("-9999.9", "NA", x)
#     }))
#
#     filePath= sprintf("%s_cleaned/%s.txt",dir,stationNum_city_prov)
#     write.table(df, filePath, append = FALSE, sep = " ", dec = ".",
#                 row.names = FALSE, col.names = TRUE)
#   }
# }
# check_start_year_cutoff <- function(meas){
#   temp_object <- find_meas_data(meas)
#   txt_files_ls <- temp_object[[1]]
#   names <- temp_object[[2]]
#   most_recent_year <-c()
#   for (i in 1:length(txt_files_ls)){
#     txt_files_df <- read.table(file = txt_files_ls[i], header = TRUE, sep = " ",dec = ".", colClasses
#     x_temp <- suppressWarnings(as.numeric(as.character(unlist(txt_files_df[,'Year']))))
#     most_recent_year[i] <- max(x_temp)
#   }
#   most_recent_year <- most_recent_year[!is.na(most_recent_year)]
#   # debug(logger, paste('|most_recent_year ' , '|', most_recent_year,"|"))
#   start_year_cutoff <- min(most_recent_year, na.rm = TRUE)
#   # debug(logger, paste('|min ' , '|',start_year_cutoff,"|"))
#   return(start_year_cutoff)
# }

# get_prov_vector <- function(meas, month, year_to_start){
#   if(file.exists(paste('/RData/','constant_values','.RData', sep=''))){
#     load(paste('/RData/','constant_values','.RData', sep=''), .GlobalEnv)
#     prov_vector <- unique(city_prov_vector[, 'prov'])
#     prov_vector <- prov_vector[ , order(names(prov_vector))]
#     return(prov_vector)
#   }
```

```
# }

# library(sfb)
# library(raster)
# map <- function(){
#   # can0 <- getData("GADM",country="CAN",level=0)
#   provinces <- c("Ontario")
#   # can1 <- getData('GADM', country="CAN", level=1)
#   # can1 <- readRDS("gadm36_CAN_1_sp.rds")
#   # ca.provinces <- can1[can1$NAME_1 %in% provinces,]
#   # can2<-getData('GADM', country="CAN", level=2) # counties
#   can2 <- readRDS("C:/Environment_Canada_Shiny_App/gadm36_CAN_2_sp.rds")
#   ca.cities <- can2[can2$NAME_1 %in% provinces,]
#   prov <- 'ON'
#   prov_df_city_slope <- output_df_all[ which(output_df_all$prov==prov),]
#   prov_df_city_slope <- prov_df_city_slope[,c('city', 'slope')]
#   prov_df_city_slope$city <- str_to_title(prov_df_city_slope$city)
#
#   munic_div<- read.csv('C:/Environment_Canada_Shiny_App/Data/mmah-list-of-ontario-municipalities-en-u
#   munic_div$Municipality <- gsub("<.*?>","",as.character(munic_div$Municipality))
#   temp <- munic_div$Municipality
#   temp <- gsub(",.*", "", temp)
#   munic_div$Municipality <- temp
#   prov_df_temp <- merge(munic_div, prov_df_city_slope, by.x ='Municipality', by.y ="city")
#
#   ca.cities@data$id <- rownames(ca.cities@data)
#
#   prov_df <- merge(ca.cities@data,prov_df_temp , by.x = 'NAME_2', by.y = 'Geographic.area')
#
#   check<-st_as_sf(ca.cities)
#
#   check5 <- merge(check, munic_div, by.x = 'NAME_2', by.y = 'Geographic.area')
#   check7 <- merge(check5, prov_df_city_slope, by.x = 'Municipality', by.y = 'city')
#
#   copy_check7 <- check7
#   st_geometry(copy_check7) <- NULL
#   unique_copy_check7_slope <- data.frame(unique(copy_check7$NAME_2))
#   new_new <-data.frame()
#   for(i in 1: nrow(unique_copy_check7_slope)){
#     name <- unique_copy_check7_slope[i,]
#     index <- copy_check7[which(copy_check7$NAME_2 == name),]
#     mean_slope <- mean(index$slope)
#     index$slope <- mean_slope
#     new_new <- rbind(new_new, index)
#   }
#   check7$slope <-new_new$slope
#
#     gg<- ggplot(data = check7)+
#       geom_sf(aes(fill= slope))+
#       scale_fill_gradient(name = 'Trends',
#                           low = "blue", high = "gold2")
#      gg+ geom_path(data= ca.cities, mapping = aes(x=long, y =lat, group = group))
#     # return(base_sp)
```

```
# }
#
```