Location Based Analysis on Github Repositories

Yanwei Song

LakshmiManaswitha chimakurthi

4/27/2017

Hypothesis

The Usage of programming languages in GitHub differs by the developer's location.

Abstract

GitHub is a popular platform for collaboration on open source projects. It also provides a rich API to query various aspects of the public activity. This combination of a popular social coding website with a rich API presents an opportunity for researchers to gather empirical data about software development practices. There are an overwhelmingly large number of competing platforms to choose from in software development. Knowing the usage of the top programming languages can assist the developers of that particular location

who are trying to increase their employ-ability, as well as software engineers deciding which technology to use in their next big project. In our analysis we will find out the usage of top GitHub programming languages in different countries.

Introduction

GitHub is the most widely used social code hosting platform, based on Git, a distributed version control system. It introduces a social at to software development where users can browse, fork and even contribute to the projects created and maintained by others. Such platform facilitates agile development and has the potential to address problems such as collaboration, communication, and code conflicts. Our analysis is to find out how the contributions in the GitHub's top programming languages varies in different countries. For our analysis we restrict to the top most 10 languages used on GitHub and the top most 10 active developer locations. The main aim of the analysis is to show how the usage of these languages vary in different developer locations.

Key documents

This project is set up on `cleaned-projects.csv` file, which comes from GHTorrent (http://ghtorrent.org). The GHTorrent provides a snapshot of GitHub metadata which is constructed by listening to the public GitHub events. It provides a simple way to get urls of almost all GitHub projects. There are three variables in `cleaned-projects.csv` file: relative url of the project, language of the projects, and if the projects is fork (`0` for not fork projects, `1` for forked projects). Since the GHTorrent can only listen to ''public' events from GitHub, it does not contain the latest information and some projects listed in GHTorrent may not been made private, or deleted without GHTorrent noticing them.

The second document this report references is GitHub API (https://developer.github.com/v3/). The method of scraping GitHub user's locations and repository's information comes from it.

The third document in the report is GoogleAPI (https://developers.google.com/maps/documentation/geocoding/start). We learned how to use unformulated locations to scrape the country names.

Methods

1. Data sampling and collecting

There are 200,000 repositories sampled from `cleaned-projects.csv` in total. GitHub data will been scraped from GitHub API. Country name will been scraped by using Google API based on the GitHub user's location. There are seven steps in total to prepare data.

Since the `feather` library useful to store intermediate results in a quick to load format, and the data file is big to load each time. So we will write our data into feather files each time when we get the data sets.

1. Data Loading

Since the loading time of `cleaned_projects.csv` is too long for R, so I will use `fread` function from the `data.table` package to read csv file, then write it as a feather file and load it.

```
# this step only need when I first load data set.
df <- fread("cleaned_projects.csv")
# load feather file
# this two lines only used in first time load feather tibble
colnames(df) <- c("url", "p.language", "fork")
# write as a feather file
write_feather(df, "cleaned_projects.feather")
```

2. Data Filtering and Cleaning

Since \\ is the disturbance information in url, so it will been removed from our data frame.

```
# clean: data.frame -> data.frame
#
# Purpose: takes in the data frama, removes the fork equal 1 in fork column, "\\" in
url column from the data set.
clean <- function(mydf){
  # collect not fork data from the data set
  mydf %<>%
    filter(fork == 0) %>%
    collect(n = Inf)
  mydf$url <- str_replace_all(mydf$url, "[\\\\]", "")
  return(mydf)
}
df.c <- clean(df)
# write in feather file
write_feather(df.c, "cleanedData.feather")
#df.c <- read_feather("cleanedData.feather")
```

3. Data Sampling

There are 50,000 repositories sampled from cleaned data frame, user name and repository name will be included in sampled data set.

```r
# sample.rep: data.frame, number -> data.frame
#
# Purpose: sample repositories from cleaned data frame, number here is the number we
prepare to sample
sample.rep <- function(mydf, n){
  # randomly pick out 10,000 rows
  df.new <- mydf %>%
    sample_n(n)
  # split the url to two part: owner, and repo
  repo.url <- data.frame(str_split_fixed(df.new$url, "/", 2))
  colnames(repo.url) <- c("owner", "repo")
  cbind(df.new, repo.url)
}

# 1st 50,000 sample
df.rep <- sample.rep(df.c, 50000)
# write as a feather file
write_feather(df.rep, "50000sample.feather")
```

4. GitHub Data Acquisition

There are two parts here, acquiring the user's location, project's information from GitHub API. We would like to scrape the user's locations based on the user names, collect the number of forks, watchers, stargazers, languages, and create time based on the user name and repository name.

Since the GitHub API limitation is 5,000 requests per hour with token, the token and system sleep time (0.73s) will be set before data scraping.

Collecting user's location

```
# setting github token
Sys.setenv(GITHUB_PAT = "Token obtained from GitHub account API")
# get_user: character -> data.frame
#
# Purpose: collect user's location based on user's information
get_user <- function(url){
  rep.v <- str_split_fixed(url, "/", 2)
  # acquire github user's information
  tryCatch({
    # wait 0.73s to complete the api rate limitation 5000/h
    Sys.sleep(0.73)
    user_i <- gh("GET /users/:username", username = rep.v[,1], .limit = 20000)["locat
ion"]
    message("OK >> ", url)
    # identify if location is null
    if(is.null(user_i$location)){
      user_i <- data.frame(url = url, location = NA)
      return(user_i)
    } else {
      user_i <- data.frame(url = url, location = user_i$location)
      return(user_i)
    }
  },
  # set errors as warning message
  error = function(error_mes) {
    Sys.sleep(0.73)
    m <- error_mes$headers$status
    message(m, " >> ", url)
  })
}

# scrape data partly
user_inforP1 <- rbindlist(lapply(df.rep$url[1:10000], function(x)get_user(x)))
user_inforP2 <- rbindlist(lapply(df.rep$url[10001:20000], function(x)get_user(x)))
user_inforP3 <- rbindlist(lapply(df.rep$url[20001:30000], function(x)get_user(x)))
user_inforP4 <- rbindlist(lapply(df.rep$url[30001:40000], function(x)get_user(x)))
user_inforP5 <- rbindlist(lapply(df.rep$url[40001:50000], function(x)get_user(x)))
# combain these parts
user_infor <- rbindlist(list(user_inforP1, user_inforP2, user_inforP3,
                             user_inforP4, user_inforP5))
# write in a feather file
write_feather(user_infor, "userLocation.feather")

## same process for the sample files 2-6.
```

The same process above is repeated with all the sampled data files to collect all the data regarding the users location.

Collecting the number of watchers, forks, stargazers, languages, and created time

```r
# acquire: character -> data.frame
# Args:
#   x :Sampled data frame of size 50000 with urlname,language
# Returns:
# attributes:List of 50000 elements with each sublist containing 4 elements
# Purpose: collect the number of watchers, forks, stargazers, and created time
acquire <- function(url){
  rep.v <- str_split_fixed(url, "/", 2)
  tryCatch({
    Sys.sleep(0.73)
    proj_i <- gh("GET /repos/:owner/:repo", owner = rep.v[,1],
                 repo = rep.v[,2], .limit = 15000)[c("subscribers_count", "stargazers
_count",
                                                    "forks_count", "created_at", "la
nguage")]
    # identify if language is null
    message("OK >> ", url)
    proj_i <- data.frame(url = url, watchers_count = proj_i$subscribers_count,
                         stargazers_count = proj_i$stargazers_count,
                         forks_count = proj_i$forks_count,
                         createdTime = proj_i$created_at,
                         language = proj_i$language)
      return(proj_i)
    },
  # set errors as warning message
  error = function(error_m){
    Sys.sleep(0.73)
    m <- error_m$headers$status
    message(m, " >> ", url)
  })
}
# scrape data partly
proj_inforP1 <- rbindlist(lapply(user_infor$url[1:10000], function(x)get_user(x)))
proj_inforP2 <- rbindlist(lapply(user_infor$url[10001:20000], function(x)get_user(x))
)
proj_inforP3 <- rbindlist(lapply(user_infor$url[20001:30000], function(x)get_user(x))
)
proj_inforP4 <- rbindlist(lapply(user_infor$url[30001:40000], function(x)get_user(x))
)
proj_inforP5 <- rbindlist(lapply(user_infor$url[40001:50000], function(x)get_user(x))
)
# combain these parts
proj_inforV1 <- rbindlist(list(proj_inforP1, proj_inforP2, proj_inforP3,
                               proj_inforP4, proj_inforP5))

## same process for sample file 2 - 6
```

The same process above is repeated for all the sampled data samples to collect all the projects information.

5. Scraping country name

The user's location has various levels of address. Some users provide only the country name and some contain city or state names. To address this problem and maintain uniformity we use Google geocoding API which takes the input of the users location and it returns the country name for the user entered location. The geocode() function from R's ggmap package is used to interact with the Google Geocoding API. The Google geocoding API has a limit of 2500 requests per day. An API key is provided by registering with the Google account.

The geocodeQueryCheck() gives the information regarding the remaining requests.

```r
# Geocoding of users location using Google Geocoding API
devtools::install_github("dkahle/ggmap")
register_google(key = "Key obtained from Google Geocoding API")

# get_country: vector, character -> list
#
# Purpose: this function will takes the locations and the number of locations,
# and scrape the country name from google api. It will provides the message
# for each request.
get_country <- function(l, n) {
  c <- 0
  loc <- list()
  for (i in 1:n) {
    c <- c + 1
    tryCatch({
      loc[[i]] <- geocode(as.character(l[i]), output = "more")$country
      message("OK >> ", c)
    },
    # set errors as warning message
    error = function(error_mes) {
      m <- error_mes$headers$status
      message(m, " >> ", c)
    })
  }
  return(loc)
}
# The output is a list which gives the country names for the locations and it returns
NULL values
# for some of the invalid locations
country <- get_country(user_infor$location[1:2500], 2500)

# nulltona: list -> list
# Purpose: Converting the NULL values into NA
# Args: The output list of get_country()
nulltona <- function(x) {
  x[sapply(x, is.null)] <- NA
  return(x)
}
# Adding the country attribute to the data frame
```

```
user_infor$country <-
  as.vector(unlist(lapply(nulltona(country), as.character)))

# convert country name to country code
user_infor$countrycode <-countrycode(user_infor$country, "country.name", "iso3c")
# right join two data set: user's locations data set, and repors infor data set
df.com <- user_infor %>%
  select(url, country, countrycode) %>%
  right_join(projectinfo, by = "url")

# perform same process for another five sample files
```

6. Data resample and acquisition

We find 50,000 samples is not enough with the deepening of our project. So there are another 350,000 samples been collected. We performed the same steps as we did before, to get samples and scrape GitHub data. Then, combined the scraped data files together. The data set we will use in following analysis is `totalNew.feather` (37.7 MB), which contains all formatted sample data set.

2. Data Visualization

1. Data Loading

After previous steps, we have 362,460 samples in one data file and 37,540 dead links. There are nine parts: repository names, country names, country code, created time, forks count, watchers count, stargazers count, create year and language in the `totalNew.feather` file.

```
# whole sample data loading
total <- read_feather("totalNew.feather")
# subset the data have country and countrycode
total_country <- total[!is.na(total$countrycode),]
# table: repos with locations, and the repos without locations
loc_tbl <- matrix(c(dim(total_country)[1], sum(is.na(total$countrycode)), dim(total)[
1]), nrow = 3) %>%
  as.data.frame() %>%
  mutate(perc = paste0(signif((V1/dim(total)[1])*100, 3), "%"))
rownames(loc_tbl) <- c("With Location", "Without Location", "Total")
colnames(loc_tbl) <- c("Count", "Percentage")
kable(as.matrix.data.frame(loc_tbl))
```

	Count	Percentage
With Location	149111	41.2%
Without Location	213140	58.8%
Total	362251	100%

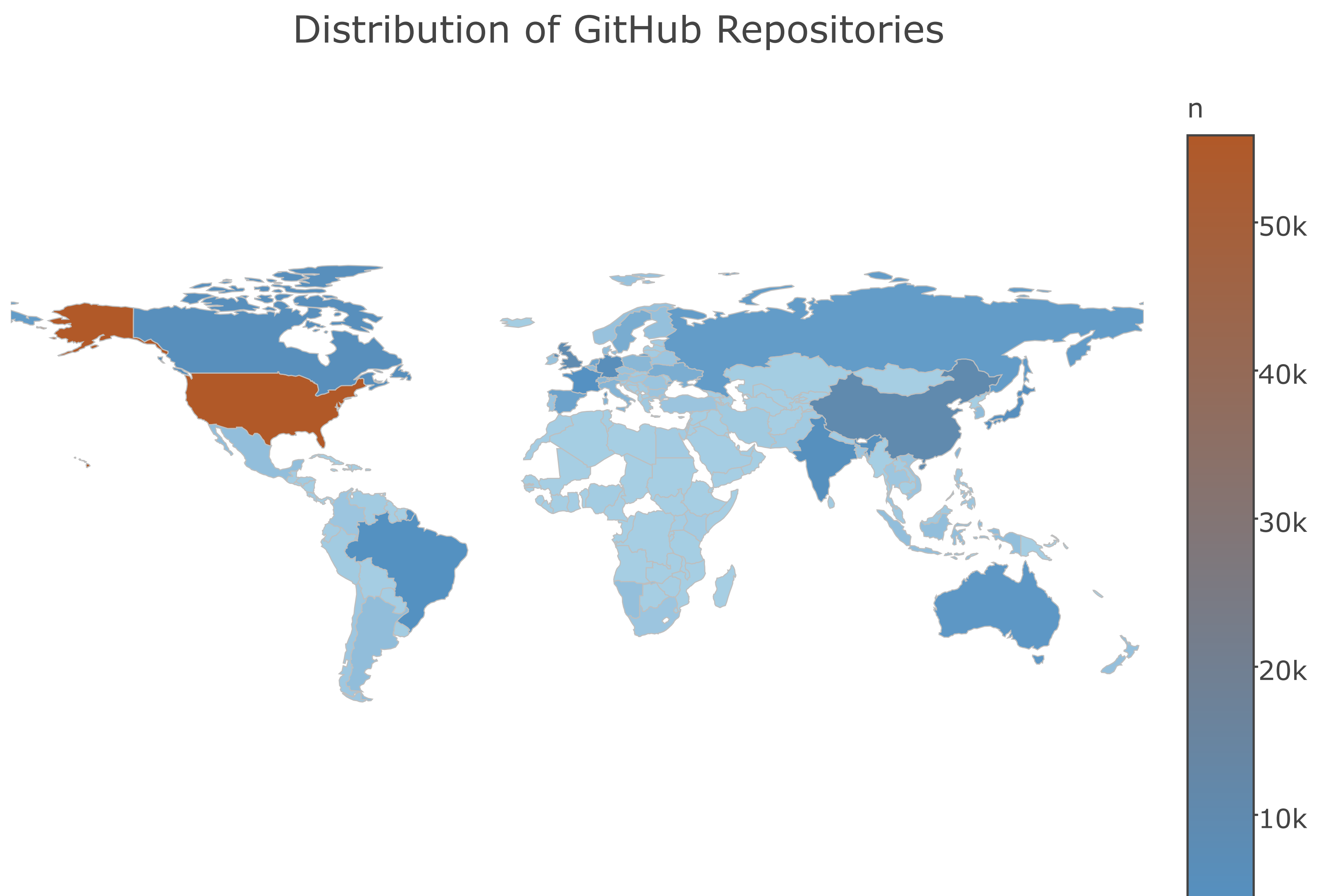
The above table describes the number of repositories sampled from the GHTorrent Data set and scraped using the GitHub API.The data set also contains some dead links of the repositories.Out of all the scraped repositories without dead links only 41% of the repositories developer's mentioned their location in their GitHub profile.The other 58% developers did not mention their geographical location in their profile.

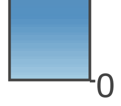
2. Graph and table

Choropleth Map: whole data set

Choropleth map will been used to demonstrate the situation of the GitHub repositories geographical distribution.

```
# choropleth map: country
l <- list(color = toRGB("grey"), width = 0.5)
g <- list(
  showframe = F,
  showcoastlines = F,
  projection = list(type = "Mercator")
)
total_country %>%
  count(countrycode) %>%
  arrange(-n) %>%
  plot_geo() %>%
  add_trace(
    z = ~n, color = ~n, colors = "Paired",
    text = ~countrycode, locations = ~countrycode, marker = list(line = l)) %>%
  layout(geo = g, title = "Distribution of GitHub Repositories")
```





Above choropleth map tells that GitHub repositories come from different parts of the world, but most of the repositories come from the United States (55.822K). United Kingdom, China, Australia, United States, Poland, Germany, Canada, United States, Germany, United States are the top 10 countries. The specific information about the top 10 countries, and top 10 programming languages will show in the results part.

Results

1. Previous analysis

In our previous analysis the non uniformity of the data for the year 2015 is due to filtering out the data which contain `\\N` for the language column in the GHTorrent data set. Not all data which has `\\N` in the GHTorrent data set has null values for the language, some data which even contains `\\N` for the language has its language in GitHub. So we sampled the data again from the GHTorrent data set without filtering out the repositories with `\\N` in language. We eliminated the repositories with NULL values in the language in the acquire function (function used to scrape data using GitHub API).

```r
# table: years - for the whole sample
tbl1 <- total %>%
  count(year) %>%
  mutate(perc = paste0(signif((n/sum(n))*100, 3), "%"))%>%
  kable(col.names = c("Year", "Number of Repositories", "Percentage"))
# table: for the repos with location
tbl2 <- total[!is.na(total$country), ] %>%
  count(year) %>%
  mutate(perc = paste0(signif((n/sum(n))*100, 3), "%")) %>%
  kable(col.names = c("Year", "Number of Repositories", "Percentage"))
```

The tables above describes the number of repositories created year wise from the starting year 2008 to 2017.

The first table is the yearly repositories distribution of the whole data set, the second table based on the sample with location. The percentage of year 2016 is obviously larger than year 2015 in the first table, but in the second table the difference between year 2016 and year 2015 is smaller than the first table. Thus, we suspect that the usage of GitHub tends to increase in the past nine years (year 2008 to year 2016), but the repositories with location do not increase as much as the first table shows.

2. Top 10 countries and top 10 programming languages

In this part we would like to focus on top 10 countries, and find out the mostly used (top 10) programming languages in whole data sample. Since we still in year 2017 and the number of repositories created in this year is not enough, we will ignore the repositories created in 2017.

Table of Top 10 Countries

The number of the repositories and the percentage of top 10 countries will be displayed in the following tables. The top 10 countries will be compared with other countries.

```
# data subset
# count data: find the frequencies of different programming languages in different co
untries
countr_vs_l <- total_country %>%
  group_by(country, language) %>%
  count() %>%
  arrange(-n)
# the number of repo of different countries
countr_count <- total_country %>%
  count(country, countrycode) %>%
  arrange(-n) %>%
  mutate(prob = n/dim(total_country)[1])
# table: top 10 ountries
countr10_tbl <- rbindlist(
  list(countr_count[1:10, -(1:2)],
       as.data.frame(matrix(c(dim(total_country)[1]-sum(countr_count$n[1:10]),
                              dim(total_country)[1],
                              (1 - sum(countr_count$prob[1:10])), 1),
                            nrow = 2))))
countr10_tbl$prob <- paste0(signif((countr10_tbl$prob*100), 3), "%")
rownames(countr10_tbl) <- c(as.character(countr_count$country[1:10]), "Other Countrie
s", "Total")
kable(as.matrix.data.frame(countr10_tbl), col.names = c("Number of Repositories", "Pe
rcentage"))
```

	Number of Repositories	Percentage
United States	55822	37.4%
China	10285	6.9%
United Kingdom	9577	6.42%
Germany	6137	4.12%
Canada	5528	3.71%
Japan	5039	3.38%
India	4937	3.31%
Brazil	3811	2.56%
France	3519	2.36%
Australia	3183	2.13%
Other Countries	41273	27.7%

Total	149111	100%

The above table describes about the number of repositories created in each developer's country.The developers in GitHub are from various geographical locations of the world. Since our analysis focuses only on top 10 developer's countries all the repositories from other countries fall under the category of Others which almost accounts to 27% of the total repositories on GitHub. The top 10 countries are found by counting the number of repositories created in each developer's location. United States ranks as the top 1 country. Almost 37% of all the repositories developers are from United States.

Table of Top 10 Languages

There are two tables be generated about the top 10 languages. One is the top 10 languages in the whole sample data set, another one is the top 10 languages in the data set with location. To eliminate the bias of the data do not have location, we will compare the top 10 languages in whole data set and the data set with location to see if we get seem top 10 programming languages.

```r
# top 10 languages
# table_print: data.frame -> formatted table
#
# Purpose: pick top 10 languages in data frame and print it as formatted table.
table_print <- function(df, x){
  tempdf <- df %>%
    count(language) %>%
    arrange(-n) %>%
    mutate(perc = signif((n/sum(n))*100, 3)) %>%
    top_n(10, perc)
  temp_tbl <- rbindlist(list(tempdf[, -1],
                             as.data.frame(matrix(c(dim(df)[1]-sum(tempdf$n),
                                                     dim(df)[1],
                                                     (100-sum(tempdf$perc)), 100),
                                                   nrow = 2))))
  rownames(temp_tbl) <- c(as.character(tempdf$language), "Others", "Total")
  return(as.matrix.data.frame(temp_tbl))
}
# top 10 languages in the whole data set
kable(table_print(total), col.names = c("Number of Repositories", "Percentage"))
```

	Number of Repositories	Percentage
JavaScript	66032	18.20
Java	53869	14.90
Python	31918	8.81
Ruby	27850	7.69
HTML	25283	6.98
PHP	24736	6.83

CSS	20927	5.78
C++	15818	4.37
C#	14151	3.91
C	13563	3.74
Others	68104	18.79
Total	362251	100.00

```
# top 10 languages in data sample with locations
kable(table_print(total_country), col.names = c("Number of Repositories", "Percentage
"))
```

	Number of Repositories	Percentage
JavaScript	30851	20.70
Java	18052	12.10
Python	13808	9.26
Ruby	12866	8.63
PHP	10476	7.03
HTML	8842	5.93
CSS	8602	5.77
C++	5896	3.95
C	5225	3.50
C#	5148	3.45
Others	29345	19.68
Total	149111	100.00

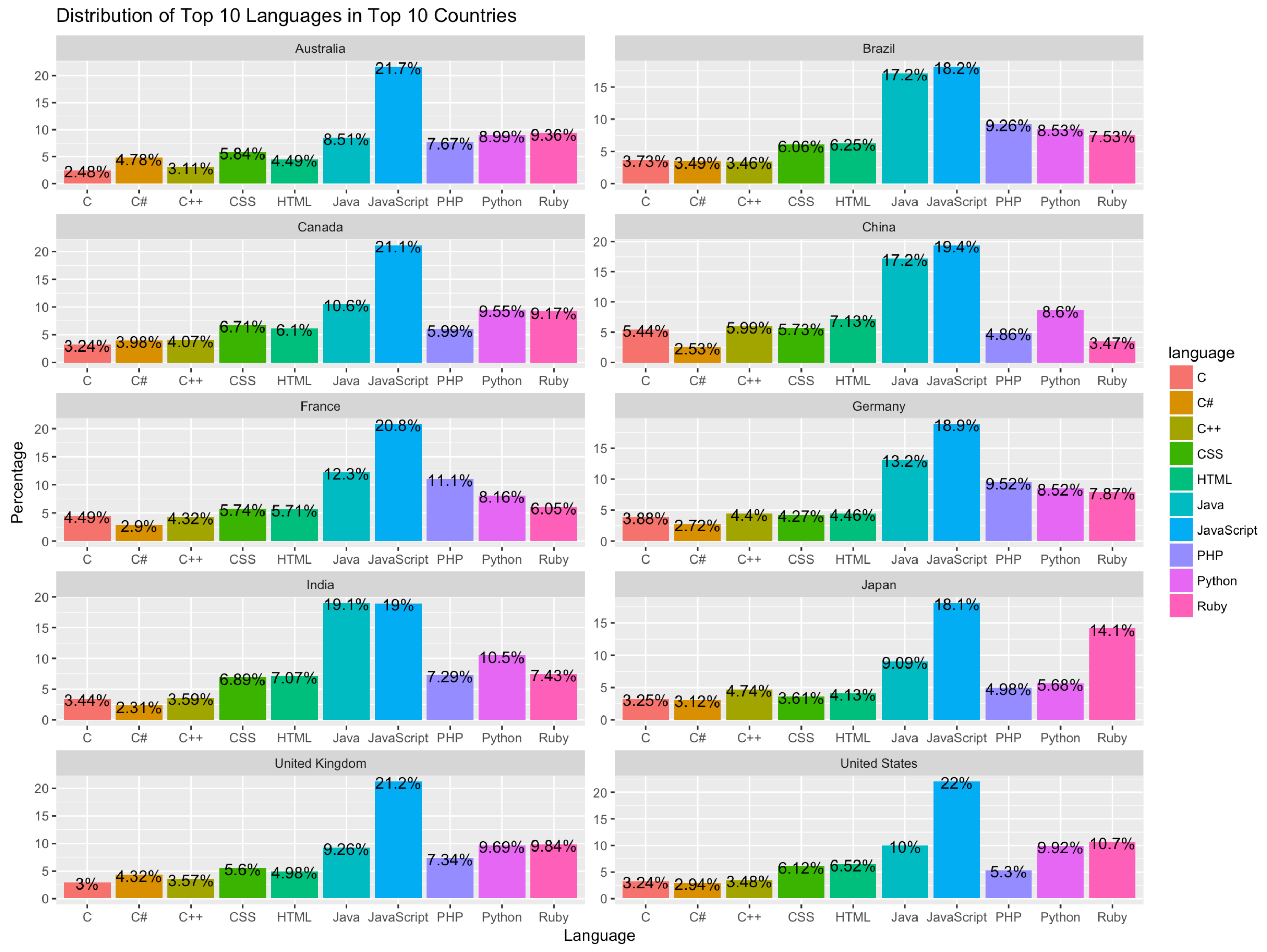
The tables above show that the top 10 programming languages are same in different data sets, and these languages are JavaScript, Java, Python, Ruby, HTML, PHP, CSS, C++, C#, C. These tables also describes the number of repositories for each language in GitHub and its percentage from the total repositories which contains developers location. Since in our analysis we focus only on the top 10 GitHub programming languages all the other languages comes under Others which forms only around 19% of the total repositories. The Top 10 languages are counted by the number of repositories for each language. JavaScript is the most popular language as more than 18% of the total repositories use JavaScript as their programming language.

3. Usage Distribution of top 10 langugaes in top 10 countries

```r
countr_l_perc <- countr_vs_l %>%
  left_join(countr_count, by = "country") %>%
  # calculate the percentage based on different ountries
  mutate(prob = (n.x/n.y)*100)
# 10 languages
top10_l <- total_country %>%
    count(language) %>%
    arrange(-n) %>%
    top_n(10, n)

# 10 languages in top 10 countries
top10_c <- countr_l_perc %>%
  filter(language %in% top10_l$language) %>%
  filter(country %in% countr_count$country[1:10])

# Graph
#
# graph to show 10 languages in 1o different countries
# top 10 languages here are selected from whole data set (contains repos with locatio
n and repos without locations)
top10_c %>%
  ggplot(aes(x = language, y = prob, fill = language)) +
  geom_bar(stat = "identity") +
  facet_wrap(~country, nrow = 5, scales = "free") +
  geom_text(aes(label = paste0(signif(prob, 3), "%")), vjust = .6) +
  labs(x = "Language", y = "Percentage", title = "Distribution of Top 10 Languages in
Top 10 Countries")
```



The above graph tells us about the variation in the usage of programming languages on GitHub at different developer locations.

How does the usage of programming languages differ among the countries?

From the above graph it is seen that the percentage usage of a programming languages in each country is different from the usage of the same languages in other countries. There is also a difference in the usage of programming languages in each country.

The percentages described in the graph are the percentages of the number of repositories of a language in that particular country. These percentages are obtained by using the aggregated metrics of the number of repositories.

The usage of JavaScript is high in all countries whereas in India,the usage of Java and JavaScript are almost equal.

There is a much difference seen between the usage of Java and JavaScript in United Kingdom and United States than other countries.

In United Kingdom,the usage of Ruby is higher compared to the Java and Python.

The usage of Ruby is high in Japan, United Kingdom and United States compared to PHP, Python, Java and HTML whereas only 3% of China's repositories are of Ruby language.

PHP users are more in France, Germany and Brazil than Python and Ruby users whereas the situation differs in other countries.

HTML usage is higher in China than Ruby and PHP.

The usage of Python is more in India than other countries. On contrast Japan has the lowest usage of Python.

Japan has the lowest usage of C# than other languages.

Every country has more number of C users than C# but in Australia and United Kingdom the usage of C# is more than C.

Usage Percentage of Programming Language in each Country

```
# tables
#
# percentages of each languages in different countries
top10_c_l <- list()
for(i in 1:10){
  top10_c_l[[i]] <- as.data.frame(matrix(
    signif(top10_c$prob[top10_c$countrycode == countr_count$countrycode[i]], 3),
                         nrow = 1,
    dimnames = list("Country" = countr_count$country[i],
                    "Language" = as.character(top10_c$language[top10_c$countrycode ==
countr_count$countrycode[i]]))))
}
top10_c_l <- rbindlist(top10_c_l, use.names = T)
rownames(top10_c_l) <- as.character(countr_count$country[1:10])
top10_c_l <- as.matrix.data.frame(top10_c_l)
kable(top10_c_l)
```

	JavaScript	Ruby	Java	Python	HTML	CSS	PHP	C++	C	C#
United States	22.0	10.70	10.00	9.92	6.52	6.12	5.30	3.48	3.24	2.94
China	19.4	3.47	17.20	8.60	7.13	5.73	4.86	5.99	5.44	2.53
United Kingdom	21.2	9.84	9.26	9.69	4.98	5.60	7.34	3.57	3.00	4.32
Germany	18.9	7.87	13.20	8.52	4.46	4.27	9.52	4.40	3.88	2.72
Canada	21.1	9.17	10.60	9.55	6.10	6.71	5.99	4.07	3.24	3.98
Japan	18.1	14.10	9.09	5.68	4.13	3.61	4.98	4.74	3.25	3.12
India	19.0	7.43	19.10	10.50	7.07	6.89	7.29	3.59	3.44	2.31
Brazil	18.2	7.53	17.20	8.53	6.25	6.06	9.26	3.46	3.73	3.49
France	20.8	6.05	12.30	8.16	5.71	5.74	11.10	4.32	4.49	2.90
Australia	21.7	9.36	8.51	8.99	4.49	5.84	7.67	3.11	2.48	4.78

The above table describes about the percentage usage of each of the top 10 languages in the top 10 developer countries. The percentage is calculated by counting the number of repositories in each language of that country. From this table we can analyse the usage of programming language in each developer's country.

Ranking of Programming Language in each Country

```
# arrange position of each language in different countries
# language top 1-10 will been label as 1-10
top10_c_l_rank <-list()
for(i in 1:10){
  temp <- factor(top10_c_l[i,])
  levels(temp) <- c(10:1)
  top10_c_l_rank[[i]] <- t(as.data.frame(temp))
  top10_c_l_rank[[i]] <- as.data.frame(top10_c_l_rank[[i]])
}
top10_c_l_rank <- rbindlist(top10_c_l_rank, use.names = T)
rownames(top10_c_l_rank) <- as.character(countr_count$country[1:10])
kable(as.matrix.data.frame(top10_c_l_rank))
```

	JavaScript	Ruby	Java	Python	HTML	CSS	PHP	C++	C	C#
United States	1	2	3	4	5	6	7	8	9	10
China	1	9	2	3	4	6	8	5	7	10
United Kingdom	1	2	4	3	7	6	5	9	10	8
Germany	1	5	2	4	6	8	3	7	9	10
Canada	1	4	2	3	6	5	7	8	10	9
Japan	1	2	3	4	7	8	5	6	9	10
India	2	4	1	3	6	7	5	8	9	10
Brazil	1	5	2	4	6	7	3	10	8	9
France	1	5	2	4	7	6	3	9	8	10
Australia	1	2	4	3	8	6	5	9	10	7

The above table describes about the usage of the Programming Languages in each developer's country. A rank is assigned to each of the top 10 languages according to the usage of the language in that country. The ranks for the programming languages in a country varies according to its usage. JavaScript is top most used programming language in almost all countries except India so it is assigned a rank 1 in all other countries. In India a rank 1 is assigned to Java since it is the most used language in India. Ruby is assigned a rank 2 since it is the 2nd most used language in United States and a rank 9 in China as it is in the 9th position in China according to the usage.

Conclusion

Our analysis helps to find out he usage trends of the top programming languages in GitHub at various developer's locations. The variation is seen in the usage of programming languages at different developer locations. From the above analysis it is clear that the usage of the programming languages in GitHub are different in various developer's locations and it supports our hypothesis.

Our analysis helps in identifying developers geographic location which is especially important to employers to see the usage trends of the programming languages in those particular locations and decide on which technology to be employed so that it is most likely to lead a product supported by a large pool of available developers during both the development stage and later in the maintenance stage.

Known Limitations

This hypothesis is valid for the repositories in GitHub which contain their developer's location. Since all developers do not include their geographical location in their Github profile, it is difficult to deal with the repositories which do not contain the developers location. The analysis is also based only on the repositories owned by the developer and it does not take into account the contributions made by the developer to other projects in GitHub.