

# RでWorld Population Prospects(WPP)を使う

新保一成

2017年5月20日

# 準備

- <http://www.fbc.keio.ac.jp/~shimpo/db/wpp2015.html>からデータベース・ファイルwpp2015.dbをダウンロード.
- 同サイトでwpp2015.dbに収録されているテーブルの定義を確認.

# Why RDBMS and R?

- Big Dataという言葉が示すように、利用可能なデータのサイズは巨大化している.
- 一方で、コンピューター上の資源(メモリーやディスク)は有限である.
- これらを効率的に使うためには、巨大なデータの中から必要なものを選択してメモリー上に格納する必要がある.
- その目的の達成するためにSQLiteなどのRDBMS(関係データベース管理システム)が有効である.
- 一旦コンピューター上に格納したデータをRパッケージの1つであるdplyr, tidyrによって効率的に処理することができる.

# パッケージのインストール

- 以下のパッケージが必要.
  - RSQLite(RからSQLiteを使う)
  - tidyverse(dplyr, tidyr, ggplot2などデータの整理整頓に便利なパッケージの集まり)
  - pyramid(人口ピラミッド)

# パッケージのロード

```
library(RSQLite)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages -----
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

# データベースの接続と切断

- dbConnect: データベースの接続
  - dbDriverでRDBMSにSQLiteを使うことを指示
  - dbnameにデータベース・ファイルを指定
  - dbConnectで接続. dbConnectの返回值conを通じて以後データベースと通信する.

```
driv <- dbDriver("SQLite")  
dbname <- "wpp2015.db"  
con <- dbConnect(driv, dbname)
```

- dbDisconnect: データベースの切断
  - データベースに接続する必要がなくなったら必ず切断する

```
dbDisconnect(con)
```

# SQL: SELECT文

- データベースからデータを取得するには、Rなどの外部プログラムからSQLで命令を発行しなければならない。  
SQLとは関係データベースシステム(relational database system, RDBMS)へ外部プログラムから命令を発行する言語である。
- SQL言語には様々な機能があるが、当面はテーブルからデータを抽出するためのSELECT文を理解していれば十分。
- あるテーブルに登録されている変数を取得するためのSELECT文は次のとおり。変数名に'\*'を指定すると、そのテーブルに含まれる全ての変数を取得する。

SELECT 変数名, 変数名, ... FROM テーブル名

# SQL: 条件付きSELECT文

- SELECT文にWHERE句を追加して、条件に適合したデータのみを取得することができる.

SELECT 変数名, 変数名, ... FROM テーブル名 WHERE 条件

- WHERE句で使える演算子を次頁に示す.



## SQL WHERE句で使える比較演算子

演算子	説明	演算子	説明
=	等しい	AND	左辺と右辺の条件の両方が真
>	大きい	OR	左辺と右辺のどちらかが真
<	小さい	NOT	否定
>=	大きい, または等しい	BETWEEN a AND b	対象変数がaとbの範囲内
<=	小さい, または等しい	IN	対象変数が一覧の1つに一致
!=	等しくない	LIKE	対象変数がパターンに一致
<>	等しくない		

# dbGetQuery: データの取得

- RのデータベースインターフェースDBIが提供するdbGetQuery関数でSQL文を発行してデータを取得する.
- dbGetQueryの基本構文は以下のとおりで, dbConnectで接続したコネクションとSQL文の順で指定する.
- 返り値xはデータフレームで, テーブルに登録されている変数名がそのままデータフレームの変数名になる.

```
x <- dbGetQuery(データベースコネクション, SQL文)
```

# 国・地域データの取得

- WPPの国・地域データの全てを読み込んでみよう.
- 国・地域データはlocationテーブルにある.
- 全てを読み込むからSELECT文の変数名には'\*'を指定する.

```
location    <- dbGetQuery(con, "SELECT * FROM location")
loc_type    <- dbGetQuery(con, "SELECT * FROM loc_type")
region      <- dbGetQuery(con, "SELECT * FROM region")
major_area  <- dbGetQuery(con, "SELECT * FROM major_area")
```

# データの閲覧

- dbGetQueryの結果はデータフレームlocationに保存されている.
- 右上のWorkspace画面のEnvironmentタブに生成されたオブジェクトのリストがある.
- データフレームはその中のDataにリストアップされている.
- locationをクリックすると左上のEdit画面にlocationタブが作成され, locationの内容を閲覧することができる.
- 左下のコンソール画面で, View(location)を実行しても同じ結果を得ることができる.
- 次頁にlocationからloc\_id, loc\_name, iso3, ldr, incomeを選択して, ランダムに抽出した7レコードを表示した.

location: ランダムに抽出した7レコード

	loc_id	loc_name	iso3	ldr	income
179	40	Austria	AUT	0	HIC
71	562	Niger	NER	1	LIC
41	180	Democratic Republic of the Congo	COD	1	LIC
208	630	Puerto Rico	PRI	1	HIC
163	20	Andorra	AND	0	HIC
116	922	Western Asia	NA	NA	NA
61	854	Burkina Faso	BFA	1	LIC

# 演習1: dplyrによるデータ操作

- 課題
  - 合計特殊出生率(TFR)に関する所得グループ別の要約統計量(四分位数, 標本平均)を1950-55年から20年毎に計算する.
- 計算戦略
  - TFR(period\_main.tfr)と所得グループ(location.income)は, それぞれ異なるテーブルperiod\_mainとlocationに含まれる. → period\_mainとlocationの列をマージする.
  - 世界, 先進国, アジアなどの地域のincomeは欠損値(=NA)である. → incomeがNAの地域を除く
  - 特定の所得グループ, 期間にデータを絞り込む
  - 要約統計量を計算する

# dplyr: 1つのデータ・テーブル 内のデータ操作(1)

- `filter(data, condition)`: dataのうちconditionに一致する行だけに絞り込む.
- `selectr(data, var1, var2, ...)`: dataから変数名で列を選択する.
- `mutate(data, definitions)`: 新たな変数をdefinitionsに従ってdataに列を追加する.
- `arrange(data, keys)`: keysをキーにしてdataの行をソートする.
- `summarize(data, aggregation)`: aggregationの命令にしたがってdataの変数を集計する.

# dplyr: 1つのデータ・テーブル 内のデータ操作(2)

- `sample_n(data, n)`: dataからランダムに $n$ 行を抽出する.
- `sample_frac(data, p)`: dataからランダムに $p \times 100\%$ の行を抽出する.
- `n(data)`: 行数をカウントする.
- `distinct(data, col)`: 列colに関して重複のない行を作成する.



# dplyr: 2つのデータ・テーブル の結合(マージ)

- 結合関数は, `by`で指定した列をキーにして2つのテーブルを結合する. ただし, `by`が与えられない場合には, 2つのテーブルに共通する全ての列をキーにして結合する.
  - `left_join(x, y, by)`: `x`の全ての行を残し, `y`の対応した行のみを残す.
  - `inner_join(x, y, by)`: `x`と`y`の両方にある行のみ残す.
  - `semi_join(x, y, by)`: `y`にマッチする`x`の行のみ残す.
  - `anti_join(x, y, by)`: `y`にマッチしない`x`の行のみ残す.

# Rの論理演算子

- filterでデータを絞り込むときに論理演算子を使って条件を記述する。先に説明したSQLの論理演算子と記号が異なるものがあるので注意してほしい。

演算子	説明	演算子	説明
==	等しい	&	左辺と右辺の条件の両方が真
>	大きい		左辺と右辺のどちらかが真
<	小さい	!	否定
>=	大きい, または等しい	%in%	対象変数が一覧の1つに一致
<=	小さい, または等しい		
!=	等しくない		

# 演習1 コード: データの読み込み

- dbConnectでwpp2015.dbに接続できていることを確認
1. locationテーブルの全て,
  2. period\_mainテーブルに関してはloc\_id, periodおよびtfrの中位推計値(projection = 2)のみを読み込む. period\_mainのデータは, データフレームtfrに読み込む.
  3. SQLが長くなる場合には, 例のように適当な箇所でSQLを区切ってpaste関数で繋げればよい.

```
location <- dbGetQuery(con, "SELECT * FROM location")
sql <- paste("SELECT loc_id, period, mid_period, tfr FROM period_main",
             "WHERE projection = 2")
tfr <- dbGetQuery(con, sql)
```

# 演習1 コード: データ・テーブルの結合

1. locationのうちloc\_id, iso3, ldr, ldc, incomeをselect関数で選択し, tfrにleft\_joinで結合する.
2. incomeが欠損値(=NA)の行を削除する. RでNAとは, Not Available, すなわち欠損値を指す. is.na関数はNAならTRUE, そうでないならFALSEを返すので, is.na関数を!で否定すれば, NAでないものがTRUEになる.
3. 次頁にtfrからランダムに抽出した10行を表示する.

```
tfr <- left_join(tfr, select(location, loc_id, iso3, ldr, ldc, income),  
                by = "loc_id")  
tfr <- filter(tfr, !is.na(income))
```

```
sample_n(tfr, 10)
```

##	loc_id	period	mid_period	tfr	iso3	ldr	ldc	income
## 477	52	2080-2085	2083	1.864	BRB	1	0	HIC
## 1847	246	2030-2035	2033	1.802	FIN	0	0	HIC
## 743	90	2060-2065	2063	2.371	SLB	1	1	LMIC
## 967	124	1980-1985	1983	1.634	CAN	0	0	HIC
## 2461	348	1950-1955	1953	2.686	HUN	0	0	HIC
## 1954	266	1965-1970	1968	4.933	GAB	1	0	UMIC
## 6274	882	1965-1970	1968	7.353	WSM	1	0	LMIC
## 4858	670	2085-2090	2088	1.781	VCT	1	0	UMIC
## 1773	234	1960-1965	1963	NA	FRO	0	0	HIC
## 2599	364	2040-2045	2043	1.578	IRN	1	0	UMIC

# 演習1 コード: 特定グループ, 期間の要約統計量の計算

1. 高所得国の2010-2015年に絞り込む.
2. TFRの四分位数, 標本平均を計算する.  $q_0$  =最小値(0%点),  $q_{25}$  =第1四分位(25%点),  $q_{50}$  =中央値,  $q_{75}$  =第3四分位(75%点),  $q_{100}$  =最大値(100%点), mean = 標本平均. 国によってはTFRが欠損値(NA)なので, 要約統計量を計算する関数でna.rm = TRUEとして欠損値を除いて計算するように指示している.

```
tfr_group <- filter(tfr, income == "HIC" & period == "2010-2015")
tfr_summary <- summarize(tfr_group,
  q0    = min(tfr, na.rm = TRUE),
  q25   = quantile(tfr, probs = 0.25,
                    na.rm = TRUE),
  q50   = median(tfr, na.rm = TRUE),
  mean  = mean(tfr, na.rm = TRUE),
  q75   = quantile(tfr, probs = 0.75,
                    na.rm = TRUE),
  q100  = max(tfr, na.rm = TRUE))
```

```
tfr_summary
```

```
##      q0    q25    q50      mean  q75  q100
## 1 1.192 1.463 1.782 1.840092 2.05 4.967
```

# dplyr: %>% (パイプ) の利用

- dplyrではパイプ機能を使うことができる. %>%がパイプで左辺の結果を右辺に渡す. パイプによって
  - 中間的な変数を作る必要がなくなり, 論理の流れが明確になるので, コードの見通しがよくなる.
  - 右側の関数に渡すデータ名を書かないので, コードの見通しがよくなる.
- 演習1のこれまでのコードをパイプで書き直す(次頁).



- 演習1のデータ読み込み以降をパイプを使って2段階に分ける(コードは次頁).
  1. 1段階目で国・地域データと結合し，所得グループとTFRそのものに欠損値があるデータを除いて，結果をあらためてtfrに保存している.
  2. 2段階目で，高所得国と2010-2015年に絞り込み，要約統計量を計算している．1段階目でTFRの欠損値をあらかじめ除去したので(6行目)，要約統計量の計算でna.rm = TRUEの必要がない.
  3. 中間変数tfr\_groupを生成する必要がなくなっている.

```

sql <- paste("SELECT loc_id, period, mid_period, tfr FROM period_main",
             "WHERE projection = 2")
tfr <- dbGetQuery(con, sql)
tfr <- tfr %>%
  left_join(select(location, loc_id, iso3, ldr, ldc, income),
            by = "loc_id") %>%
  filter(!is.na(income) & !is.na(tfr))
tfr_summary <- tfr %>%
  filter(income == "HIC" & period == "2010-2015") %>%
  summarize(q0    = min(tfr),
            q25    = quantile(tfr, probs = 0.25),
            q50    = median(tfr),
            mean    = mean(tfr),
            q75    = quantile(tfr, probs = 0.75),
            q100   = max(tfr))
tfr_summary

```

```

##      q0    q25    q50      mean  q75  q100
## 1 1.192 1.463 1.782 1.840092 2.05 4.967

```

# dplyr: group\_byによる繰り返し計算

- 先の例では、高所得者の2010-2015年についてだけTFRに関する要約統計量を計算した.
- group\_byに列を渡すと、その列で定義されるグループごとにそれ以降の計算を実行する.
- 例えば、group\_by(income)とすれば、それ以降の計算を所得グループ毎に実行する.
- 次頁で、group\_byを使って、各所得グループに属する国の数をカウントしてみる.

```
location %>%  
  group_by(income) %>%  
  summarise(n = n())
```

```
## # A tibble: 5 × 2  
##   income      n  
##   <chr> <int>  
## 1    HIC     78  
## 2    LIC     31  
## 3   LMIC     50  
## 4   UMIC     53  
## 5   <NA>     61
```

- group\_by(income)の下でsummarizeは所得グループ毎に計算されたことがわかる。
- 194カ国のうち、高所得国が78カ国、中所得国が114カ国、低所得国が31カ国ある。
- 授業で、最も貧しいと国連が認定した後発開発途上国 (LDC, Least Developed Countries)が48カ国あると聞いたが？

- LDCの数をカウントしてみよう.

```
location %>%  
  filter(ldc == 1) %>%  
  summarize(n = n())
```

```
##      n  
## 1  48
```

確かにLDCは48力国ある

- LDCの中には低所得国以外に分類されている国があるということだ.
- 国連はLDCに認定するのに所得, 人的資源開発, 経済的脆弱性の3指標を使う.
- 所得条件は, 1人あたりGNIの過去3年間の平均が低所得国に分類される水準にあること. したがって, 1時点の1人あたりGNIが低所得国以上ということはある.

```
location %>%  
  filter(ldc == 1 & !is.na(income)) %>%  
  group_by(income) %>%  
  summarize(n = n())
```

```
## # A tibble: 4 × 2  
##   income      n  
##   <chr> <int>  
## 1    HIC      1  
## 2    LIC     29  
## 3   LMIC     16  
## 4   UMIC      2
```

# dplyr: mutateとif\_elseで新しい 国グループを作る

- OECDのDACリストではLDCが別掲され, HIC, UMIC, LMIC, LICからLDC諸国は除かれている.
- DACリストの分類に対応した列dacをlocationテーブルに追加してみよう.
  - 新しい列の追加はmutate関数で行うが,
  - 各行について, LDCであるなら'LDC'とし, そうでなければincomeの値を適用する. これをif\_else関数で実現する.

```
location <- location %>%  
  mutate(dac = if_else(ldc == 1, 'LDC', income))  
location %>%  
  group_by(dac) %>%  
  summarize(n = n())
```

```
## # A tibble: 6 × 2  
##   dac      n  
##   <chr> <int>  
## 1   HIC    77  
## 2   LDC    48  
## 3   LIC     2  
## 4  LMIC   34  
## 5  UMIC   51  
## 6  <NA>   61
```



- 低所得国がたったの2カ国になってしまうので、後発開発途上国と低所得国を1つのグループとして、それを改めて dac とする.

```
location <- location %>%  
  mutate(dac = if_else(ldc == 1 | income == 'LIC', 'LDC_LIC', income))  
location %>%  
  group_by(dac) %>%  
  summarize(n = n())
```

```
## # A tibble: 5 × 2  
##       dac      n  
##   <chr> <int>  
## 1    HIC     77  
## 2 LDC_LIC    50  
## 3   LMIC     34  
## 4   UMIC     51  
## 5   <NA>     61
```

# 演習1: 解答例

```
sql <- paste("SELECT loc_id, period, mid_period, tfr FROM period_main",  
            "WHERE projection = 2")  
tfr <- dbGetQuery(con, sql)  
tfr <- tfr %>%  
  left_join(select(location, loc_id, iso3, ldr, ldc, income, dac),  
            by = "loc_id") %>%  
  filter(!is.na(income) & !is.na(tfr))  
tfr_summary <- tfr %>%  
  filter(mid_period <= ceiling(2010 + 2015) / 2) %>%  
  group_by(period, dac) %>%  
  summarize(q0    = min(tfr),  
            q25   = quantile(tfr, probs = 0.25),  
            q50   = median(tfr),  
            mean  = mean(tfr),  
            q75   = quantile(tfr, probs = 0.75),  
            q100  = max(tfr))
```

```
head(tfr_summary)
```

```
## Source: local data frame [6 x 8]
```

```
## Groups: period [2]
```

```
##
```

##	period	dac	q0	q25	q50	mean	q75	q100
##	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	1950-1955	HIC	1.980	2.68450	3.566	3.966609	5.09050	7.252
## 2	1950-1955	LDC_LIC	3.462	6.00000	6.400	6.449245	6.90000	8.000
## 3	1950-1955	LMIC	2.810	5.52100	6.402	6.086294	6.97250	7.630
## 4	1950-1955	UMIC	2.526	5.54650	6.138	5.857979	6.70000	7.599
## 5	1955-1960	HIC	1.950	2.53425	3.539	3.982078	5.15675	7.252
## 6	1955-1960	LDC_LIC	5.117	6.13400	6.601	6.555469	6.94700	8.150

# ggplot2: Line Graph

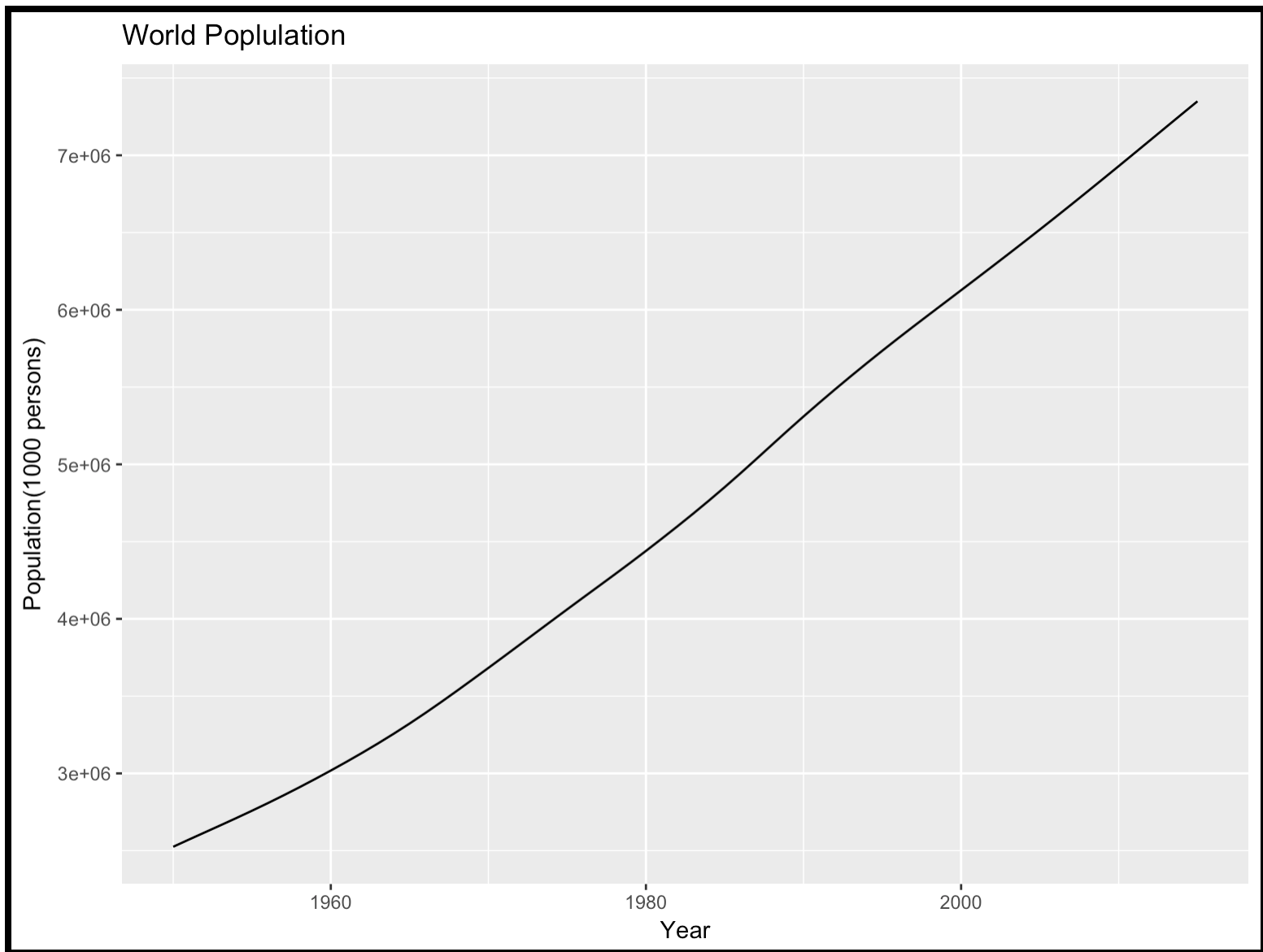
- pop\_annualテーブルの国・地域別の男女別および男女計総人口のデータを1950年から2015年の人口の推移を線グラフで描いてみよう.
- 対象は, 世界全体(loc\_id = 900), 先進国(loc\_id = 901), 開発途上国(loc\_id = 902)とする.
- WPPの先進国とは, ヨーロッパ, 北米, オーストラリア/ニュージーランド, 日本である.
- 以下のように, データを取得する. RStudioでpopの中身を確認してみよう.

```
sql <- paste("SELECT loc_id, year, pop_total FROM pop_annual",  
             "WHERE projection = 2 AND year <= 2015",  
             "AND loc_id in(900, 901, 902)")  
pop <- dbGetQuery(con, sql)
```

# ggplot2: 世界総人口の推移(1)

- ggplot2では、グラフの要素を'+'記号で連結することによって描画する。
  - ggplot関数のaesにx軸, y軸などの基本的な設定をし,
  - 続いて線グラフを描きたい場合にはgeom\_line()を指定し,
  - 軸ラベルには変数名が表示されるが, それを変えたいときには以下のように変更することもできる.
- [ggplot2のCheat Sheet](#), [ggplot2用例集 入門編](#), [ggplo2逆引き](#)などを参考にされたい.

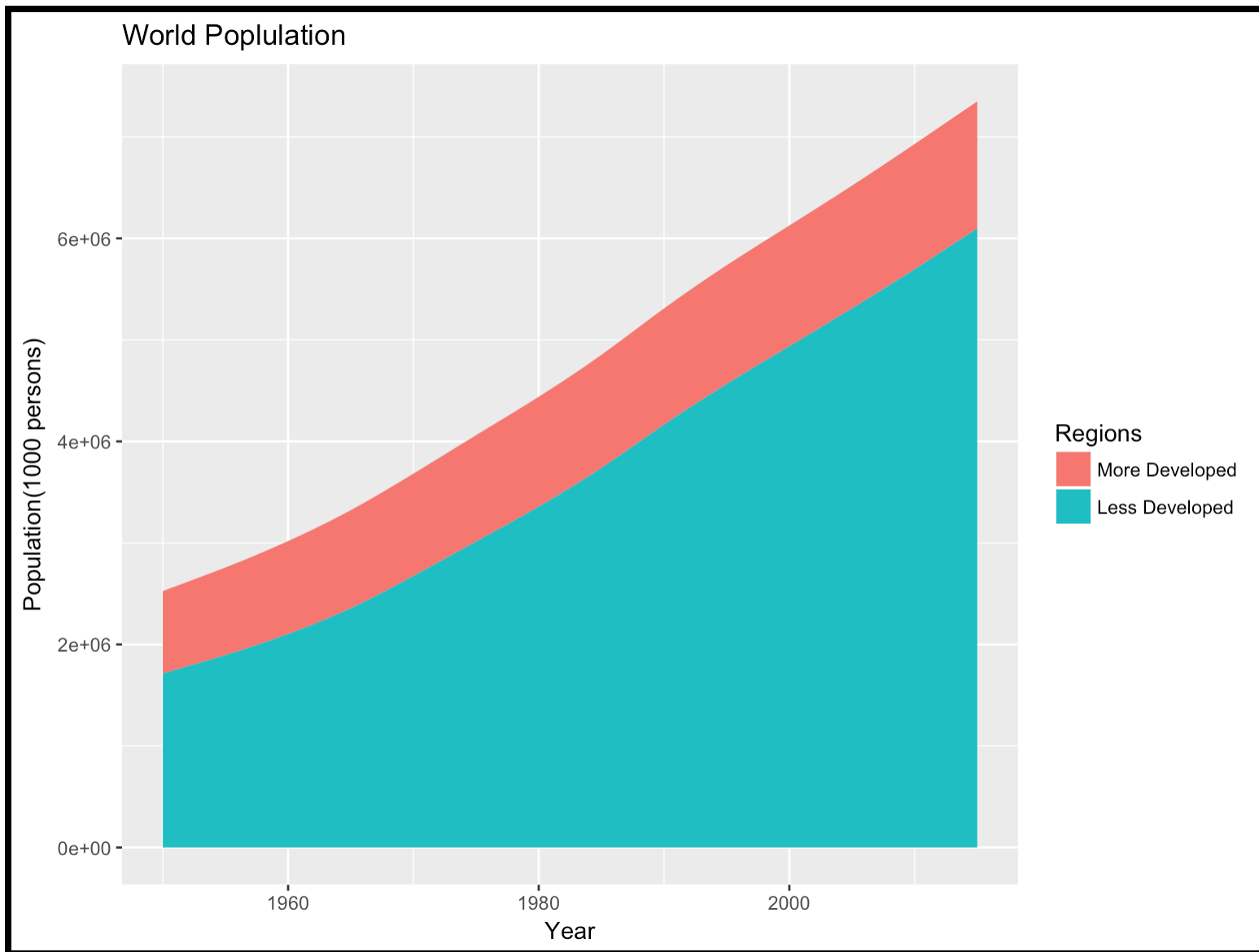
```
pop %>%  
  filter(loc_id == 900) %>%  
  ggplot(aes(x = year, y = pop_total)) +  
  geom_line() +  
  ggtitle("World Population") +  
  labs(x = "Year", y = "Population(1000 persons)")
```



# ggplot2: 世界総人口の推移(2)

- 先進国と開発途上国の積み上げグラフとして世界総人口の推移を描く
  - aesでどの変数で塗りつぶすかをfillに指定
  - 積み上げグラフを描きたいのでgeom\_areaを使う
  - 凡例をわかりやすくするように、いろいろやっている

```
pop %>%  
  filter(loc_id %in% c(901, 902)) %>%  
  ggplot(aes(x = year, y = pop_total, fill = factor(loc_id))) +  
  geom_area() +  
  ggtitle("World Population") +  
  labs(x = "Year", y = "Population(1000 persons)", fill = "Regions") +  
  scale_fill_discrete(labels = c("More Developed", "Less Developed"))
```





# 人口ピラミッド: パッケージ のロード

- 中澤港先生(神戸大学)が開発したpyramidパッケージを使って人口ピラミッドを描く
- pyramidパッケージはggplot2の一部ではないので, helpを読んでどのように使うかを確認する.
- 自分の仕事を全うするために, どんなパッケージが利用できるのかを探す能力も大事である.
- 最初にpyramidパッケージをロードする.

```
library(pyramid)
```

# 人口ピラミッド: 仕様

- pyramidの基本的な仕様は以下のとおり.

```
pyramid(Left, Right, Center)  
pyramid(data)
```

- 人口ピラミッドは年齢(階層)別の男女それぞれの人口データを年齢の低い順に積み上げた図であるから,
  - Leftに男性(女性)の年齢別人口データ,
  - Rightに女性(男性)の年齢別データを与え,
  - Centerには年齢(階層)のラベルを与える.
  - もちろん, 3つのデータは同じ年齢順に並んでいなければならない.
  - Leftを1列目, Rightを2列目, Centerを3列目に配置したデータフレームdataを渡すこともできる.

# 人口ピラミッド: データの読み込み

- 男女別年齢階層別のデータは, pop\_age\_sex\_1yテーブルから以下のように取得できる.
- 男女別のデータのみ(sex IN(1, 2)), 国・地域は世界(loc\_id = 900), 年次は2015年(year = 2015)という条件を付けて読み込んでいる.
- RStudioのデータ・ビューでpopの内容を確認してみよう.

```
sql <- paste("SELECT loc_id, year, sex, age_grp, age_start, pop",  
             "FROM pop_age_sex_1y",  
             "WHERE sex IN(1, 2) AND loc_id = 900 AND year = 2015")  
pop <- dbGetQuery(con, sql)
```

# 人口ピラミッド: データの加工

- 人口ピラミッドを人口の数値そのものではなくて構成割合で描く
  - popは1地域の1年時の男女別年齢階層別人口データであるから, pop列の合計値は総人口である.
  - したがってpop列に対して $\text{pop} / \text{sum}(\text{pop})$ は, 総人口に占める男女別年齢階層別の構成割合になる.
  - これをmutateで新しい列shareとして追加する. (1行目)
- 男女別のデータおよび年齢階層ラベルをそれぞれmale, female, ceterとして作成する. (2, 3, 4行目)

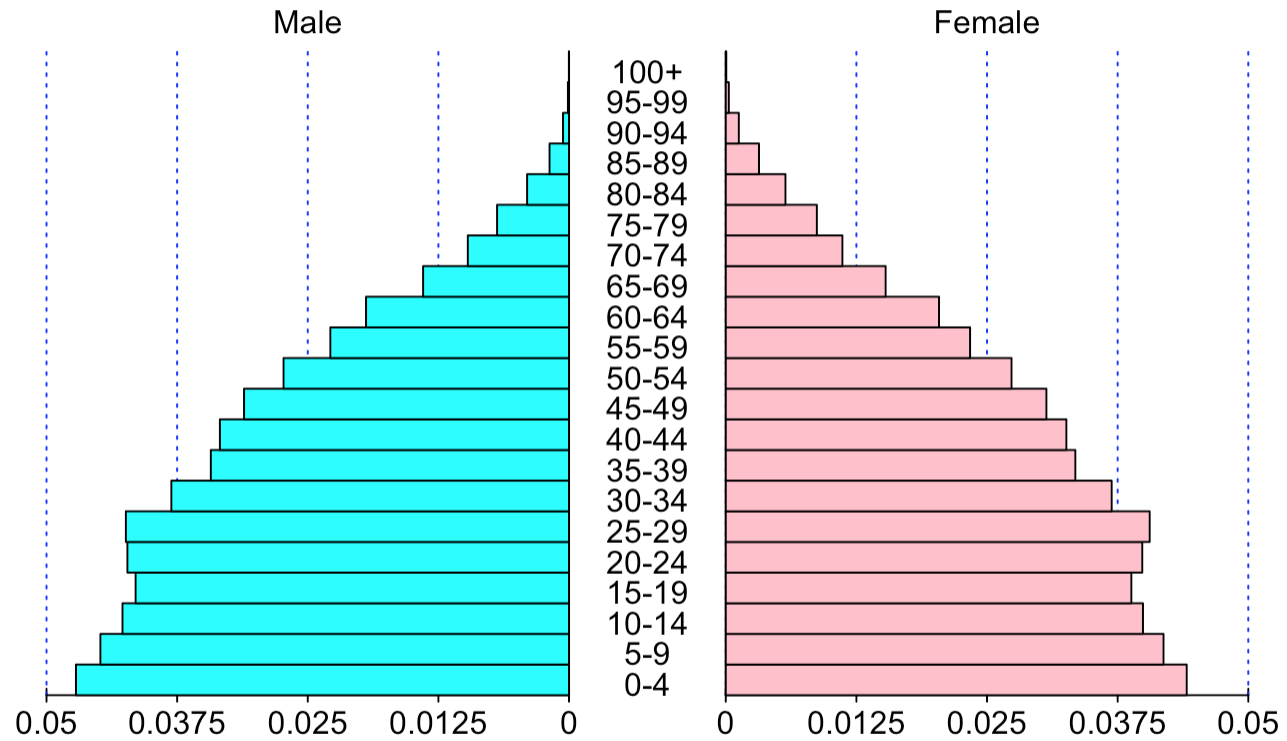
- 5行目, 6行目は, 左側と右側の軸を共通にするための処理. 本筋ではない.
  - 構成割合の最大値を幅にするために, 区切りの良い数値を求めている. 各自, 何をしているか検討してほしい.

```
pop <- pop %>% mutate(share = pop / sum(pop))  
male <- filter(pop, sex == 1)  
female <- filter(pop, sex == 2)  
center <- male$age_grp  
max.share <- ceiling(max(pop$share) * 100) / 100  
laxis <- seq(0, max.share, len = 5)
```

# 人口ピラミッド: 描く

```
pyramids(Left = male$share, Right = female$share, Center = center,  
          Laxis = laxis,  
          Clab = "", Llab = "Male", Rlab = "Female",  
          main = "World(2015)")
```

# World(2015)



# 人口ピラミッド: 関数化

- 複数の国・地域の人口ピラミッドを描くのに、上記のプロセスを繰り返すのは煩雑である.
- $x$ というデータについて標本平均を計算するには、 $1 / \text{length}(x) * \text{sum}(x)$ とするが、これを $\text{mean}(x)$ で計算できるようになっている.
- 人口ピラミッドについても、例えば、`draw_pyramid(loc_id, year, title)`で描くことができれば便利である.
- これを関数化するというが、`loc_id`(国・地域コード)、年次(`year`)、`title`(タイトル)を引数とする関数を作成してみよう.

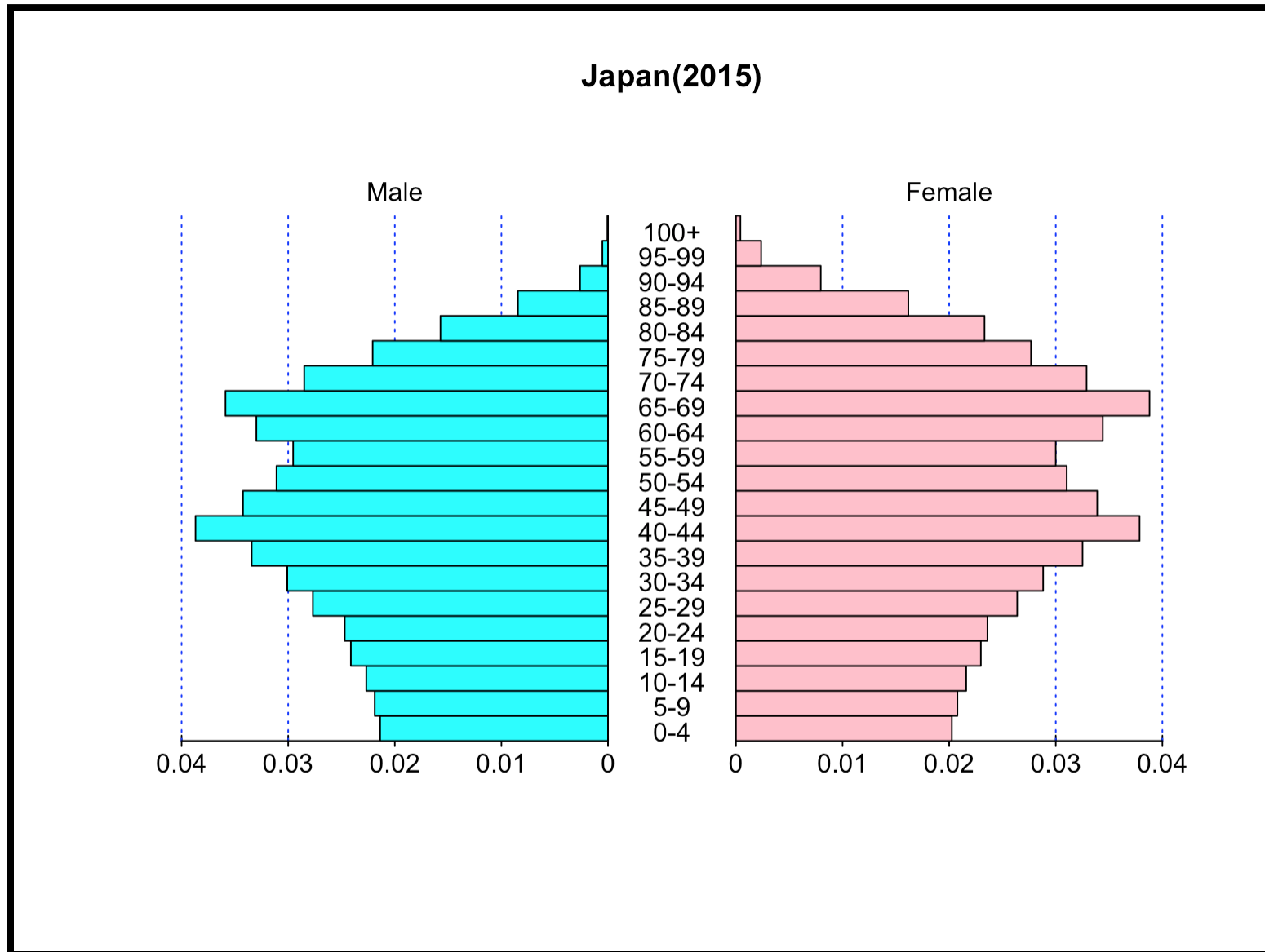


```

draw_pyramid <- function(loc_id, year, title = "Population Pyramid") {
  sql <- paste("SELECT loc_id, year, sex, age_grp, age_start, pop",
              "FROM pop_age_sex_1y WHERE sex IN(1, 2) AND",
              "loc_id =", loc_id, "AND year =", year)
  pop <- dbGetQuery(con, sql)
  pop <- pop %>% mutate(share = pop / sum(pop))
  male   <- filter(pop, sex == 1)
  female <- filter(pop, sex == 2)
  center <- male$age_grp
  max.share <- ceiling(max(pop$share) * 100) / 100
  laxis <- seq(0, max.share, len = 5)
  pyramids(Left = male$share, Right = female$share, Center = center,
           Laxis = laxis, Clab = "", Llab = "Male", Rlab = "Female",
           main = title)
}

```

```
draw_pyramid(392, 2015, "Japan(2015)") # 関数呼び出し
```



# 演習2: 人口ピラミッド

- 先進国と途上国をそれぞれ選択し，人口ピラミッドの経年変化を観察しよう.
- `draw_pyramid`関数を使うこと.