Data Cleaning and Preprocessing

George Mwangi

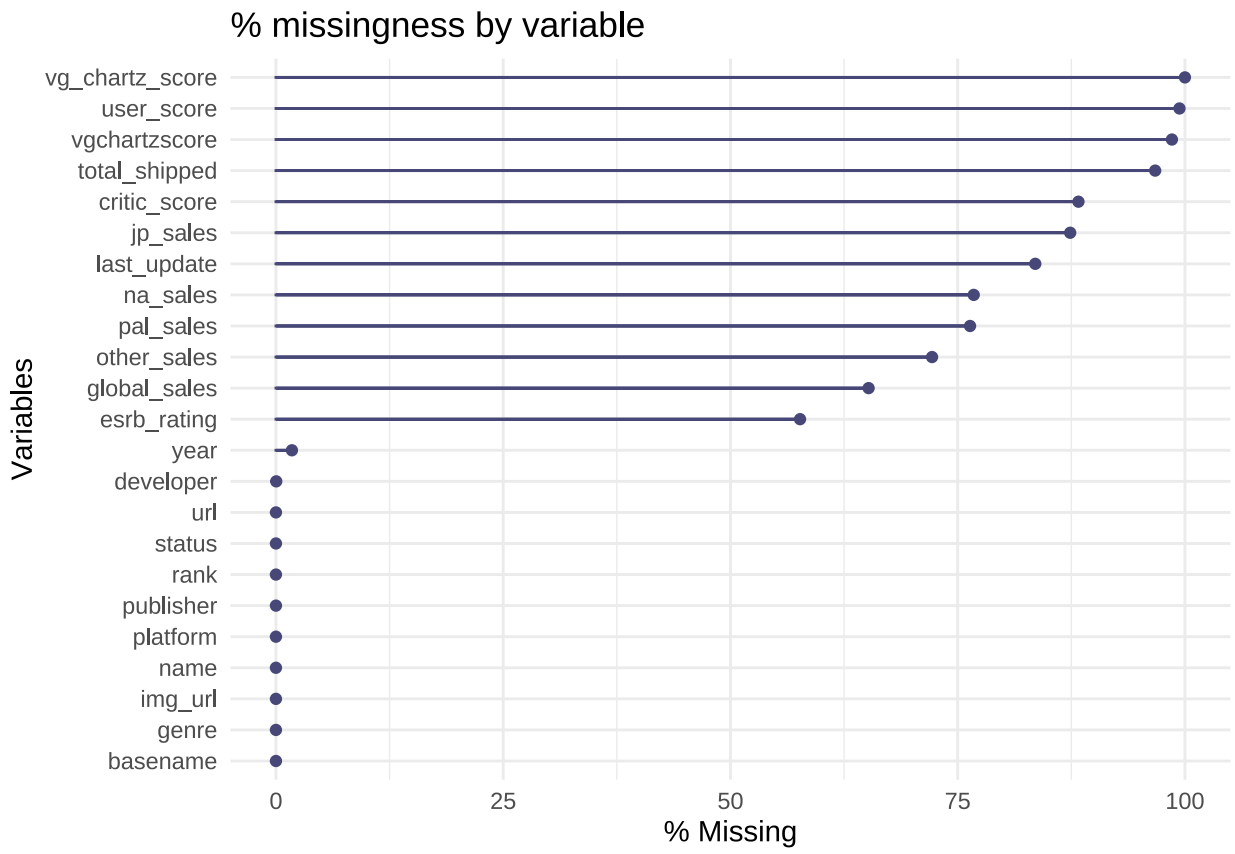
2023-03-06

Read Data and Load Packages

```
# load important packages
pacman::p_load(tidyverse, naniar, dlookr, DataExplorer, janitor, tidymodels)


#read in the csv file and clean the column names
vgsales_df <- read_csv("datasets/vgsales-12-4-2019.csv", show_col_types = F) %>%
  # start with cleaning column names into a consistent convention
  clean_names()
```

```
#this code is used to show the percentage of missing values in each variable
gg_miss_var(vgsales_df,show_pct = T)+
  labs(title = "% missingness by variable")
```




```
# create a report using the data in vgsales_df, output the report in html format, name the report vgsal
create_report(
  data = vgsales_df,
  output_format = "html_document",
  output_file = "vgsales_eda.html",
  output_dir = "./"
)
```

```
# find out the count of genre by year(which year which genre sold more)
vgsales_df %>%
  count(year, genre, sort = T) %>%
  # filter first 10 results
  slice_head(n = 10)
```

```
## # A tibble: 10 x 3
##     year genre         n
##    <dbl> <chr>     <int>
##  1  2014 Misc       1273
##  2  2013 Misc        936
##  3  2009 Misc        844
##  4  2012 Misc        721
##  5  2010 Misc        673
##  6  2011 Misc        671
##  7  2009 Action      572
##  8  2008 Misc        496
```

```
##  9  2009 Adventure   481
## 10  2011 Action      480
```

```r
# prepare data for modeling with sales as the target variable


vgsales_df_clean <- vgsales_df %>%
  # remove columns that are not needed for analysis
  # please refer to the above eda report on the missing variable part
  select(
    -c(basename, url, img_url, vg_chartz_score, vgchartzscore,
       user_score, total_shipped, critic_score, last_update)
    ) %>%
   # change character columns to factors
  mutate_if(is.character, as.factor) %>%
  # reduce factor levels of genre, esrb_rating, platform, developer
  # also refer to the charts in the eda report to see the most common levels in each factor variable
  mutate(
    genre = fct_lump_n(genre, n = 10, other_level = "Other"),
    esrb_rating = fct_lump_n(esrb_rating, n = 4, other_level = "Other"),
    platform = fct_lump_n(platform, n = 10, other_level = "Other"),
    developer = fct_lump_n(developer, n = 10, other_level = "Other"),
    publisher = fct_lump_n(publisher, n = 10, other_level = "Other")
    ) %>%
  # I want to create a variable (sales) that combines all sales features with sales aspect
  # before that we should replace NA's with zero to for mutate to work
  replace_na(list(global_sales = 0,
                  na_sales = 0,
                  pal_sales = 0,
                  jp_sales = 0,
                  other_sales = 0)) %>%
  # create a variable sales
  mutate(sales = global_sales + pal_sales + jp_sales + other_sales) %>%
  # deselect the variables that made sales,
  # also deselect status because it contains only one value input(1) confirm with `count(vgsales_df, st
  # name and rank also dont seem to add value our model
  select(-c(other_sales, jp_sales, pal_sales, na_sales, global_sales, status, rank, name)) %>%
  # remove rows with missing values
  # for simplicity, am choosing removing the rows with NA's
  # we could the knn algorithm to impute NA's in the feature engineering recipe
  drop_na() %>%
   # remove duplicate rows
  distinct()
```

The above cleaning pipeline can be changed to a function to clean future sorces of data

```r
# create a deep eda web report of the cleaned data with this code
# use the arrows in each analysis part to see the graphs and statistics
eda_web_report(
  .data = vgsales_df_clean,
  target = "sales",
  output_file = "vgsales_deep_dive.html",
  output_dir = "./"
)
```

```r
# split data into training and testing sets, allocating 70% to the training set
sales_split <- initial_split(vgsales_df_clean, prop = .70)

# print sales_split
sales_split
```

```
## <Training/Testing/Total>
## <11992/5140/17132>
```

```r
# prepare a feature engineering pipeline
my_recipe <- recipe(sales ~ ., data = training(sales_split)) %>%
  # center and scale the year variable
  step_normalize(year) %>%
  # create dummy variables for all the nominal variables with one-hot encoding
  step_dummy(all_nominal(), one_hot = TRUE)

# print recipe
my_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          6
##
## Operations:
##
## Centering and scaling for year
## Dummy variables from all_nominal()
```

```r
# Preprocess training data
training_data_processed <- my_recipe %>%
  # train recipe with training data
  prep(training(sales_split)) %>%
  # setting new data to NULL returns the preprocessed training data
  bake(new_data = NULL)


# preprocess testing data
testing_data_processed <- my_recipe %>%
  # train recipe with training data
  prep(training(sales_split)) %>%
  bake(new_data = testing(sales_split))

# we can now proceed to specify a model to fit the data with parsnip package
```