Reproducible pipeline

This appendix documents the exact code and steps used to preprocess and analyze the review corpora for **Design District DAA** and **Krasny Treugolnik** (**KTR**). It is designed to be readable in print and runnable from the project folder Thesis-DAA-KTR/.

A. Folder layout

```
Thesis-DAA-KTR/
├─ README.md
├─ LICENSE
├─ .gitignore
├─ environment.yml # conda env (Python)
├─ requirements-lock.txt # pinned pip versions
├─ R/
│ ├─ analysis_R.R # base summaries/plots (all data)
│ ├─ r_analysis.R # extended stats/theme codes (all data)
│ ├─ r_analysis_per_case.R # per-case outputs (DAA / KTR)
│ └─ fix_reviews_per_year.R # clamp temporal axes; rebuild per-year plots
├─ src/
│ └─ process_reviews.py # Python preprocessing & date normalization
├─ data/
│ └─ raw/
│ └─ reviews_master.xlsx # or reviews_master.csv (master file)
├─ out/
│ ├─ reviews_master_clean.feather / .parquet
│ ├─ counts_by_case_platform.csv, ...
│ ├─ DAA/ reviews_DAA.feather, figures & tables
│ └─ KTR/ reviews_KTR.feather, figures & tables
├─ run_all.sh
└─ Makefile
```

B. Reproducibility & versions

Python 3.12 with: pandas, pyarrow, matplotlib, openpyxl, python-dateutil, dateparser, chardet.

R (≥4.3) with: tidyverse, arrow, janitor, lubridate, tidytext, ggthemes, ggwordcloud (per-case WC), widyr (optional), showtext (optional), sf/terra/spatstat for the GIS add-on.

Needed to be installed once; better to avoid install.packages(...) inside analysis scripts (kept clean for execution). Use environment.yml and requirements-lock.txt for exact versions.

C. Data schema

Required columns in the master table:

case ∈ {DAA, KTR}

source_platform ∈ {GoogleMaps, YandexMaps}

rating_1_5 ∈ {1,2,3,4,5} (can be NA)

text (raw, may be empty)

stance ∈ {positive, negative, neutral, mixed} (fixed; do not edit)

theme_codes (semicolon-separated snake_case; may be empty)

pulled_by (string)

post_date (as seen on the platform; may be relative)

has_text ∈ {yes, no} (if no → stance must remain neutral by design)

D. Date normalization policy

Anchor date: 2025-09-16 (end of data collection). Relative times like “N years ago” are converted against this anchor.

Absolute dates are parsed to YYYY-MM-DD.

Year-only strings (e.g., 2021) normalize to YYYY-07-01 (neutral mid-year).

Conservative window: years outside 1995-2025 are treated as NA in post_date_norm to prevent artefacts (e.g., 1900).

E. How to run

```
# 1) Preprocess (Python) → clean layer + starter outputs
python src/process_reviews.py --input "data/raw/reviews_master.xlsx" --outdir out

# 2) R-level summaries/figures (all data)
Rscript R/analysis_R.R
Rscript R/r_analysis.R

# 3) Per-case outputs (figures, wordclouds, tests)
CASE=DAA Rscript R/r_analysis_per_case.R
CASE=KTR Rscript R/r_analysis_per_case.R

# 4) Clamp year axis and rebuild per-year plots per case
CASE=DAA Rscript R/fix_reviews_per_year.R
CASE=KTR Rscript R/fix_reviews_per_year.R
```

F. Python. Preprocessing script

(src/process_reviews.py)

Validates schema, reports missing/odd values, normalizes dates, saves clean Feather/Parquet, produces starter CSVs and a few basic PNGs.

Gemini

```
#!/usr/bin/env python3

critical = [c for c in ["case","source_platform","stance"] if c in schema.get("missing_required", [])]
if critical:
print("ERROR: missing critical columns:", ", ".join(critical)); return
def main():
    critical = [c for c in ["case","source_platform","stance"] if c in schema.get("missing_required", [])]
    if critical:
        print("ERROR: missing critical columns:", ", ".join(critical)); return

for col in REQUIRED_COLS:
if col not in df.columns: df[col] = pd.NA
    for col in REQUIRED_COLS:
        if col not in df.columns: df[col] = pd.NA

# duplicates report key
dup_cols = ["case","source_platform","text","rating_1_5"]
dup_key = df[dup_cols].astype(str).agg("||".join, axis=1)
dup_counts = dup_key.value_counts()
possible_dups = dup_counts[dup_counts > 1]
    # duplicates report key
    dup_cols = ["case","source_platform","text","rating_1_5"]
    dup_key = df[dup_cols].astype(str).agg("||".join, axis=1)
    dup_counts = dup_key.value_counts()
    possible_dups = dup_counts[dup_counts > 1]

# date normalization
df["post_date_norm"] = df["post_date"].apply(parse_post_date)
# clamp exotic years
MIN_YEAR, MAX_YEAR = 1995, 2025
yr = df["post_date_norm"].dt.year
df.loc[yr.lt(MIN_YEAR) | yr.gt(MAX_YEAR), "post_date_norm"] = pd.NaT
    # date normalization
    df["post_date_norm"] = df["post_date"].apply(parse_post_date)
    # clamp exotic years
    MIN_YEAR, MAX_YEAR = 1995, 2025
    yr = df["post_date_norm"].dt.year
    df.loc[yr.lt(MIN_YEAR) | yr.gt(MAX_YEAR), "post_date_norm"] = pd.NaT

# save clean
feather_path = os.path.join(args.outdir, "reviews_master_clean.feather")
parquet_path = os.path.join(args.outdir, "reviews_master_clean.parquet")
df.to_feather(feather_path); df.to_parquet(parquet_path, index=False)
    # save clean
    feather_path = os.path.join(args.outdir, "reviews_master_clean.feather")
    parquet_path = os.path.join(args.outdir, "reviews_master_clean.parquet")
    df.to_feather(feather_path); df.to_parquet(parquet_path, index=False)

# simple summaries
total = len(df)
ctab = pd.crosstab(df["case"], df["source_platform"], dropna=False)
df["rating_1_5"] = pd.to_numeric(df["rating_1_5"], errors="coerce")
    # simple summaries
    total = len(df)
```

```
    ctab = pd.crosstab(df["case"], df["source_platform"], dropna=False)
    df["rating_1_5"] = pd.to_numeric(df["rating_1_5"], errors="coerce")

# exports
ctab.to_csv(os.path.join(args.outdir, "counts_by_case_platform.csv"))
stance_by = pd.crosstab([df["case"], df["source_platform"]], df["stance"], dropna=False)
stance_by.to_csv(os.path.join(args.outdir, "stance_by_case_platform.csv"))
    # exports
    ctab.to_csv(os.path.join(args.outdir, "counts_by_case_platform.csv"))
    stance_by = pd.crosstab([df["case"], df["source_platform"]], df["stance"], dropna=False)
    stance_by.to_csv(os.path.join(args.outdir, "stance_by_case_platform.csv"))

# theme codes list
codes_path = os.path.join(args.outdir, "theme_codes_raw.txt")
all_codes = []
for s in df["theme_codes"].fillna(""):
s = str(s).strip()
if s: all_codes.extend([t.strip() for t in s.split(";") if t.strip()])
with open(codes_path, "w", encoding="utf-8") as f:
f.write(";".join(sorted(set(all_codes))))
    # theme codes list
    codes_path = os.path.join(args.outdir, "theme_codes_raw.txt")
    all_codes = []
    for s in df["theme_codes"].fillna(""):
        s = str(s).strip()
        if s: all_codes.extend([t.strip() for t in s.split(";") if t.strip()])
    with open(codes_path, "w", encoding="utf-8") as f:
        f.write(";".join(sorted(set(all_codes))))

# basic figs (matplotlib; default colors)
import matplotlib
if not df["rating_1_5"].dropna().empty:
plt.figure(); df["rating_1_5"].dropna().astype(int).plot(kind="hist", bins=[1,2,3,4,5,6], rwidth=0.9)
plt.title("Histogram: rating_1_5"); plt.xlabel("Rating (1–5)"); plt.ylabel("Count")
plt.savefig(os.path.join(args.outdir, "hist_rating_1_5.png"), dpi=200, bbox_inches="tight"); plt.close()
    # basic figs (matplotlib; default colors)
    import matplotlib
    if not df["rating_1_5"].dropna().empty:
        plt.figure(); df["rating_1_5"].dropna().astype(int).plot(kind="hist", bins=[1,2,3,4,5,6], rwidth=0.9)
        plt.title("Histogram: rating_1_5"); plt.xlabel("Rating (1–5)"); plt.ylabel("Count")
        plt.savefig(os.path.join(args.outdir, "hist_rating_1_5.png"), dpi=200, bbox_inches="tight"); plt.close()

# console report
print("\n=== SUMMARY ===")
print(f"Total reviews: {total}")
print("\ncase × source_platform:\n", ctab)
na_share = float(df["post_date_norm"].isna().mean())
print(f"\nShare with NA post_date_norm: {na_share:.3%}")
if len(possible_dups):
print("\nPossible duplicate groups:", len(possible_dups))
print(possible_dups.head(10))
if schema.get("missing_required"):
print("\nMissing (non-critical) columns:", schema["missing_required"])
if schema.get("value_issues"):
print("\nValue issues:", schema["value_issues"])
    # console report
    print("\n=== SUMMARY ===")
    print(f"Total reviews: {total}")
    print("\ncase × source_platform:\n", ctab)
    na_share = float(df["post_date_norm"].isna().mean())
    print(f"\nShare with NA post_date_norm: {na_share:.3%}")
    if len(possible_dups):
        print("\nPossible duplicate groups:", len(possible_dups))
        print(possible_dups.head(10))
    if schema.get("missing_required"):
        print("\nMissing (non-critical) columns:", schema["missing_required"])
    if schema.get("value_issues"):
        print("\nValue issues:", schema["value_issues"])

if __name__ == "__main__":
main()
    main()
```

```
  File "/tmp/ipython-input-3227805830.py", line 5
    print("ERROR: missing critical columns:", ", ".join(critical)); return
    ^
IndentationError: expected an indented block after 'if' statement on line 4
```

Далее: Объяснить ошибку

G. R — base summaries/plots

(R/analysis_R.R)

Clean, non-interactive script. No install.packages(...) here; see README for setup. Writes CSV/PNG into ./out.

```
# analysis_R.R — base summaries & plots for reviews_master_clean.feather
})

out_dir <- "./out"; if (!dir.exists(out_dir)) dir.create(out_dir, recursive = TRUE)
df <- read_feather(file.path(out_dir, "reviews_master_clean.feather")) %>% clean_names()

df <- df %>% mutate(
post_date_norm = as_date(post_date_norm),
rating_1_5_num = suppressWarnings(as.integer(rating_1_5)),
has_text = factor(has_text),
stance = factor(stance),
case = factor(case),
source_platform = factor(source_platform)
)

# 1) counts
write_csv(df %>% count(case, source_platform, name = "n") %>% arrange(case, source_platform),
file.path(out_dir, "R_counts_by_case_platform.csv"))
write_csv(df %>% count(stance, name = "n") %>% arrange(desc(n)),
file.path(out_dir, "R_stance_overall.csv"))

# stance by platform/case with shares
stab <- df %>% count(source_platform, stance, name = "n") %>% group_by(source_platform) %>% mutate(share = n/sum(n)) %>%
write_csv(stab, file.path(out_dir, "R_stance_by_platform.csv"))
scab <- df %>% count(case, stance, name = "n") %>% group_by(case) %>% mutate(share = n/sum(n)) %>% ungroup()
write_csv(scab, file.path(out_dir, "R_stance_by_case.csv"))

# time tables
df_time <- df %>% filter(!is.na(post_date_norm)) %>% mutate(year = year(post_date_norm), ym = floor_date(post_date_norm,
write_csv(df_time %>% count(year, name = "n") %>% arrange(year), file.path(out_dir, "R_counts_by_year.csv"))
write_csv(df_time %>% count(ym, name = "n") %>% arrange(ym), file.path(out_dir, "R_counts_by_year_month.csv"))

# plots
theme_pub <- theme_minimal(base_size = 12)

p_rating <- df %>% filter(!is.na(rating_1_5_num)) %>% ggplot(aes(x = rating_1_5_num)) +
geom_histogram(binwidth = 1, boundary = 0.5) + scale_x_continuous(breaks = 1:5) +
labs(title = "Rating 1–5 (histogram)", x = "Rating", y = "Count") + theme_pub

ggsave(file.path(out_dir, "R_hist_rating_1_5.png"), p_rating, width = 7, height = 4.5, dpi = 200)

p_sp <- stab %>% ggplot(aes(x = source_platform, y = n, fill = stance)) + geom_col() +
labs(title = "Stance by Source Platform", x = "Platform", y = "Count", fill = "Stance") + theme_pub

ggsave(file.path(out_dir, "R_bar_stance_by_platform.png"), p_sp, width = 8, height = 5, dpi = 200)

p_sc <- scab %>% ggplot(aes(x = case, y = n, fill = stance)) + geom_col() +
labs(title = "Stance by Case", x = "Case", y = "Count", fill = "Stance") + theme_pub

ggsave(file.path(out_dir, "R_bar_stance_by_case.png"), p_sc, width = 7, height = 4.5, dpi = 200)

p_ts <- df_time %>% count(ym, name = "n") %>% ggplot(aes(x = ym, y = n)) + geom_line() + geom_point(size = 1.2) +
labs(title = "Reviews per Month", x = "Year–Month", y = "Count") + theme_pub

ggsave(file.path(out_dir, "R_ts_reviews_per_month.png"), p_ts, width = 8, height = 4.5, dpi = 200)

p_ts_pl <- df_time %>% count(source_platform, ym, name = "n") %>% ggplot(aes(x = ym, y = n, color = source_platform)) +
geom_line() + geom_point(size = 1.2) + labs(title = "Reviews per Month by Platform", x = "Year–Month", y = "Count", colo

ggsave(file.path(out_dir, "R_ts_reviews_per_month_by_platform.png"), p_ts_pl, width = 9, height = 5, dpi = 200)

message("Base R analysis saved to ./out")
```

H. R — extended stats & themes

(R/r_analysis.R)

Adds boxplots, per-year counts, top themes, heatmap theme × case, χ^2 and t/Wilcoxon tests.

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
suppressPackageStartupMessages({
ggsave(file.path(out_dir, "stance_by_case.png"), p_sc, width = 7, height = 4.5, dpi = 200)

p_sp <- tab_stance_platform |> ggplot(aes(x = source_platform, y = n, fill = stance)) + geom_col() + labs(title = "Stance
ggsave(file.path(out_dir, "stance_by_platform.png"), p_sp, width = 8, height = 5, dpi = 200)
```

```
p_box <- df |> filter(!is.na(rating_1_5_num)) |> ggplot(aes(x = stance, y = rating_1_5_num)) + geom_boxplot(outlier.alpha

ggsave(file.path(out_dir, "ratings_boxplot.png"), p_box, width = 8, height = 5, dpi = 200)

# per-year
df_time <- df |> filter(!is.na(post_date_norm)) |> mutate(year = year(post_date_norm))
write_csv(df_time |> count(year, name = "n") |> arrange(year), file.path(out_dir, "counts_by_year_R.csv"))

p_year <- df_time |> count(year, name = "n") |> ggplot(aes(x = year, y = n)) + geom_line() + geom_point(size = 1.4) + sca

ggsave(file.path(out_dir, "reviews_by_year.png"), p_year, width = 7.5, height = 4.5, dpi = 200)

# theme_codes long + top15 + heatmap
theme_long <- df |> select(case, source_platform, theme_codes) |> mutate(theme_codes = if_else(is.na(theme_codes) | theme

write_csv(theme_long, file.path(out_dir, "theme_codes_long.csv"))

top_themes <- theme_long |> count(theme_codes, name = "n") |> arrange(desc(n)) |> slice_head(n = 15)
write_csv(top_themes, file.path(out_dir, "top15_themes.csv"))

p_top <- top_themes |> mutate(theme_codes = fct_reorder(theme_codes, n)) |> ggplot(aes(x = theme_codes, y = n)) + geom_co

ggsave(file.path(out_dir, "top_themes_barplot.png"), p_top, width = 7, height = 5, dpi = 200)

heat_dat <- theme_long |> count(case, theme_codes, name = "n") |> group_by(case) |> mutate(share = n / sum(n)) |> ungroup

p_heat <- heat_dat |> ggplot(aes(x = case, y = fct_reorder(theme_codes, n), fill = share)) + geom_tile() + scale_fill_con

ggsave(file.path(out_dir, "themes_heatmap.png"), p_heat, width = 7.5, height = 8.5, dpi = 200)

# stats
xtab_sp <- df |> count(source_platform, stance) |> pivot_wider(names_from = stance, values_from = n, values_fill = 0) |>
chisq_result <- suppressWarnings(chisq.test(xtab_sp)); capture.output(chisq_result, file = file.path(out_dir, "chisq_star

rate_case <- df |> filter(!is.na(rating_1_5_num)) |> select(case, rating_1_5_num)

tt <- t.test(rating_1_5_num ~ case, data = rate_case)
wt <- wilcox.test(rating_1_5_num ~ case, data = rate_case, exact = FALSE)

capture.output(tt, file = file.path(out_dir, "ttest_rating_by_case.txt"))
capture.output(wt, file = file.path(out_dir, "wilcoxon_rating_by_case.txt"))

xtab_ht <- df |> count(has_text, stance) |> pivot_wider(names_from = stance, values_from = n, values_fill = 0) |> column_
chisq_ht <- suppressWarnings(chisq.test(xtab_ht)); capture.output(chisq_ht, file = file.path(out_dir, "chisq_stance_by_ha

message("Extended R analysis saved to ./out")
```

I. R — per-case outputs & wordclouds

(R/r_analysis_per_case.R)

Produces separate tables/plots for DAA or KTR. Call with CASE=DAA or CASE=KTR.

```
suppressPackageStartupMessages({
})

case_target <- Sys.getenv("CASE", "DAA")
in_dir <- file.path("out", case_target)
out_dir <- in_dir
if (!dir.exists(out_dir)) dir.create(out_dir, recursive = TRUE)

in_file <- file.path(in_dir, paste0("reviews_", case_target, ".feather"))
df <- read_feather(in_file) %>% clean_names() %>% mutate(
post_date_norm = as_date(post_date_norm),
rating_1_5_num = suppressWarnings(as.integer(rating_1_5)),
has_text = factor(has_text), stance = factor(stance), source_platform = factor(source_platform)
)

# tables
stab <- df %>% count(source_platform, stance, name="n") %>% group_by(source_platform) %>% mutate(share = n/sum(n)) %>% un
write_csv(stab, file.path(out_dir, paste0("R_", case_target, "_stance_by_platform.csv")))

htab <- df %>% count(source_platform, has_text, name="n") %>% group_by(source_platform) %>% mutate(share = n/sum(n)) %>%
write_csv(htab, file.path(out_dir, paste0("R_", case_target, "_hastext_by_platform.csv")))

mrt <- df %>% group_by(source_platform) %>% summarise(n=n(), mean_rating = mean(rating_1_5_num, na.rm=TRUE), sd_rating =
write_csv(mrt, file.path(out_dir, paste0("R_", case_target, "_mean_rating_by_platform.csv")))

# plots
p1 <- stab %>% ggplot(aes(x=source_platform, y=n, fill=stance)) + geom_col() + labs(title=paste("Stance by Platform —", c
```

```
ggsave(file.path(out_dir, paste0("R_", case_target, "_stance_by_platform.png")), p1, w=8, h=5, dpi=200)

p2 <- df %>% filter(!is.na(rating_1_5_num)) %>% ggplot(aes(x=stance, y=rating_1_5_num, fill=stance)) + geom_boxplot(outl

ggsave(file.path(out_dir, paste0("R_", case_target, "_box_rating_by_stance.png")), p2, w=7, h=4.5, dpi=200)

min_year <- 1995; max_year <- 2025

df_time <- df %>% filter(!is.na(post_date_norm)) %>% mutate(year = year(post_date_norm)) %>% filter(between(year, min_yea

tab_year <- df_time %>% count(year, name="n") %>% arrange(year)
write_csv(tab_year, file.path(out_dir, paste0("R_", case_target, "_counts_by_year.csv")))

p3 <- tab_year %>% ggplot(aes(x=year, y=n)) + geom_line() + geom_point(size=1.2) + scale_x_continuous(limits = c(min_yea

ggsave(file.path(out_dir, paste0("R_", case_target, "_ts_reviews_per_year.png")), p3, w=7.5, h=4.5, dpi=200)

# real wordclouds (has_text == yes)
df_txt <- df %>% filter(has_text=="yes", !is.na(text), text!="")
if (nrow(df_txt) > 0) {
stop_ru <- c("и","в","во","не","что","он","на","я","с","со","как","а","то","все","она","так","его","но","да","ты","к","у
stop_en <- tidytext::stop_words$word
stops <- unique(c(stop_ru, stop_en))
tokens <- df_txt %>% select(text, stance, source_platform) %>% unnest_tokens(word, text, token = "words") %>% mutate(word
wc <- tokens %>% count(word, sort = TRUE) %>% slice_max(n, n = 250)
p_wc <- ggplot(wc, aes(label = word, size = n)) + ggwordcloud::geom_text_wordcloud_area() + scale_size_area(max_size = 10
ggsave(file.path(out_dir, paste0("R_", case_target, "_wordcloud.png")), p_wc, width=8, height=6, dpi=200)
}

# chi-square (stance × platform) and t-test (rating by platform)
xtab <- df %>% count(source_platform, stance) %>% pivot_wider(names_from=stance, values_from=n, values_fill=0) %>% column
if (nrow(xtab) >= 2) capture.output(suppressWarnings(chisq.test(xtab)), file=file.path(out_dir, paste0("R_", case_target,

rt <- df %>% filter(!is.na(rating_1_5_num)) %>% select(source_platform, rating_1_5_num)
if (dplyr::n_distinct(rt$source_platform) >= 2) capture.output(t.test(rating_1_5_num ~ source_platform, data=rt), file=f

message("Per-case outputs written to ", out_dir)
```

J. R — clamp per-year plots

(R/fix_reviews_per_year.R)

Convenience script to regenerate per-year plots with a fixed window; call with CASE=DAA or CASE=KTR.

```
suppressPackageStartupMessages({ library(tidyverse); library(arrow); library(lubridate) })

case_target <- Sys.getenv("CASE", "DAA")
in_dir <- file.path("out", case_target)
in_f <- file.path(in_dir, paste0("reviews_", case_target, ".feather"))
out_png <- file.path(in_dir, paste0("R_", case_target, "_reviews_per_year.png"))

stopifnot(file.exists(in_f))

df <- arrow::read_feather(in_f) %>% mutate(post_date_norm = as_date(post_date_norm)) %>% filter(!is.na(post_date_norm)) %

tab_year <- df %>% count(year, name = "n") %>% arrange(year)

p_year <- ggplot(tab_year, aes(x = year, y = n)) + geom_line() + geom_point(size = 1.1) + scale_x_continuous(breaks = seq

ggsave(out_png, p_year, width = 7.5, height = 4.5, dpi = 200)
```

Закрыть

```
## L. README

*Reproducibility*: All figures and tables in the thesis
chapter were generated by the scripts listed in Appendix A.
The pipeline is designed to be platform-independent
(relative paths, UTF-8). To rebuild all outputs, see Section
E. How to run. For data privacy/licensing reasons, raw
exports are not published; only aggregated outputs are
versioned.
```

L. README

Reproducibility: All figures and tables in the thesis chapter were generated by the scripts listed in Appendix A. The pipeline is designed to be platform-independent (relative paths, UTF-8). To rebuild all outputs, see Section E. How to run. For data privacy/licensing reasons, raw exports are not published; only aggregated outputs are versioned.