0setup.R

```
## copy and paste IMPORTS at the beginning of your scripts ----
# ## IMPORTS ----
# rm(list = ls()) #clean environment
# graphics.off() #clean plots
# setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
# temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
# source("0setup.R", local = temp_env)
# games <- temp_env$setup()
# rm(temp_env) #delete temporary environment after data has been loaded
#
# #select variables for analysis
# gamesc <- games %>%
#   select(Average.playtime.forever, Estimated.owners,
#          Peak.CCU, rating, Price,
#          Recommendations, Required.age,
#          Positive, Negative,
#          total_reviews, positive_ratio)

## list of packages needed ----
packages <- c(
  "tidyverse", "DataExplorer", "dplyr", "ggplot2",
  "leaps", "glmulti", "nlme", "nnet", "pscl", "car",
  "rlang", "corrplot", "ggcorrplot", "lmtest",
  "nortest"
)

## subfunctions ----
setup <- function() {
```

```
  # reset stored values
  rm(list = ls())
  # set working dir to script's location
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
  install_and_load(packages)
  games <- load_and_clean_games()
  return(games)
}

install_and_load <- function(packages) {
  # Find missing packages
  missing_packages <- packages[!(packages %in% installed.packages()[, "Package"])]

  # Install missing packages
  if (length(missing_packages) > 0) {
	install.packages(missing_packages, dependencies = TRUE)
  }

  # Load all packages
  invisible(lapply(packages, library, character.only = TRUE))
}

load_and_clean_games <- function() {
  filepath <- "../steam_data/games.csv"
  games <- read.csv(filepath)
  games <- games %>% filter(Average.playtime.forever>0 & Peak.CCU>0)
  games[is.na(games)] <- 0
  total_reviews <- games$Positive + games$Negative
  positive_ratio <- (games$Positive / total_reviews) * 100  # ratio of positive reviews

  # create variable total_reviews
```

```
  games <- games %>% mutate(total_reviews = mapply(create_total_reviews, Positive,
Negative))

  # create variable positive_ratio
  games <- games %>% mutate(positive_ratio = mapply(create_positive_ratio, Positive,
Negative))

  # create variable rating
  games <- games %>% mutate(rating = mapply(create_rating, Positive, Negative))
  # define rating category order
  rating_levels <- c("Overwhelmingly Positive", "Very Positive", "Positive",
                     "Mostly Positive", "Mixed",
                     "Mostly Negative", "Negative", "Very Negative",
                     "Overwhelmingly Negative", "Not enough reviews")
  # rating as factor
  games <- games %>%
        mutate(rating = factor(rating, levels = rating_levels, ordered = TRUE))
  games <- na.omit(games)

  #estimated owner as factor
  games$Estimated.owners <- as.factor(games$Estimated.owners)
  games$Estimated.owners <-
        fct_recode(games$Estimated.owners,
        "0-20k" = "0 - 20000",
        "20k-50k" = "20000 - 50000",
        "50k-100k" = "50000 - 100000",
        "100k-200k" = "100000 - 200000",
        "200k-500k" = "200000 - 500000",
        "500k-1M" = "500000 - 1000000",
        "1M-2M" = "1000000 - 2000000",
        "2M-5M" = "2000000 - 5000000",
        "5M-10M" = "5000000 - 10000000",
```

```
            "10M-20M" = "10000000 - 20000000",
            "20M-50M" = "20000000 - 50000000",
            "50M-100M" = "50000000 - 100000000",
            "100M-200M" = "100000000 - 200000000"
            )
  games$Estimated.owners <- fct_relevel(
            games$Estimated.owners,
            "0-20k", "20k-50k", "50k-100k", "100k-200k", "200k-500k",
            "500k-1M", "1M-2M", "2M-5M", "5M-10M", "10M-20M",
            "20M-50M", "50M-100M", "100M-200M"
  )
  return(games)
}

create_rating <- function(Positive, Negative) {
  total_reviews <- Positive + Negative
  positive_ratio <- (Positive / total_reviews) * 100  # Ratio des avis positifs

  if (total_reviews >= 500) {
            if (positive_ratio >= 95 && positive_ratio <= 100) {
            return("Overwhelmingly Positive")
            } else if (positive_ratio >= 80 && positive_ratio < 95) {
            return("Very Positive")
            } else if (positive_ratio >= 70 && positive_ratio < 80) {
            return("Mostly Positive")
            } else if (positive_ratio >= 40 && positive_ratio < 70) {
            return("Mixed")
            } else if (positive_ratio >= 20 && positive_ratio < 40) {
            return("Mostly Negative")
            } else if (positive_ratio >= 0 && positive_ratio < 20) {
            return("Overwhelmingly Negative")
```

```
    }
}

else if (total_reviews >= 50 && total_reviews <= 499) {
    if (positive_ratio >= 80 && positive_ratio <= 100) {
    return("Very Positive")
    } else if (positive_ratio >= 70 && positive_ratio < 80) {
    return("Mostly Positive")
    } else if (positive_ratio >= 40 && positive_ratio < 70) {
    return("Mixed")
    } else if (positive_ratio >= 20 && positive_ratio < 40) {
    return("Mostly Negative")
    } else if (positive_ratio >= 0 && positive_ratio < 20) {
    return("Very Negative")
    }
}

else if (total_reviews >= 10 && total_reviews <= 49) {  # 10-49 reviews
    if (positive_ratio >= 80 && positive_ratio <= 100) {
    return("Positive")
    } else if (positive_ratio >= 70 && positive_ratio < 80) {
    return("Mostly Positive")
    } else if (positive_ratio >= 40 && positive_ratio < 70) {
    return("Mixed")
    } else if (positive_ratio >= 20 && positive_ratio < 40) {
    return("Mostly Negative")
    } else if (positive_ratio >= 0 && positive_ratio < 20) {
    return("Negative")
    }
}
```

```
  else {
        return("Not enough reviews")
  }
}

create_total_reviews <- function(Positive, Negative) {
  total_reviews <- Positive + Negative
  return(total_reviews)
}

create_positive_ratio <- function(Positive, Negative) {
  total_reviews <- Positive + Negative
  positive_ratio <- (Positive / total_reviews) * 100
  return(positive_ratio)
}
```

1.univariate_tests.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

# select variables for analysis
cleaned_games <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

## average playtime forever
boxplot(cleaned_games$Average.playtime.forever,
        main = "Distribution du temps de jeu moyen",
        ylab = "Temps de jeu moyen (minutes)")

## recommendations
boxplot(cleaned_games$Recommendations,
        main = "Distribution du nombre de recommendations",
        ylab = "Nombre de recommendations")

## negative
```

```
boxplot(cleaned_games$Negative,
        main = "Distribution du nombre d'avis négatifs",
        ylab = "Nombre d'avis négatifs")

## positive
boxplot(cleaned_games$Positive,
        main = "Distribution du nombre d'avis positifs",
        ylab = "Nombre d'avis positifs")

## price
boxplot(cleaned_games$Price,
        main = "Distribution du prix des jeux",
        ylab = "Prix en dollars")

## peak ccu
boxplot(cleaned_games$Peak.CCU,
        main = "Distribution des maximums de joueurs simultanés atteint",
        ylab = "Nombre de joueurs simultanés")

## required.age
# define age category
cleaned_games$age_Category <- ifelse(cleaned_games$Required.age < 12 |
cleaned_games$Required.age == "", "Tout public",
                  ifelse(cleaned_games$Required.age >= 12 &
cleaned_games$Required.age < 16, "+12",
                        ifelse(cleaned_games$Required.age <= 16 &
cleaned_games$Required.age < 18, "+16", "+18")))
# barplot
bar_plot_age <- ggplot(cleaned_games, aes(x = age_Category)) +
  geom_bar(fill = "steelblue") +
  labs(title = "Fréquence des catégories d'âge",
        x = "Âge requis",
```

```
        y = "Nombre de jeux") +
  theme_minimal()
print(bar_plot_age)

## estimated owner

bar_plot_owner <- ggplot(cleaned_games, aes(x = Estimated.owners)) +
  geom_bar(fill = "steelblue") +
  labs(title = "Distribution of Estimated Owners",
        x = "Estimated Owners",
        y = "Number of Games") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Rotate labels for readability
print(bar_plot_owner)
```

2.univariate_tests_log.R

```r
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
cleaned_games <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

## Peak.CCU density log10 +1 ----
ggplot(cleaned_games, aes(x = Peak.CCU + 1)) +
  geom_density(fill = "blue", alpha = 0.3) +
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Density Plot of Peak CCU (log10 scale)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))

## price density log10 +1 ----
ggplot(cleaned_games, aes(x = Price + 1)) +
  geom_density(fill = "blue", alpha = 0.3) +
```

```
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Density Plot of Price (log10 scale)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))

## Positive density log10 +1 ----
ggplot(cleaned_games, aes(x = Positive + 1)) +
  geom_density(fill = "blue", alpha = 0.3) +
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Density Plot of number of positive reviews (log10 scale)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))

## Negative density log10 +1 ----
ggplot(cleaned_games, aes(x = Negative + 1)) +
  geom_density(fill = "blue", alpha = 0.3) +
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Density Plot of number of negative reviews (log10 scale)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))

## Recommendations density log10 +1 ----
ggplot(cleaned_games, aes(x = Recommendations + 1)) +
  geom_density(fill = "blue", alpha = 0.3) +
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Density Plot of number of recommendations (log10 scale)") +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))

## Average playtime forever log10 +1 ----
ggplot(cleaned_games, aes(x = Average.playtime.forever + 1)) +
```

```r
geom_density(fill = "blue", alpha = 0.3) +
scale_x_log10() +
theme_minimal() +
ggtitle("Density Plot of Average playtime forever (log10 scale)") +
theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14))
```

3.correlation_matrix.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
cleaned_games <- games %>%
        select(Average.playtime.forever, Estimated.owners,
        Peak.CCU, rating, Price,
        Recommendations, Required.age,
        Positive, Negative)

####correlation matrix ####

# keep numerical variables
numeric_vars <- cleaned_games[, sapply(cleaned_games, is.numeric)]

#correlation matrix
cor_matrix <- cor(numeric_vars, method = "spearman", use = "pairwise.complete.obs")
print(cor_matrix)

#illustration

corrplot(cor_matrix, method = "color", type = "upper", order = "hclust",
```

```
    col = colorRampPalette(c("blue", "white", "red"))(200), tl.col = "black")

#heatmap

x11()
ggcorrplot(cor_matrix, method = "square", type = "lower", lab = TRUE)



#strong correlations : Positive - Recommendations (0.83) / Negative - Positive (0.80)
#Moderate correlations: Peak.CCU - Recommendations (0.59) / Negative - Recommendations (0.67) / Peak.CCU - Positive (0.66)
#Weaker correlations: Price - Average.playtime.forever (0.30) / Price - Positive (0.27)

# keep numerical variables

numeric_vars <- cleaned_games[, sapply(cleaned_games, is.numeric)]

# Create the correlation matrix
correlation_matrix <- cor(numeric_vars)
corrplot(correlation_matrix, method = "circle")

# Plot the correlation matrix without the mirror image
corrplot(correlation_matrix, method = "circle", type = "upper")
```

4.rating.R

```
####  outdated file
#### below functions were added to 0setup instead

## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

##create rating variable ----
create_rating <- function(Positive, Negative) {
  total_reviews <- Positive + Negative
  positive_ratio <- (Positive / total_reviews) * 100  # Ratio des avis positifs

  if (total_reviews >= 500) {
         if (positive_ratio >= 95 && positive_ratio <= 100) {
         return("Overwhelmingly Positive")
```

```
        } else if (positive_ratio >= 80 && positive_ratio < 95) {
        return("Very Positive")
        } else if (positive_ratio >= 70 && positive_ratio < 80) {
        return("Mostly Positive")
        } else if (positive_ratio >= 40 && positive_ratio < 70) {
        return("Mixed")
        } else if (positive_ratio >= 20 && positive_ratio < 40) {
        return("Mostly Negative")
        } else if (positive_ratio >= 0 && positive_ratio < 20) {
        return("Overwhelmingly Negative")
        }
}

else if (total_reviews >= 50 && total_reviews <= 499) {
        if (positive_ratio >= 80 && positive_ratio <= 100) {
        return("Very Positive")
        } else if (positive_ratio >= 70 && positive_ratio < 80) {
        return("Mostly Positive")
        } else if (positive_ratio >= 40 && positive_ratio < 70) {
        return("Mixed")
        } else if (positive_ratio >= 20 && positive_ratio < 40) {
        return("Mostly Negative")
        } else if (positive_ratio >= 0 && positive_ratio < 20) {
        return("Very Negative")
        }
}

else if (total_reviews >= 10 && total_reviews <= 49) {  # 10-49 reviews
        if (positive_ratio >= 80 && positive_ratio <= 100) {
        return("Positive")
        } else if (positive_ratio >= 70 && positive_ratio < 80) {
```

```
            return("Mostly Positive")
            } else if (positive_ratio >= 40 && positive_ratio < 70) {
            return("Mixed")
            } else if (positive_ratio >= 20 && positive_ratio < 40) {
            return("Mostly Negative")
            } else if (positive_ratio >= 0 && positive_ratio < 20) {
            return("Negative")
            }
  }

  else {
            return("Not enough reviews")
  }
}
gamesc <- gamesc %>% mutate(rating = mapply(create_rating, Positive, Negative))

# Définir l'ordre des niveaux du plus positif au plus négatif
rating_levels <- c("Overwhelmingly Positive", "Very Positive", "Positive",
                   "Mostly Positive", "Mixed",
                   "Mostly Negative", "Negative", "Very Negative",
                   "Overwhelmingly Negative", "Not enough reviews")

# Convertir rating en facteur avec un ordre défini
gamesc <- gamesc %>%
  mutate(rating = factor(rating, levels = rating_levels, ordered = TRUE))

# Réorganiser les colonnes : Positive, Negative et rating en 2e, 3e et 4e position
gamesc <- gamesc %>%
  select(1, Positive, Negative, rating, everything())

effectifs <- gamesc %>% count(rating)
```

5.bivarriate_tests.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
  select(Average.playtime.forever, Estimated.owners,
            Peak.CCU, rating, Price,
            Recommendations, Required.age,
            Positive, Negative,
            total_reviews, positive_ratio)

## normality tests for quantitative variables (bivar tests) ----
# select variables to test
names(gamesc)
numeric_cols <- gamesc %>%
  select(Average.playtime.forever, Peak.CCU, Price, Recommendations, Required.age,
            Positive, Negative, total_reviews, positive_ratio)

# log transformation
numeric_cols_log <- numeric_cols %>%
  mutate(across(everything(), ~ log10(. + 1)))
```

```
# lillie tests
lillie_results <- lapply(numeric_cols_log, lillie.test)

# store results in a dataframe
lillie_table <- data.frame(
  "p-value lillie tests" = sapply(lillie_results, function(x) x$p.value)
)
print(lillie_table)

## spearman because there is no normality ----
names(gamesc)
numeric_cols <- gamesc %>%
  select(Average.playtime.forever, Peak.CCU, Price, Recommendations, Required.age,
            Positive, Negative, total_reviews, positive_ratio)
names(numeric_cols)

# spearman for every combination with average.playtime.forever
spearman_results <- list()
rho_values <- list()  # store rho values in a list

for (var in names(numeric_cols)[-1]) {  # exclude average playtime
  test_result <- suppressWarnings(cor.test(numeric_cols$Average.playtime.forever,
numeric_cols[[var]], method = "spearman"))

  # store rho and p values in lists
  spearman_results[[var]] <- test_result$p.value
  rho_values[[var]] <- test_result$estimate
}

# store tests results in dataframe
spearman_table <- data.frame(
```

```
  "p-value spearman" = unlist(spearman_results),
  "rho (Spearman)" = unlist(rho_values)
)

print(spearman_table)



## kruskal wallis because no normality and data is not paired ----
names(gamesc)
# target variable
y <- gamesc$Average.playtime.forever

# categorical predictors
quali <- gamesc %>%
  select(Estimated.owners, rating)

# empty table to store results
kruskal_table <- data.frame(
  Test = character(),
  H_statistic = numeric(),
  p_value = numeric(),
  stringsAsFactors = FALSE
)

# kruskal test for all the variables of the list
for (var in names(quali)) {
  test_result <- kruskal.test(y ~ gamesc[[var]], data = gamesc)

  # add results to table
  kruskal_table <- rbind(kruskal_table, data.frame(
           Test = var,  # display variable name instead of test name
```

```
            H_statistic = test_result$statistic,
            p_value = test_result$p.value
  ))
}

print(kruskal_table)

## display all bivariate tests results ----
print(lillie_table)
print(spearman_table)
print(kruskal_table)
```

6.rating_test.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after
import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

## tables ----

# frequency
table(gamesc$rating)
# percentages
prop.table(table(gamesc$rating))*100

## graph ----

ggplot(gamesc, aes(x = fct_rev(rating))) +
         geom_bar(aes(y = ..prop.., group = 1), fill = "steelblue", color = "black") +
```

```
theme_minimal() +
labs(title = "Proportion of Game ratings",
x = "rating",
y = "Proportion") +
coord_flip() +
scale_y_continuous(labels = scales::percent_format(accuracy = 1))
```

7.lm_and_hypotheses.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

# Function to create a linear model
create_lm <- function(dataset, Y, X, categories) {
         if (length(categories) == 0) {
         formula <- as.formula(paste(Y, "~", paste(X, collapse = "+")))
         } else {
         formula <- as.formula(paste(Y, "~", paste(c(X, categories), collapse = "+")))
         }
         model <- lm(formula = formula, data = dataset)
         return(model)
}
```

```
# Function to graph and test model hypotheses
check_lm_hypotheses <- function(model, data) {
  cat("Vérification des hypothèses pour le modèle :", deparse(model$call), "\n\n")

  # hypothese 1: relation de linearite entre Y et X
  # en cas de linerite le nuage de points doit etre centre autour de 0 sans motif evident
  plot(model, which = 1, main = "1. Résidus vs valeurs ajustées")

  # hypothese 2: homoscedasticite des erreurs
  # ecart type constant peu importe la valeur ajustee
  # entonnoir = heteroscedasticite
  plot(model, which = 3, main = "2.1. Écarts à l'effet de levier")

  # residus absolus vs valeurs ajustees
  # si c'est croissant/decroissant, la variance n est pas constante
  data$residu_abs <- abs(residuals(model))
  data$ajuste <- fitted(model)
  ggplot(data, aes(x = ajuste, y = residu_abs)) +
        geom_point(alpha = 0.4) +
        geom_smooth(method = "loess", col = "red") +
        labs(title = "2.2. Hétéroscédasticité : résidus absolus vs ajustés",
        x = "Valeurs ajustées", y = "Résidus absolus") +
        theme_minimal() -> p1
  print(p1)

  # hypothese 3: independance des erreurs
  # test de Durbin-Watson : attend une valeur proche de 2
  # si proche de 0 : autocorrelation positive. si > 2.5 : autocorrelation negative
  cat("\nTest de Durbin-Watson (attendu ≈ 2) :\n")
  print(dwtest(model))
  dwtest(model, alternative = c("two.sided"))
```

```
# ACF : si les barres depassent, il y a autocorrelation des erreurs
acf(residuals(model), main = "3. ACF des résidus")

# hypothese 4: normalite des erreurs
# ca doit suivre la ligne, sinon residus anormaux
plot(model, which = 2, main = "4. QQ-plot des résidus")
residus <- model$residuals
#boxplot des residus
boxplot(residus, main = "Boxplot des résidus")
#histogramme "naif"
hist(residus[residus < quantile(residus, 1)], breaks = 50,
     main = "Histogramme des résidus",
     xlab = "Résidus", col = "green", border = "green")
#histogramme plus lisible en retirant le top 1% qui casse tout
hist(residus[residus < quantile(residus, 0.99)], breaks = 50,
     main = "Histogramme des résidus (sans top 1%)",
     xlab = "Résidus (censurés à 99%)", col = "blue", border = "blue")

# hypothese 5 multicolinearite
# VIF (Variance Inflation Factor) : mesure le lien entre les variables explicatives
# VIF > 5 = multicolinearite moderee VIF > 10 = multicolinearite severe
cat("\nVIF (Variance Inflation Factor) :\n")
vif_vals <- vif(model)
print(vif_vals)
cat("\nVariables avec VIF > 5 :\n")
print(names(vif_vals[vif_vals > 5]))

# observations influentes: distances de Cook
# indique si certaines observations influencent beaucoup le modele
# attention aux points au dessus de la ligne rouge
cooks <- cooks.distance(model)
```

```r
  seuil <- 4 / nrow(data)
  # cat("\nObservations influentes (Cook > 4/n) :\n")
  # influents <- which(cooks > seuil)
  # print(influents)

  plot(cooks, type = "h",
       main = "6. Distance de Cook avec seuil 4/n",
       ylab = "Distance de Cook", xlab = "Index de l'observation")
  abline(h = seuil, col = "red", lty = 2, lwd = 2)
  legend("topright", legend = paste0("Seuil = 4/n ≈ ", round(seuil, 5)),
       col = "red", lty = 2, lwd = 2)
}

# Function to create a GLS model (not used)
create_gls <- function(dataset, Y, X, categories = c(), correlation_struct = corCompSymm()) {
       # Build the formula for the model
       predictors <- if (length(categories) == 0) X else c(X, categories)
       formula_str <- paste(Y, "~", paste(predictors, collapse = "+"))

       # Convert the formula string to a formula object
       formula <- as.formula(formula_str)

       # Remove rows with NA to avoid fitting issues
       dataset_clean <- na.omit(dataset)

       # Fit GLS model with optional correlation structure
       modele.gls <- gls(formula,
              data = dataset_clean,
              correlation = correlation_struct)

       return(modele.gls)
```

```r
}

# Fonction pour appliquer les transformations sur une liste de variables
apply_transformations <- function(data, variables) {
	for (var in variables) {
	# Log transformation: log10(x + 1) to avoid log(0)
	data[[var]] <- log10(data[[var]] + 1)

	# sqrt
	# data[[var]] <- sqrt(data[[var]])

	# Standardization: (x - mean) / sd
	# data[[var]] <- scale(data[[var]], center = TRUE, scale = TRUE)

	# Normalization: (x - min) / (max - min)
	# data[[var]] <- (data[[var]] - min(data[[var]])) / (max(data[[var]]) - min(data[[var]]))
	}
	return(data)
}

# Fonction pour détecter les points trop influents
detect_cook <- function(model, threshold = 4 / nrow(model$model)) {
  cooks <- cooks.distance(model)
  which(cooks > threshold)
}

# Fonction pour détecter les résidus trop gros
detect_large_residuals <- function(model, threshold = 3) {
  rstudent_res <- rstudent(model)
  which(abs(rstudent_res) > threshold)
}
```

```
# Détection des outliers dans les données (z-score > 3)
detect_outliers_data <- function(dataset, threshold = 3) {
  numeric_data <- dataset[sapply(dataset, is.numeric)]
  z_scores <- scale(numeric_data)
  which(apply(abs(z_scores) > threshold, 1, any))
}

# Nettoyage global
clean_model <- function(model, dataset) {
  idx_cook <- detect_cook(model)
  idx_resid <- detect_large_residuals(model)
  idx_outliers <- detect_outliers_data(dataset)
  idx_to_remove <- unique(c(idx_cook, idx_resid, idx_outliers))
  cleaned_data <- dataset[-idx_to_remove, ]
  return(list(data = cleaned_data, removed = idx_to_remove))
}

## first model with only numeric variables ----
names(gamesc)
Y <- "Average.playtime.forever"
X <- c("Peak.CCU", "Positive", "Negative", "Recommendations", "Price", "Required.age")
categories <- c("Estimated.owners")
variables <- c(Y, X, categories)  # Combined variables list
print(variables)
model <- create_lm(gamesc, Y, X, categories)

summary(model)

# R2 too low
```

```r
## second model with log transformation to be closer to linearity ----
names(gamesc)
Y <- "Average.playtime.forever"
X <- c("Peak.CCU", "Positive", "Negative", "Recommendations",
       "Price", "Required.age")
categories <- c("Estimated.owners")
variables <- c(Y, X, categories)  # Combined variables list
print(variables)
variables_to_transform <- c("Average.playtime.forever","Peak.CCU",
                  "Positive", "Negative", "Recommendations", "Price")
#tranformation with log
gamesc_log <- apply_transformations(gamesc, variables_to_transform)
model_log <- create_lm(gamesc_log, Y, X, categories)
summary(model_log)

## third model without outliers, high influence point, and extreme errors ----
cleaning <- clean_model(model_log, gamesc_log)
gamesc_log_clean <- cleaning$data
model_log_clean <- create_lm(gamesc_log_clean, Y, X, categories)
summary(model_log_clean)
cat("Nombre de points retirés :", length(cleaning$removed), "\n")

## hypotheses check ----
check_lm_hypotheses(model, gamesc)
check_lm_hypotheses(model_log, gamesc_log)
check_lm_hypotheses(model_log_clean, gamesc_log_clean)
```

8.lm_selection.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
  select(Average.playtime.forever, Estimated.owners,
         Peak.CCU, rating, Price,
         Recommendations, Required.age,
         Positive, Negative,
         total_reviews, positive_ratio)

# function that finds the best linear model for given criteria
select_model_glmulti <- function(data, crit = "aic", level = 1) {
  formula <- as.formula("Average.playtime.forever ~ .")
  result <- glmulti(formula, data = data, level = level,
                    fitfunction = "lm", crit = crit,
                    plotty = FALSE, method = "h")
  return(summary(result)$bestmodel)

}

# function that runs step by step algorithm for given criteria and direction
```

```
run_stepwise <- function(data, direction = "forward", crit = "aic") {
	modele.trivial <- lm(Average.playtime.forever ~ 1, data = data)
	modele.complet <- lm(Average.playtime.forever ~ ., data = data)
	# penality k according to criteria
	k <- switch(crit,
	"aic" = 2,
	"bic" = log(nrow(data)),
	"F"   = NULL)
	# run algorithm
	if (crit == "F") {
	result <- step(modele.trivial, scope = list(lower = modele.trivial, upper = 
modele.complet),
			data = data, direction = direction, test = "F")
	} else {
	result <- step(modele.trivial, scope = list(lower = modele.trivial, upper = 
modele.complet),
			data = data, direction = direction, k = k)
	}
	return(result)
}

turn_data_to_num <- function(data) {
  y <- data[["Average.playtime.forever"]]
  XX <- model.matrix(~ ., data = data %>% select(-Average.playtime.forever))[, -1]
  colnames(XX) <- make.names(colnames(XX))
  data_num <- as.data.frame(cbind(Average.playtime.forever = y, XX))
  return(data_num)
}

compare_models <- function(model_list, model_names = NULL) {
  if (is.null(model_names)) {
	model_names <- paste0("Model_", seq_along(model_list))
```

```
}

results <- lapply(seq_along(model_list), function(i) {
    model <- model_list[[i]]
    name <- model_names[i]

    # Extract basic stats
    aic_val <- AIC(model)
    bic_val <- BIC(model)
    adj_r2 <- summary(model)$adj.r.squared
    rse <- sigma(model)  # residual standard error

    # Assumptions
    sw_test <- shapiro.test(residuals(model))$p.value
    bp_test <- bptest(model)$p.value
    vif_vals <- tryCatch({
    vif(model)
    }, error = function(e) rep(NA, length(coefficients(model))))

    mean_vif <- if (is.numeric(vif_vals)) mean(vif_vals, na.rm = TRUE) else NA

    # Return a row
    data.frame(
    Model = name,
    AIC = round(aic_val, 2),
    BIC = round(bic_val, 2),
    Adj_R2 = round(adj_r2, 3),
    RSE = round(rse, 2),
    Shapiro.p = round(sw_test, 4),
    BP.p = round(bp_test, 4),
    Mean_VIF = round(mean_vif, 2),
```

```
            stringsAsFactors = FALSE
            )
  })

  do.call(rbind, results)
}

ubisoft <- games %>% filter(Publishers == "Ubisoft") %>%
  select(Average.playtime.forever, Estimated.owners,
            Peak.CCU, rating, Price,
            Recommendations, Required.age,
            Positive, Negative,
            total_reviews, positive_ratio)

## aic bic ----

ubisoft_aic <- select_model_glmulti(ubisoft, crit = "aic")
ubisoft_aic <- lm(ubisoft_aic, ubisoft)
summary(ubisoft_aic)

ubisoft_bic <- select_model_glmulti(ubisoft, crit = "bic")
ubisoft_bic <- lm(ubisoft_bic, ubisoft)
summary(ubisoft_bic)

## aic bic after turning whole data into numeric ----
# (too long, and owners is good on its own)

# ubisoft_num <- turn_data_to_num(ubisoft)
# ubisoft_aic_num <- select_model_glmulti(ubisoft_num, crit = "aic")
# ubisoft_aic_num <- lm(ubisoft_aic, ubisoft_num)
# summary(ubisoft_aic_num)
```

```
## forward selection ----

# aic
ubisoft_aic_for <- run_stepwise(ubisoft, direction = "forward", crit = "aic")
# bic
ubisoft_bic_for <- run_stepwise(ubisoft, direction = "forward", crit = "bic")
# Fischer
ubisoft_F_for <- run_stepwise(ubisoft, direction = "forward", crit = "F")

## backward selection ----

# aic
ubisoft_aic_back <- run_stepwise(ubisoft, direction = "backward", crit = "aic")
# bic
ubisoft_bic_back <- run_stepwise(ubisoft, direction = "backward", crit = "bic")
#s Fischer
ubisoft_F_back <- run_stepwise(ubisoft, direction = "backward", crit = "F")

## both directions ----

# aic
ubisoft_aic_both <- run_stepwise(ubisoft, direction = "both", crit = "aic")
# bic
ubisoft_bic_both <- run_stepwise(ubisoft, direction = "both", crit = "bic")
# Fischer
ubisoft_F_both <- run_stepwise(ubisoft, direction = "both", crit = "F")

## comparison ----
```

```
models_to_compare <- list(ubisoft_aic_for, ubisoft_bic_for, ubisoft_F_for,
                          ubisoft_aic_both, ubisoft_bic_both, ubisoft_F_both)
names_to_use <- c("AIC forward", "BIC forward", "Fischer forward",
                  "AIC both", "BIC both", "Fischer both")

compare_models(models_to_compare, names_to_use)
```

9.classification.R

```
## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
        select(Average.playtime.forever, Estimated.owners,
        Peak.CCU, rating, Price,
        Recommendations, Required.age,
        Positive, Negative,
        total_reviews, positive_ratio)

# setting class reference to 0-20k
gamesc$Estimated.owners <- relevel(gamesc$Estimated.owners, ref = "0-20k")

# log transformation
gamesc$Average.playtime.forever <- log1p(gamesc$Average.playtime.forever)
gamesc$Peak.CCU <- log1p(gamesc$Peak.CCU)
gamesc$Positive <- log1p(gamesc$Positive)
gamesc$Negative <- log1p(gamesc$Negative)
gamesc$Recommendations <- log1p(gamesc$Recommendations)
gamesc$Price <- log1p(gamesc$Price)
```

```
# standardisation
X <- c("Average.playtime.forever", "Peak.CCU", "Positive", "Negative",
        "Recommendations", "Price", "Required.age")
gamesc_scaled <- as.data.frame(scale(gamesc[, X]))
gamesc_scaled$Estimated.owners <- gamesc$Estimated.owners

# create logit model
model_logit <- multinom(Estimated.owners ~ ., data = gamesc_scaled)

# testing coefficient significance
z <- summary(model_logit)$coefficients / summary(model_logit)$standard.errors
p_values <- 2 * (1 - pnorm(abs(z)))
cat("\n--- P-values des coefficients ---\n")
print(round(p_values, 4))

# AIC of model
cat("\n--- AIC du modèle ---\n")
print(AIC(model_logit))

# pseudo R²
cat("\n--- Pseudo R² ---\n")
print(pR2(model_logit))

# VIF for multicolinearity
mod_lineaire_temp <- lm(
        as.numeric(as.factor(Estimated.owners)) ~ Peak.CCU + Positive + Negative +
Recommendations + Price + Required.age,
        data = gamesc_scaled
)
cat("\n--- VIF (multicolinéarité) ---\n")
print(vif(mod_lineaire_temp))
```

```r
# prediction quality, confusion matrix
pred <- predict(model_logit)
cat("\n--- Matrice de confusion ---\n")
print(table(Predicted = pred, Actual = gamesc_scaled$Estimated.owners))

# accuracy calculation
accuracy <- mean(pred == gamesc_scaled$Estimated.owners)
cat("\n--- Taux de bonnes prédictions ---\n")
print(round(accuracy, 4))
```

10.poly.R

```
# tests polynomial models up to degree 3 for each given variable
# variable to predict is Average.playtime.forever

## IMPORTS ----
rm(list = ls())
graphics.off()
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
temp_env <- new.env()
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env)

gamesc <- games %>%
    select(Average.playtime.forever, Estimated.owners,
    Peak.CCU, rating, Price,
    Recommendations, Required.age,
    Positive, Negative,
    total_reviews, positive_ratio)

check_polynomial_models <- function(data, response_var, predictors) {
    for (var in predictors) {
    cat("\n", strrep("-", 60), "\n")
    cat("Analyse pour la variable explicative :", var, "\n")

    # creating models with deg 1, 2 and 3
    f1 <- as.formula(paste(response_var, "~", var))
    f2 <- as.formula(paste(response_var, "~", var, "+ I(", var, "^2)"))
    f3 <- as.formula(paste(response_var, "~", var, "+ I(", var, "^2) + I(", var, "^3)"))
    m1 <- lm(f1, data = data)
```

```
m2 <- lm(f2, data = data)
m3 <- lm(f3, data = data)

# anova
cat("\n> Résultat ANOVA entre les modèles de degré 1 à 3 :\n")
print(anova(m1, m2, m3))

# R2 adjusted
r2_adj <- c(
round(summary(m1)$adj.r.squared, 4),
round(summary(m2)$adj.r.squared, 4),
round(summary(m3)$adj.r.squared, 4)
)
names(r2_adj) <- c("Degré 1", "Degré 2", "Degré 3")
cat("\n> R² ajusté :\n")
print(r2_adj)

# AIC
cat("\n> AIC des modèles :\n")
print(AIC(m1, m2, m3))

# graph to compare models performances
print(
ggplot(data, aes(x = !!sym(var), y = !!sym(response_var))) +
geom_point(alpha = 0.5) +
geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "black") +
geom_smooth(method = "lm", formula = y ~ poly(x, 2), se = FALSE, color = "red") +
geom_smooth(method = "lm", formula = y ~ poly(x, 3), se = FALSE, color = "blue") +
labs(title = paste("Ajustements polynomiaux pour", var),
        subtitle = "Noir = linéaire, rouge = degré 2, bleu = degré 3")
)
```

```
    }
}

# function to transform data with log10
apply_transformations <- function(data, variables) {
    for (var in variables) {
    # log transformation: log10(x + 1) to avoid log(0)
    data[[var]] <- log10(data[[var]] + 1)
    }
    return(data)
}

# log tranformations
variables_to_transform <- c("Average.playtime.forever","Peak.CCU",
        "Positive", "Negative", "Recommendations", "Price")
gamesc_log <- apply_transformations(gamesc, variables_to_transform)

X <- c("Peak.CCU", "Positive", "Negative", "Recommendations", "Price", "Required.age")
check_polynomial_models(gamesc, response_var = "Average.playtime.forever", predictors =
X)
```

11.report.R

```
# this script will generate all the graphs and data in our annex
# starting from our first linear model attempt
# refer to files 1, 2, 3, 5, and 6 for the rest of the annex

## IMPORTS ----
rm(list = ls()) #clean environment
graphics.off() #clean plots
setwd(dirname(rstudioapi::getActiveDocumentContext()$path)) #set working directory
temp_env <- new.env() #temporary environment to avoid uneccessary variables after import
source("0setup.R", local = temp_env)
games <- temp_env$setup()
rm(temp_env) #delete temporary environment after data has been loaded

#select variables for analysis
gamesc <- games %>%
	select(Average.playtime.forever, Estimated.owners,
	Peak.CCU, rating, Price,
	Recommendations, Required.age,
	Positive, Negative,
	total_reviews, positive_ratio)

## functions to create models and test hypotheses ----
# function to create a linear model
create_lm <- function(dataset, Y, X, categories) {
	if (length(categories) == 0) {
	formula <- as.formula(paste(Y, "~", paste(X, collapse = "+")))
	} else {
	formula <- as.formula(paste(Y, "~", paste(c(X, categories), collapse = "+")))
```

```
	}
	model <- lm(formula = formula, data = dataset)
	return(model)
}

# function to graph and test model hypotheses
check_lm_hypotheses <- function(model, data) {
	cat("Vérification des hypothèses pour le modèle :", deparse(model$call), "\n\n")

	# hypothese 1: relation de linearite entre Y et X
	# le nuage de points doit etre centre autour de 0 sans motif evident
	# sinon la relation n est pas bien modelise, non-linearite possible
	plot(model, which = 1, main = "1. Résidus vs valeurs ajustées")

	# hypothese 2: homoscedasticite des erreurs
	# on veut un ecart type constant peu importe la valeur ajustee
	# entonnoir = heteroscedasticite
	plot(model, which = 3, main = "2.1. Écarts à l'effet de levier")

	# residus absolus vs valeurs ajustees
	# si c'est croissant/decroissant, la variance n est pas constante
	data$residu_abs <- abs(residuals(model))
	data$ajuste <- fitted(model)
	ggplot(data, aes(x = ajuste, y = residu_abs)) +
	geom_point(alpha = 0.4) +
	geom_smooth(method = "loess", col = "red") +
	labs(title = "2.2. Hétéroscédasticité : résidus absolus vs ajustés",
	x = "Valeurs ajustées", y = "Résidus absolus") +
	theme_minimal() -> p1
	print(p1)
```

```
# hypothese 3: independance des erreurs
# test de Durbin-Watson : attend une valeur proche de 2
# si proche de 0 : autocorrelation positive. si > 2.5 : autocorrelation negative
cat("\nTest de Durbin-Watson (attendu ≈ 2) :\n")
print(dwtest(model))
dwtest(model, alternative = c("two.sided"))
# ACF : si les barres depassent, il y a autocorrelation des erreurs
acf(residuals(model), main = "3. ACF des résidus")

# hypothese 4: normalite des erreurs
# ca doit suivre la ligne, sinon residus anormaux
plot(model, which = 2, main = "4. QQ-plot des résidus")
residus <- model$residuals
#boxplot des residus
boxplot(residus, main = "Boxplot des résidus")
#histogramme "naif"
hist(residus[residus < quantile(residus, 1)], breaks = 50,
main = "Histogramme des résidus",
xlab = "Résidus", col = "green", border = "green")
#histogramme plus lisible en retirant le top 1%
hist(residus[residus < quantile(residus, 0.99)], breaks = 50,
main = "Histogramme des résidus (sans top 1%)",
xlab = "Résidus (censurés à 99%)", col = "blue", border = "blue")

# hypothese 5 multicolinearite
# VIF (Variance Inflation Factor) : mesure le lien entre les variables explicatives
# VIF > 5 = multicolinearite moderee VIF > 10 = multicolinearite severe
cat("\nVIF (Variance Inflation Factor) :\n")
vif_vals <- vif(model)
print(vif_vals)
cat("\nVariables avec VIF > 5 :\n")
```

```
	print(names(vif_vals[vif_vals > 5]))

	# observations influentes: distances de Cook
	# indique si certaines observations influencent beaucoup le modele
	# attention aux lignes au dessus de la ligne rouge
	cooks <- cooks.distance(model)
	seuil <- 4 / nrow(data)
	# cat("\nObservations influentes (Cook > 4/n) :\n")
	# influents <- which(cooks > seuil)
	# print(influents)

	plot(cooks, type = "h",
	main = "6. Distance de Cook avec seuil 4/n",
	ylab = "Distance de Cook", xlab = "Index de l'observation")
	abline(h = seuil, col = "red", lty = 2, lwd = 2)
	legend("topright", legend = paste0("Seuil = 4/n ≈ ", round(seuil, 5)),
	col = "red", lty = 2, lwd = 2)
}

# function to transform data with log10
apply_transformations <- function(data, variables) {
	for (var in variables) {
	# log transformation: log10(x + 1) to avoid log(0)
	data[[var]] <- log10(data[[var]] + 1)

	}
	return(data)
}

# function to detect high influence point with Cook's distance
detect_cook <- function(model, threshold = 4 / nrow(model$model)) {
```

```
    cooks <- cooks.distance(model)
    which(cooks > threshold)
}

# function to detect abnormally large residuals
detect_large_residuals <- function(model, threshold = 3) {
    rstudent_res <- rstudent(model)
    which(abs(rstudent_res) > threshold)
}

# function to detect outliers
detect_outliers_data <- function(dataset, threshold = 3) {
    numeric_data <- dataset[sapply(dataset, is.numeric)]
    z_scores <- scale(numeric_data)
    which(apply(abs(z_scores) > threshold, 1, any))
}

# remove outliers, high influence observations and extreme residuals
clean_model <- function(model, dataset) {
    idx_cook <- detect_cook(model)
    idx_resid <- detect_large_residuals(model)
    idx_outliers <- detect_outliers_data(dataset)
    idx_to_remove <- unique(c(idx_cook, idx_resid, idx_outliers))
    cleaned_data <- dataset[-idx_to_remove, ]
    return(list(data = cleaned_data, removed = idx_to_remove))
}

## 1 first simple model ----
Y <- "Average.playtime.forever"
X <- c("Peak.CCU", "Positive", "Negative", "Recommendations", "Price", "Required.age")
categories <- c("Estimated.owners")
```

```
model <- create_lm(gamesc, Y, X, categories)
summary(model)
check_lm_hypotheses(model, gamesc)

## 2 model with log transformation to be closer to linearity ----
Y <- "Average.playtime.forever"
X <- c("Peak.CCU", "Positive", "Negative", "Recommendations",
       "Price", "Required.age")
categories <- c("Estimated.owners")

# log tranformations
variables_to_transform <- c("Average.playtime.forever","Peak.CCU",
                            "Positive", "Negative", "Recommendations", "Price")
gamesc_log <- apply_transformations(gamesc, variables_to_transform)

model_log <- create_lm(gamesc_log, Y, X, categories)
summary(model_log)
check_lm_hypotheses(model_log, gamesc_log)

## 3 model with log but without outliers, high influence point, and extreme errors ----
cleaning <- clean_model(model_log, gamesc_log)
gamesc_log_clean <- cleaning$data

model_log_clean <- create_lm(gamesc_log_clean, Y, X, categories)
summary(model_log_clean)
# cat("Nombre de points retirés :", length(cleaning$removed), "\n")

## 4 model with only ubisoft games, no transformation ----

ubisoft <- games %>% filter(Publishers == "Ubisoft") %>%
```

```
        select(Average.playtime.forever, Estimated.owners,
        Peak.CCU, rating, Price,
        Recommendations, Required.age,
        Positive, Negative,
        total_reviews, positive_ratio)
Y <- "Average.playtime.forever"
X <- c("Peak.CCU", "Recommendations", "Price", "Required.age")
categories <- c()
categories <- c("Estimated.owners", "rating")

model_ubisoft <- create_lm(ubisoft, Y, X, categories)
summary(model_ubisoft)
check_lm_hypotheses(model_ubisoft, ubisoft)

## functions to test selection algorithms ----

# function that finds the best linear model for given criteria
select_model_glmulti <- function(data, crit = "aic", level = 1) {
        formula <- as.formula("Average.playtime.forever ~ .")
        result <- glmulti(formula, data = data, level = level,
                fitfunction = "lm", crit = crit,
                plotty = FALSE, method = "h")
        return(summary(result)$bestmodel)

}

# function that runs step by step algorithm for given criteria and direction
run_stepwise <- function(data, direction = "forward", crit = "aic") {
        modele.trivial <- lm(Average.playtime.forever ~ 1, data = data)
        modele.complet <- lm(Average.playtime.forever ~ ., data = data)
        # penality k according to criteria
```

```
	k <- switch(crit,
	"aic" = 2,
	"bic" = log(nrow(data)),
	"F"   = NULL)
	# run algorithm
	if (crit == "F") {
	result <- step(modele.trivial, scope = list(lower = modele.trivial, upper = 
modele.complet),
	          data = data, direction = direction, test = "F")
	} else {
	result <- step(modele.trivial, scope = list(lower = modele.trivial, upper = 
modele.complet),
	          data = data, direction = direction, k = k)
	}
	return(result)
}

# turn qualitative variables to numerical data, using indicator function
turn_data_to_num <- function(data) {
	y <- data[["Average.playtime.forever"]]
	XX <- model.matrix(~ ., data = data %>% select(-Average.playtime.forever))[, -1]
	colnames(XX) <- make.names(colnames(XX))
	data_num <- as.data.frame(cbind(Average.playtime.forever = y, XX))
	return(data_num)
}

# compare models by displaying key values in a table
compare_models <- function(model_list, model_names = NULL) {
	if (is.null(model_names)) {
	model_names <- paste0("Model_", seq_along(model_list))
	}
```

```
results <- lapply(seq_along(model_list), function(i) {
model <- model_list[[i]]
name <- model_names[i]

# Extract basic stats
aic_val <- AIC(model)
bic_val <- BIC(model)
adj_r2 <- summary(model)$adj.r.squared
rse <- sigma(model)  # residual standard error

# Assumptions
sw_test <- shapiro.test(residuals(model))$p.value
bp_test <- bptest(model)$p.value
vif_vals <- tryCatch({
vif(model)
}, error = function(e) rep(NA, length(coefficients(model))))

mean_vif <- if (is.numeric(vif_vals)) mean(vif_vals, na.rm = TRUE) else NA

# Return a row
data.frame(
Model = name,
AIC = round(aic_val, 2),
BIC = round(bic_val, 2),
Adj_R2 = round(adj_r2, 3),
RSE = round(rse, 2),
Shapiro.p = round(sw_test, 4),
BP.p = round(bp_test, 4),
Mean_VIF = round(mean_vif, 2),
stringsAsFactors = FALSE
)
```

```r
        })

        do.call(rbind, results)
}


## testing algorithms selection on ubisoft games ----

## glmulti

ubisoft_aic <- select_model_glmulti(ubisoft, crit = "aic")
ubisoft_aic <- lm(ubisoft_aic, ubisoft)

ubisoft_bic <- select_model_glmulti(ubisoft, crit = "bic")
ubisoft_bic <- lm(ubisoft_bic, ubisoft)

## forward selection
ubisoft_aic_for <- run_stepwise(ubisoft, direction = "forward", crit = "aic")
ubisoft_bic_for <- run_stepwise(ubisoft, direction = "forward", crit = "bic")
ubisoft_F_for <- run_stepwise(ubisoft, direction = "forward", crit = "F")

## backward selection
ubisoft_aic_back <- run_stepwise(ubisoft, direction = "backward", crit = "aic")
ubisoft_bic_back <- run_stepwise(ubisoft, direction = "backward", crit = "bic")
ubisoft_F_back <- run_stepwise(ubisoft, direction = "backward", crit = "F")

## both directions
ubisoft_aic_both <- run_stepwise(ubisoft, direction = "both", crit = "aic")
ubisoft_bic_both <- run_stepwise(ubisoft, direction = "both", crit = "bic")
ubisoft_F_both <- run_stepwise(ubisoft, direction = "both", crit = "F")
```

```
## comparison
models_to_compare <- list(ubisoft_aic, ubisoft_bic,
                    ubisoft_aic_for, ubisoft_bic_for, ubisoft_F_for,
                    ubisoft_aic_both, ubisoft_bic_both, ubisoft_F_both)
names_to_use <- c("AIC", "BIC",
                    "AIC forward", "BIC forward", "Fischer forward",
                    "AIC both", "BIC both", "Fischer both")
# backward direction returns trivial model, so we are not comparing them
compare_models(models_to_compare, names_to_use)

## show variables used in each model
for (i in seq_along(models_to_compare)) {
        cat("\n---", names_to_use[i], "---\n")
        print(attr(terms(models_to_compare[[i]]), "term.labels"))
}

## classification to predict estimated owners ----

# setting class reference to 0-20k
gamesc$Estimated.owners <- relevel(gamesc$Estimated.owners, ref = "0-20k")

# log transformation
gamesc$Average.playtime.forever <- log1p(gamesc$Average.playtime.forever)
gamesc$Peak.CCU <- log1p(gamesc$Peak.CCU)
gamesc$Positive <- log1p(gamesc$Positive)
gamesc$Negative <- log1p(gamesc$Negative)
gamesc$Recommendations <- log1p(gamesc$Recommendations)
gamesc$Price <- log1p(gamesc$Price)

# standardisation
X <- c("Average.playtime.forever", "Peak.CCU", "Positive", "Negative",
```

```
        "Recommendations", "Price", "Required.age")
gamesc_scaled <- as.data.frame(scale(gamesc[, X]))
gamesc_scaled$Estimated.owners <- gamesc$Estimated.owners

# create logit model
model_logit <- multinom(Estimated.owners ~ ., data = gamesc_scaled)

# testing coefficient significance
z <- summary(model_logit)$coefficients / summary(model_logit)$standard.errors
p_values <- 2 * (1 - pnorm(abs(z)))
cat("\n--- P-values des coefficients ---\n")
print(round(p_values, 4))

# AIC of model
cat("\n--- AIC du modèle ---\n")
print(AIC(model_logit))

# pseudo R²
cat("\n--- Pseudo R² ---\n")
print(pR2(model_logit))

# VIF for multicolinearity
mod_lineaire_temp <- lm(
        as.numeric(as.factor(Estimated.owners)) ~ Peak.CCU + Positive + Negative +
Recommendations + Price + Required.age,
        data = gamesc_scaled
)
cat("\n--- VIF (multicolinéarité) ---\n")
print(vif(mod_lineaire_temp))

# prediction quality, confusion matrix
```

```
pred <- predict(model_logit)
cat("\n--- Matrice de confusion ---\n")
print(table(Predicted = pred, Actual = gamesc_scaled$Estimated.owners))

# accuracy calculation
accuracy <- mean(pred == gamesc_scaled$Estimated.owners)
cat("\n--- Taux de bonnes prédictions ---\n")
print(round(accuracy, 4))

# trivial classification, always predict most common class
majority_class <- names(which.max(table(gamesc_scaled$Estimated.owners)))
trivial_pred <- rep(majority_class, nrow(gamesc_scaled))

# accuracy of trivial classification
trivial_accuracy <- mean(trivial_pred == gamesc_scaled$Estimated.owners)
cat("\n--- Taux de bonnes prédictions (modèle trivial) ---\n")
print(round(trivial_accuracy, 4))

## polynomial models tests ----
check_polynomial_models <- function(data, response_var, predictors) {
	for (var in predictors) {
	cat("\n", strrep("-", 60), "\n")
	cat("Analyse pour la variable explicative :", var, "\n")

	# creating models with deg 1, 2 and 3
	f1 <- as.formula(paste(response_var, "~", var))
	f2 <- as.formula(paste(response_var, "~", var, "+ I(", var, "^2)"))
	f3 <- as.formula(paste(response_var, "~", var, "+ I(", var, "^2) + I(", var, "^3)"))
	m1 <- lm(f1, data = data)
	m2 <- lm(f2, data = data)
	m3 <- lm(f3, data = data)
```

```
        # anova
        cat("\n> Résultat ANOVA entre les modèles de degré 1 à 3 :\n")
        print(anova(m1, m2, m3))

        # R2 adjusted
        r2_adj <- c(
        round(summary(m1)$adj.r.squared, 4),
        round(summary(m2)$adj.r.squared, 4),
        round(summary(m3)$adj.r.squared, 4)
        )
        names(r2_adj) <- c("Degré 1", "Degré 2", "Degré 3")
        cat("\n> R² ajusté :\n")
        print(r2_adj)

        # AIC
        cat("\n> AIC des modèles :\n")
        print(AIC(m1, m2, m3))

        # graph to compare models performances
        print(
        ggplot(data, aes(x = !!sym(var), y = !!sym(response_var))) +
        geom_point(alpha = 0.5) +
        geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "black") +
        geom_smooth(method = "lm", formula = y ~ poly(x, 2), se = FALSE, color = "red") +
        geom_smooth(method = "lm", formula = y ~ poly(x, 3), se = FALSE, color = "blue") +
        labs(title = paste("Ajustements polynomiaux pour", var),
                subtitle = "Noir = linéaire, rouge = degré 2, bleu = degré 3")
        )
        }
}
```

```
# function to transform data with log10
apply_transformations <- function(data, variables) {
    for (var in variables) {
    # log transformation: log10(x + 1) to avoid log(0)
    data[[var]] <- log10(data[[var]] + 1)
    }
    return(data)
}

# log tranformations
variables_to_transform <- c("Average.playtime.forever","Peak.CCU",
        "Positive", "Negative", "Recommendations", "Price")
gamesc_log <- apply_transformations(gamesc, variables_to_transform)

X <- c("Peak.CCU", "Positive", "Negative", "Recommendations", "Price", "Required.age")
check_polynomial_models(gamesc, response_var = "Average.playtime.forever", predictors =
X)
```