Data Wrangling with DataFrames.jl Cheat Sheet
(for version 0.21.x)

### Create DataFrame

- `DataFrame(x = [1,2,3], y = 4:6, z = 9)`
  - Create data frame with column data from vector, range, or constant.

- `DataFrame($(x=1, y=2), $(x=3, y=4))`  
  - Create data frame from a vector of named tuples.

- `DataFrame("x" => [1,2], "y" => [3,4])`  
  - Create data frame from pairs of column name and data.

- `DataFrame(rand(3,5))`  
  - Create data frame from a matrix.

- `DataFrame()`  
  - Create an empty data frame without any columns.

- `DataFrame(x = Int[], y = Float64[])`  
  - Create an empty data frame with typed columns.

- `DataFrame(mytable)`  
  - Create data frame from any data source that implements Tables.jl Interface.

### Describe DataFrame

- `describe(df)`  
  - Summary stats for all columns.

- `describe(df, :mean, :std)`  
  - Specific stats for all columns.

- `describe(df, :extrema => extrema)`  
  - Apply custom function to all columns.

### Reshape Data - changing layout

- `stack(df, [:sibsp, :parch])`  
  - Stack columns data as rows with new variable and value columns

- `unstack(df, :variable, :value)`  
  - Unstack rows into columns using variable and value columns

### Tidy Data - the foundation of data wrangling

- Each variable is saved in its own column.
- Each observation is saved in its own row.

- In a tidy data set:
  - DataFrames.jl can help you tidy up your data.

- Tidy data makes data analysis easier and more intuitive.

### Select Observations (rows)

- **Function syntax**
  - `first(df, n)`  
    - First `n` rows.
  - `last(df, n)`  
    - Last `n` rows.

- **Indexing syntax**
  - `df[6:10, :]`  
    - Return rows 6 to 10
  - `df[df.sex .== "male", :]`  
    - Return rows having sex equals "male".

- **Filter syntax**
  - `filter(;; sex => ==("male"), df)`  
    - Return data frame with unique rows.
  - `filter(row -> row.sex == "male", df)`  
    - Return rows having sex equals "male". *Note: the first syntax performs better.*

- **Indexing syntax**
  - `df[:, ["name", "age"] ]`  
    - Select a copy of columns.

- **Select columns by index**
  - `select(df, 2:5)`  
    - Select columns by index.
  - `select(df, "^S")`  
    - Select columns by regex.

- **Select columns by regex**
  - `select(df, Not(:age))`  
    - Select all columns except the age column.
  - `select(df, Between(:name, :age))`  
    - Select all columns between name and age columns.

- **Indexing syntax**
  - `df[findfirst(==(30), df.age), :]`  
    - Return first row having age equals 30.
  - `df[findall(==(1), df.pclass), :]`  
    - Return all rows having pclass equals 1.

### Select Variables (columns)

- **Function syntax**
  - `select(df, :sex)`  
  - Select desired column(s).
  - `select(df, [:sex, :age])`  
  - Select columns by index.
  - `select(df, :sex)`  
  - Select columns by regex.

- **Select columns by index**
  - `select(df, 2:5)`  
  - Select columns by index.
  - `select(df, r"^S")`  
  - Select columns by regex.

- **Select all columns except the age column**
  - `select(df, Not(:age))`  
  - Select all columns except the age column.

- **Select all columns between name and age columns**
  - `select(df, Between(:name, :age))`  
  - Select all columns between name and age columns.

### Sort Data

- `sort(df, :age)`  
  - Sort by age

- `sort(df, :age, rev = true)`  
  - Sort by age in reverse order

- `sort(df, [:age, order(:sibsp, rev = true)])`  
  - Sort by in ascending age and descending sibsp order

### Handle Missing Data

- `dropmissing(df)`  
  - Return rows without any missing data.

- `allowmissing(df)`  
  - Allow missing data in column(s).

- `disallowmissing(df)`  
  - Do not allow missing data in column(s).

- `completecases(df)`  
  - Return Bool array with true entries for rows without any missing data.

- `completecases(df, [:age, :sex])`  
  - Return Bool array with true entries for rows without any missing data.

### View Metadata

- `names(df)`  
  - Column names.

- `propertynames(df)`  
  - Index number of a column.

- `ncol(df)`  
  - Number of columns.

### Indexing syntax

- `df[6:10, :]`  
  - Return rows 6 to 10

- `df[df.sex .== "male", :]`  
  - Return rows having sex equals "male".

- `filter(;; sex => ==("male"), df)`  
  - Return data frame with unique rows.

- `filter(row -> row.sex == "male", df)`  
  - Return rows having sex equals "male". *Note: the first syntax performs better.*

- `select(df, 2:5)`  
  - Select columns by index.

- `select(df, r"^S")`  
  - Select columns by regex.

- `select(df, Not(:age))`  
  - Select all columns except the age column.

- `select(df, Between(:name, :age))`  
  - Select all columns between name and age columns.

- `df[findfirst(==(30), df.age), :]`  
  - Return first row having age equals 30.

- `df[findall(==(1), df.pclass), :]`  
  - Return all rows having pclass equals 1.

- `Mutation: use unique! or filter!`

- `Mutation: use select!`

- `Mutation: use dropmissing!, allowmissing!, or disallowmissing!`

---

This cheat sheet is inspired by the data wrangling cheat sheets from RStudio and pandas. Examples are based on the Kaggle Titanic data set. Created by Tom Kwong, September 2020. v0.21 rev3
Cumulative and Moving Stats

**Cumulative Stats**
- `select(df, :x => cumsum)`
- Cumulative sum of column `x`.
- `select(df, :x => cumprod)`
- Cumulative product of column `x`.
- `select(df, :x => accumulate(min, v))` and `select(df, :x => accumulate(max, v))`
- Cumulative minimum/maximum of column `x`.

**Moving Stats (a.k.a Rolling Stats)**
- `select(df, :x => (v -> runmean(v, n))`
- Moving mean of column `x` with window size `n`.
- `select(df, :x => (v -> runmedian(v, n))`
- Moving median of column `x` with window size `n`.
- `select(df, :x => (v -> runmin(v, n))`
- Moving minimum of column `x` with window size `n`.
- `select(df, :x => (v -> runmax(v, n))`
- Moving maximum of column `x` with window size `n`.

*The run* functions (and more) are available from `RollingFunctions.jl` package.

Aggregating variables
- `combine(df, :survived => sum)`
- Apply a function to a column; optionally assign column name.
- `combine(df, :age => (x -> mean(skipmissing(x))))`
- Apply an anonymous function to a column.
- `combine(df, [:parch, :sibsp] => maximum)`
- Apply a function to multiple columns using broadcasting syntax.

Adding variables with aggregation results
- `transform(df, :fare => mean => :average_fare)`
- Add a new column that is populated with the aggregated value.
- `select(df, :name, :fare, :fare => mean => :average_fare)`
- Select any columns and add new ones with the aggregated value.

Adding variables by row
- `transform(df, [:parch, :sibsp] => ByRow(+) => :relatives)`
- Add new columns by applying a function over existing column(s).

Tips: Use `skipmissing` function to remove missing values.

Cumulative and Moving Stats

**Summarize Data**

**Group Data Sets**
- `gdf = groupby(df, :pclass)`
- Group data frame by one or more columns.

```julia
keys(gdf)
# Get the keys for looking up SubDataFrame's in the group.
```

```julia
gdf[(1,)]
# Look up a specific group using a tuple of key values.
```

```julia
combine(gdf, :survived => sum)
# Apply a function over a column for every group.
```

```julia
Tips: You can also use these functions to add summarized data to all rows:
- select
- select!
- transform
- transform!
```

```julia
combine(gdf, AsTable(:) => t -> sum(t.parch .+ t.sibsp))
# Apply a function to each SubDataFrame in the group and combine the single value results.
```

Ranking and Lead/Lag Functions

**Ranking Functions**
- `select(df, :x => ordinalrank)`
- `select(df, :x => comperrank)`
- `select(df, :x => denserrank)`
- `select(df, :x => tiedrank)`

*The rank functions come from `StatsBase.jl` package.

**Lead/Lag Functions**
- `select(df, :x => lead)`
- # shift up
- `select(df, :x => lag)`
- # shift down

*The lead and lag functions come from `ShiftedArrays.jl` package.

Build Data Pipeline

@pipe df |> filter(:sex => "=\("male\"\)\) |> groupby(_, :pclass) |> combine(_, :age => mean)

*The @pipe macro comes from `Pipe.jl` package. Underscores are automatically replaced by return value from the previous operation before the |> operator.*