FVON Sensor Intercomparison: Data Processing and Analysis

Michela Martinelli

2026-03-23

Table of contents

1 Summary and Code Structure

OBJECTIVE

This script processes and compares oceanographic sensor data from multiple instruments (e.g. CTD, Moana, Wisens, Starmon, Tegaru). It identifies stable depth intervals based on CTD data, then compares other sensors' readings for depth, temperature, salinity, and pressure against the CTD reference. This workflow ensures thorough ingestion, cleaning, merging, quality checking, stable interval identification, statistical summary, comparative offset analysis, and visualization of multi-sensor underwater measurement data.

WORKFLOW OVERVIEW

- **Setup and Library Loading**
    - Load required R packages for data mining, plotting, and analyses.
    - Ask the user to specify the base directory containing sensor data files.
    - Optionally delete existing 'combined_*.csv' files after user confirmation.
- **Individual Sensor Data Processing and visual inspection**
    - Define functions to load and parse raw sensor CSV/XLSX files from each sensor folder (e.g. CTD, Moana, Wisens, Starmon A/B, Tegaru A/B).
    - Convert timestamps to UTC and transform pressure readings to depth where needed.
    - Combine data files per sensor and save consolidated 'combined_data_<sensor>.csv'.
    - Generate time-versus-depth plots f for each sensor's combined data to inspect data quality.
- **Merging Sensor Datasets**
    - Load all combined sensor CSV files.
    - Extract key common variables: datetime, depth, pressure, temperature, salinity.
    - Clean and standardize column names and datetime handling.
    - Join all sensor data by UTC timestamp.
    - Generate comparative plots across sensors (depth, pressure, temperature, salinity).
- **Identify Stable Depth Intervals from CTD Data**
    - Detect contiguous stable depth intervals where CTD depth > x m for at least xx seconds.
    - Exclude specified seconds at start/end of intervals to account e.g. for sensor stabilization.

    - Assign unique ‘stable_group’ IDs to these intervals and generate the stable_depth_intervals.csv file.

- **Assign Stable Depth Groups to Other Sensors**
    - Starting from each ‘combined_data_<sensor>.csv’ dataset, assign stable group IDs by joining timestamps with stable intervals previously defined.
    - Generate new combined_stable_depth_<sensor>.csv files containing a new column (stable group) indicating the stable depth interval IDs.

- **Compute Summary Statistics**
    - Calculate mean, SD, SE, 95% CI, min, max, count, and time range for variables (depth, pressure, temperature, salinity) grouped by sensor and stable depth interval.
    - Export as stable_depth_sensor_statistics.csv.

- **Compute Sensors Offsets vs. Reference CTD**
    - Calculate average differences (offsets) between each sensor’s measurements and reference CTD for each variable within stable depth intervals.
    - Summarize and export these offsets with confidence intervals as mean_diff_summary_stats.csv.

- **Compare Differences Against Sensor Accuracy**
    - Join sensors manufacturer declared accuracies to the offsets summary previously generated.
    - Generate plots to visualize average offsets for each available variable with 95% CI and indication of manufacturer declared accuracy.

- **Aggregate Differences by Fixed Depth Bins**
    - Define 10 m intervals (bins) spanning across CTD measurements.
    - Aggregate sensor-CTD differences by sensor, variable, and depth bin.
    - Export summaries as sensor_diff_summary_by_fixed_depth_bins.csv.

- **Statistical Analysis**
    - Perform Pearson correlations and linear regressions of sensor differences versus depth bin midpoints for each sensor-variable pair.
    - Summarize slopes, p-values, R^2, and correlation coefficients.

- **Detailed Visualizations**
    - Plot stable depth intervals over time per sensor with group labels.

- Plot depth vs average temperature and salinity with 95% CIs by stable depth interval.
- Plot average difference vs depth bin with accuracy thresholds.
- Plot regression lines of mean difference vs depth per sensor and variable.

OUTPUT FILES

- Stable intervals: stable_depth_intervals.csv
- Combined stable depth sensor data: combined_stable_depth_<sensor>.csv
- Summary statistics: stable_depth_sensor_statistics.csv
- Offsets summary: mean_diff_summary_stats.csv
- Summary vs accuracy: summary_vs_sensor_accuracy.csv
- Depth bin summary: sensor_diff_summary_by_fixed_depth_bins.csv

2 Introduction

The Fishing Vessel Ocean Observing Network (FVON; https://fvon.org/) is an international, interdisciplinary group dedicated to advancing ocean observing in partnership with the global fishing industry. Fishing vessels offer a unique and cost-effective platform for collecting oceanographic data, providing valuable insights into our coastal seas. The global fishing fleets are diverse in terms of methods, practices, and cultural contexts.

To manage the wide range of data and enhance fishing-vessel-based ocean observations globally, FVON establishes best practices, facilitates the adoption of observation technologies, and maximizes the value of ocean data. As a global network, FVON collaboratively gathers rigorous, cost-effective ocean data, contributing to a transformative shift in understanding the changing oceans and supporting sustainable decision-making.

A key aspect of FVON's work is conducting sensor inter-comparisons to increase transparency and understanding of sensor characteristics and performance. FVON aims to develop and widely share operational best practices, offering guidance on the best sensor and gear combinations for specific regional needs worldwide.

In the initial phase, FVON focuses on inter-comparisons of temperature profiling instruments, with a secondary emphasis on conductivity and salinity measurements. Through a participatory approach, FVON rigorously compares different sensor products using peer-reviewed methodologies. This process not only provides valuable feedback to manufacturers, helping them improve their systems and define their market position, but also plays a critical role in ensuring that data is used in the most appropriate way. By enriching the metadata accompanying sensor-collected data, FVON helps ensure that the data is both reliable and useful for end-users, supporting better-informed decision-making and more effective network planning.

This document describes the complete workflow to process sensor data from instruments such as Moana, Wisens, Starmon, and Tegaru sensors and compare them with data from a reference CTD after simultaneous deployment (with the depth detectors at the same height).

3 Download and install Libraries

```
#===  0.1 Load and Install Required Libraries ===
# This section ensures all necessary packages are installed and loaded.
required_packages <- c(
  "tidyverse", "data.table", "ggrepel", "broom",
  "lme4", "lmerTest", "gsw", "readxl", "stringi", "knitr", "rmarkdown"
)

# Function to automatically check, install, and load packages
load_libraries <- function(pkgs) {
  new_pkgs <- pkgs[!(pkgs %in% installed.packages()[, "Package"])]
  if (length(new_pkgs)) {
    message("Installing missing packages: ", paste(new_pkgs, collapse = ", "))
    install.packages(new_pkgs, dependencies = TRUE)
  }
  for (p in pkgs) {
    suppressPackageStartupMessages(library(p, character.only = TRUE))
  }
  message("All libraries loaded successfully!")
}
```

4 Workspace Selection and Cleaning

To start, select the working directory containing sensor subfolders. Optionally, this part of the script removes pre-existing combined CSV files to avoid stale data. It is recommended to run the steps #1, #2 and #3 below separately. Then, in case you want to generate a report use the “Knit” button to run the rest of the script (you might need to install also the `libraries(rmarkdown)` and `(knitr)).`

```
# --- 0.2 Workspace selection and cleaning ---

# Function to prompt the user to enter the path to the base folder containing all sensor s
ubfolders
select_base_folder <- function() {
  if (interactive()) {
    message("Please select the base data folder (contains sensor subfolders)")
    # On Windows use choose.dir; on Mac/Linux use tk_choose.dir from tcltk package
    if (.Platform$OS.type == "windows") {
      folder <- choose.dir(caption = "Select base data folder")
      if (is.na(folder) || folder == "") stop("No folder selected.")
    } else {
      if (!requireNamespace("tcltk", quietly = TRUE)) {
        stop("Package 'tcltk' needed for folder selection on non-Windows OS. Please instal
l it or specify folder manually.")
      }
      folder <- tcltk::tk_choose.dir(caption = "Select base data folder")
      if (is.na(folder) || folder == "") stop("No folder selected.")
    }
  } else {

# When knitting non-interactively, set your folder path here:
    folder <- "C:/…../Italy_data"
```

```r
    message("Running non-interactively, using set folder: ", folder)
  }
  if (!dir.exists(folder)) stop("Folder does not exist: ", folder)
  normalizePath(folder)
}

# In case needed, the following function delete previous elaborations (Delete all combined
*.csv files recursively from base folder with confirmation

delete_all_combined_csv <- function(base_folder) {
  combined_files <- list.files(base_folder, pattern = "^combined.*\\.csv$", recursive = TR
UE, full.names = TRUE)
  if (length(combined_files) == 0) {
    message("No 'combined*.csv' files found to delete in: ", base_folder)
    return(invisible(NULL))
  }
  message("The following combined CSV files will be deleted:\n", paste(combined_files, col
lapse = "\n"))
  if (interactive()) {
    ans <- readline(prompt = "Are you sure you want to delete these files? Type 'yes' to c
onfirm: ")
    if (tolower(ans) == "yes") {
      file.remove(combined_files)
      message("Deleted ", length(combined_files), " files.")
    } else {
      message("Aborted deletion.")
    }
  } else {
    message("Non-interactive mode detected, skipping deletion of combined CSV files.")
  }
}

# Use the functions:
#1
base_folder <- select_base_folder()

#2
delete_all_combined_csv(base_folder)

#3 first create sensor folders within the base folder, then use the following function to
set folder names according to sensors tested (also remember to set the folder names accord
ingly throughout the script):
sensor_folders <- list(
  CTD = file.path(base_folder, "CTD reference sbe9"),
  Moana = file.path(base_folder, "Moana"),
  Wisens = file.path(base_folder, "Wisens"),
  Starmon_a = file.path(base_folder, "Starmon a"),
  Starmon_b = file.path(base_folder, "Starmon b"),
  Tegaru_a = file.path(base_folder, "Tegaru a"),
  Tegaru_b = file.path(base_folder, "Tegaru b")
)
```

5 CTD Data Processing

The CTD sensor data serves as the reference. This step combines all CTD files previously converted to CSV format, ensures proper parsing and visualizes depth over time to allow for an initial inspection of the uploaded data.

Seabird CTD data example:

datetime_UTC,“pressure_dbar”,“temperature_C”,“salinity_PSU”,“depth_m”
2025-05-12 06:05:44,0.951,19.5672,35.7904,0.944
2025-05-12 06:05:44.0864,0.964,19.566,35.7701,0.957
2025-05-12 06:05:44.0864,0.951,19.5646,35.7538,0.944
2025-05-12 06:05:44.1728,0.905,19.5635,35.7414,0.898
2025-05-12 06:05:44.1728,0.964,19.5625,35.7285,0.957
2025-05-12 06:05:44.2592,0.951,19.5617,35.713,0.944
2025-05-12 06:05:44.2592,0.964,19.5619,35.696,0.957
2025-05-12 06:05:44.3456,0.905,19.5621,35.6795,0.898
2025-05-12 06:05:44.3456,0.853,19.5612,35.671,0.846
2025-05-12 06:05:44.432,0.866,19.5609,35.6753,0.859
2025-05-12 06:05:44.432,0.853,19.56,35.6911,0.846
2025-05-12 06:05:44.5184,0.853,19.5566,35.7211,0.846

```r
# === 1. CTD data processing ===

## Define functions:
combine_ctd_csv_files <- function(folder_path, output_csv = "combined_CTD_data.csv") {
  csv_files <- list.files(path = folder_path, pattern = "\\.csv$", full.names = TRUE)
  if (length(csv_files) == 0) stop("No CSV files found in the specified folder.")
    all_data <- lapply(csv_files, function(file) {
    df <- read.csv(file, stringsAsFactors = FALSE)
    if (!("datetime_UTC" %in% names(df))) stop(paste("File", file, "has no 'datetime_UTC'
column."))
    if (!("depth_m" %in% names(df))) df$depth_m <- NA_real_
    df$datetime_UTC <- as.POSIXct(df$datetime_UTC, tz = "UTC")
    df
  }) %>% bind_rows() %>%
    arrange(datetime_UTC)
    write.csv(all_data, file.path(folder_path, output_csv), row.names = FALSE)
  cat("Combined CTD CSV created at:", file.path(folder_path, output_csv), "\n")
}

plot_time_vs_depth <- function(csv_file) {
  df <- read.csv(csv_file, stringsAsFactors = FALSE)
  df$datetime_UTC <- as.POSIXct(df$datetime_UTC, tz = "UTC")
  y_vals <- if (!all(is.na(df$depth_m))) df$depth_m else df$pressure_dbar
  ylim <- rev(range(y_vals, na.rm = TRUE))
  ylab <- ifelse(!all(is.na(df$depth_m)), "Depth (m)", "Depth (Pressure dbar)")
    plot(df$datetime_UTC, y_vals, type = "l", col = "red",
       xlab = "Time (UTC)", ylab = ylab, ylim = ylim, main = "Time vs Depth (CTD Data)")
}
## Use functions:
combine_ctd_csv_files(sensor_folders$CTD)
plot_time_vs_depth(file.path(sensor_folders$CTD, "combined_CTD_data.csv"))
```

6 Moana ZebraTech Data Processing

Moana data files are comma separated .csv that need custom header parsing and pressure-to-depth conversion. This step combine all CSV files, processes and visualizes accordingly.

Moana data file header and column names example:

ZebraTech BLE Version, 3.1.1
Download Location (lat/long), 43.82865,13.71255
Time Zone, UTC
Download Time,12/05/2025 07:57:24
Download Attempts,1
Moana Serial Number,1064
Moana Firmware,MOANA-3.05
Protocol Version,2
Moana calibration date,21/03/2025
Reset Codes, 0x100202
Moana Battery (V),3.62
Max Lifetime Depth (dBar),182.8
Baseline(mBar),1036
Date,Time,Depth Decibar,Temperature C
12/05/2025,07:10:21,0.7,20.132
12/05/2025,07:10:24,2.1,19.977
12/05/2025,07:10:29,3.4,19.723
12/05/2025,07:10:34,3.4,19.369

For Moana, the depth needs to be calculated from the pressure using a function in the R library(gsw).

```
# === 2. Moana data processing ===

## Define functions:

read_moana_csv <- function(file_path) {
  lines <- readLines(file_path, n = 20)
  header_line_index <- which(grepl("^Date,Time,", lines))
  if (length(header_line_index) == 0) stop("Header 'Date,Time,...' not found in file: ", f
ile_path)
  skip <- header_line_index - 1
  df <- read_csv(file_path, skip = skip, col_types = cols(), show_col_types = FALSE) %>%
    mutate(datetime_UTC = as.POSIXct(paste(Date, Time), format = "%d/%m/%Y %H:%M:%S", tz =
"UTC"),
           pressure_dbar = `Depth Decibar`,
           temperature_C = `Temperature C`) %>%
    select(datetime_UTC, pressure_dbar, temperature_C)
  df
}

## change Latitute according to area (e.g. for Italy 43, for Japan 34)

combine_moana_csv_files <- function(folder_path, output_csv = "combined_data_Moana.csv", l
```

```
atitude = 43) {
  files <- list.files(folder_path, pattern = "\\.csv$", full.names = TRUE)
  if (length(files) == 0) stop("No CSV files found in folder.")
  all_data <- lapply(files, function(f) {
    cat("Reading", f, "\n")
    read_moana_csv(f)
  }) %>% bind_rows() %>%
    # gsw_z_from_p calculates the height (negative at sea).
    # use the minus sign (-) to obtain the depth as a positive value
    mutate(depth_m =  -gsw_z_from_p(pressure_dbar, latitude)) %>%
    arrange(datetime_UTC)
    write.csv(all_data, file.path(folder_path, output_csv), row.names = FALSE)
  cat("Combined Moana CSV saved at:", file.path(folder_path, output_csv), "\n")
}

plot_time_vs_depth_moana <- function(csv_file) {
  df <- read.csv(csv_file, stringsAsFactors = FALSE)
  df$datetime_UTC <- as.POSIXct(df$datetime_UTC, tz = "UTC")
  y_vals <- if (!all(is.na(df$depth_m))) df$depth_m else df$pressure_dbar
  ylim <- rev(range(y_vals, na.rm = TRUE))
  ylab <- ifelse(!all(is.na(df$depth_m)), "Depth (m)", "Depth (Pressure dbar)")
  plot(df$datetime_UTC, y_vals, type = "l", col = "gold",
       xlab = "Time (UTC)", ylab = ylab, ylim = ylim, main = "Time vs Depth (Moana Data)")
}
## Use functions:
combine_moana_csv_files(sensor_folders$Moana)

plot_time_vs_depth_moana(file.path(sensor_folders$Moana, "combined_data_Moana.csv"))
```

7 Wisens NKE Instrumentation Data Processing

Wisens data files are comma separated csv that requires cleaning from footer, custom parsing, treatment of possible varied separators, timestamps check and pressure conversion. This step combine all CSV files, processes and visualizes accordingly.

Wisens column names and data file footer example:

Timestamp(Standard) CH0:Pressure(bar) CH1:Temperature(degC) CH2:Conductivity(mS/cm) CH3:Practical_Salinity(PSU) CH4:Depth(m)
12/05/2025 09:10:30 0,298 19,726 49,024 36,143 2,9
12/05/2025 09:10:31 0,329 19,689 49,178 36,303 3,3
12/05/2025 09:10:32 0,354 19,568 49,136 36,373 3,5
12/05/2025 09:10:33 0,345 19,619 49,165 36,353 3,4
12/05/2025 09:10:34 0,354 19,603 49,329 36,502 3,5
12/05/2025 09:10:35 0,354 19,581 49,494 36,660 3,5
12/05/2025 09:10:36 0,343 19,448 49,448 36,738 3,4
12/05/2025 09:10:37 0,356 19,433 49,553 36,839 3,5

<WISENS>

<PROBE eH0="WiSens 300" eH2="0x59F3" eH3="2.0.13"/>
<COMMENT eX0=" "/>
<BATTERY eB0="L" eB1="13000 mAh" eB2="2025-02-05T00:00:00"/>
<STATUS eS0="1.7 %" eS1="96.1 %"/>
<SENSOR_0 eS0="P" eS1="Keller" eS2="30 bar" eS3="1">
<CHANNEL_0 eCx="1" eC0="Pressure" eC1="bar" eC2="3" eC4="1" eC5="1013.0">
<CALIBRATION>
<TYPE eT0="Multipoint"/>
<POINT_01 eP0="0.0000" eP1="-0.0"/>
</CALIBRATION>
</CHANNEL_0>
</SENSOR_0>
<SENSOR_2 eS0="T" eS1="" eS2="" eS3="1">
<CHANNEL_0 eCx="1" eC0="Temperature" eC1="degC" eC2="3" eC4="1">
<CALIBRATION>
<TYPE eT0="Polynomial3" eC0="6.71965244e-06" eC1="-4.48708600e-04" eC2="1.01187046e+00" eC3="-1.26596546e-01"/>
<POINT_01 eP0="2.0534" eP1="2.2"/>
<POINT_02 eP0="5.2248" eP1="5.3"/>
<POINT_03 eP0="10.3030" eP1="10.3"/>
<POINT_04 eP0="15.2359" eP1="15.3"/>
<POINT_05 eP0="20.1055" eP1="20.1"/>
<POINT_06 eP0="24.9811" eP1="25.0"/>
<POINT_07 eP0="29.8928" eP1="29.9"/>
<POINT_08 eP0="34.8599" eP1="34.8"/>
</CALIBRATION>
</CHANNEL_0>
</SENSOR_2>
<SENSOR_4 eS0="C" eS1="" eS2="" eS3="1">
<CHANNEL_0 eCx="1" eC0="Conductivity" eC1="mS/cm" eC2="3" eC4="1">
<CALIBRATION>
<TYPE eT0="Multipoint"/>
<POINT_01 eP0="32.6730" eP1="32.8"/>
<POINT_02 eP0="51.6230" eP1="51.7"/>
</CALIBRATION>
</CHANNEL_0>
</SENSOR_4>
<SENSOR_9 eS0="Sp" eS1="" eS2="" eS3="1">
<CHANNEL_0 eCx="1" eC0="Practical_Salinity" eC1="PSU" eC2="3"/>
</SENSOR_9>
<SENSOR_12 eS0="D" eS1="" eS2="" eS3="1">
<CHANNEL_0 eCx="1" eC0="Depth" eC1="m" eC2="1" eC5="45.0"/>
</SENSOR_12>
<MEASURE>
<SAMPLING eS0="Continuous"/>
<START eS0="Manual"/>

```
 # <STOP eS0="Manual"/>
 # </MEASURE>
 # <EXPORT eX0="Standard (yyyy-mm-dd hh:mm:ss)" eX1="," eX2="
 # </WISENS>
```

```r
# === 3. Wisens data processing ===
## Define functions:
read_wisens_csv <- function(file_path) {
  lines <- str_trim(readLines(file_path, warn = FALSE))
  lines <- lines[lines != ""]
  candidate_lines <- lines[str_detect(lines, "^\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2}")
]
  if (length(candidate_lines) == 0) {
    warning("No timestamped data lines found in ", file_path)
    return(NULL)
  }

  sep <- ifelse(str_detect(candidate_lines[1], ";"), ";", ",")
  line_col_counts <- sapply(candidate_lines, function(x) length(strsplit(x, sep)[[1]]))
  valid_lines <- candidate_lines[line_col_counts == 6]
  if (length(valid_lines) == 0) {
    warning("No valid 6-column lines in ", file_path)
    return(NULL)
  }

  clean_lines <- gsub('"', '', valid_lines)
  data_text <- paste(clean_lines, collapse = "\n")
    read_func <- if (sep == ";") read_csv2 else read_csv
  df <- read_func(I(data_text), col_names = FALSE, na = c("", "nan", "NaN"), show_col_type
s = FALSE)
  if (ncol(df) < 6) {
    warning("Less than 6 columns in data from ", file_path)
    return(NULL)
  }

#the following function was needed in case of some of the Japanese datasets
  # pressure_to_depth <- function(pressure_dbar, latitude = 34) {
  #depth <- -gsw::gsw_z_from_p(pressure_dbar, latitude = latitude)
  #}

#check the number of columns in select
  df <- df %>%
    select(1:6) %>%
    rename(timestamp_str = X1, pressure_bar = X2, temperature_C = X3,
           conductivity_mScm = X4, salinity_PSU = X5, depth_m = X6) %>%
    mutate(
      datetime_CEST = as.POSIXct(str_trim(timestamp_str), format = "%Y-%m-%d %H:%M:%S", tz
= "Etc/GMT-2"),
      datetime_UTC = with_tz(datetime_CEST, "UTC"),

# if needed change time zone
      #datetime_JST = as.POSIXct(str_trim(timestamp_str), format = "%Y-%m-%d %H:%M:%S", tz
= "Etc/GMT-9"),
      #datetime_UTC = with_tz(datetime_JST, "UTC"),
```

```
      pressure_bar = as.numeric(str_trim(pressure_bar)),
      pressure_dbar = pressure_bar * 10,
      temperature_C = as.numeric(str_trim(temperature_C)),
      conductivity_mScm = as.numeric(str_trim(conductivity_mScm)),
      salinity_PSU = as.numeric(str_trim(salinity_PSU)),
      depth_m = as.numeric(str_trim(depth_m))
    ) %>%
    filter(!is.na(datetime_UTC)) %>%
    filter(!(is.na(pressure_dbar) & is.na(temperature_C) & is.na(conductivity_mScm) & is.n
a(salinity_PSU) & is.na(depth_m))) %>%
    select(-timestamp_str, -pressure_bar)
    df
}

combine_wisens_data <- function(folder_path, output_csv = "combined_data_Wisens.csv") {
  files <- list.files(folder_path, pattern = "\\.csv$", full.names = TRUE)
  if (length(files) == 0) stop("No CSV files in folder.")
    data_list <- lapply(files, function(f) {
    cat("Reading:", f, "\n")
    read_wisens_csv(f)
  })
    data_list <- Filter(Negate(is.null), data_list)
  if (length(data_list) == 0) stop("No valid Wisens data found.")
    combined <- bind_rows(data_list) %>% arrange(datetime_UTC)
  write.csv(combined, file.path(folder_path, output_csv), row.names = FALSE)
  cat("Combined Wisens CSV saved at:", file.path(folder_path, output_csv), "\n")
}

plot_time_vs_depth_wisens <- function(csv_file) {
  df <- read.csv(csv_file, stringsAsFactors = FALSE)
  df$datetime_UTC <- as.POSIXct(df$datetime_UTC, tz = "UTC")
  ylim <- rev(range(df$depth_m, na.rm = TRUE))
  plot(df$datetime_UTC, df$depth_m, type = "l", col = "pink",
       xlab = "Time (UTC)", ylab = "Depth (m)", ylim = ylim,
       main = "Time vs Depth (Wisens Data)")
}
## Use functions:
combine_wisens_data(sensor_folders$Wisens)
plot_time_vs_depth_wisens(file.path(sensor_folders$Wisens, "combined_data_Wisens.csv"))
```

8 Starmon Starmon Data Processing

Starmon data files are Excel files that requires with 2 tabs, no header, no footer. They require custom parsing and depth to pressure conversion. This step processes and visualizes accordingly the data in the DAT tab.

Starmon column names example:

DAT tab
Date & Time Temp(°C) Depth(m)
12/05/2025 10:11:00 26.41 -0.16
12/05/2025 10:11:01 26.42 -0.16
12/05/2025 10:11:02 26.42 -0.16

12/05/2025 10:11:03 26.43 -0.15
12/05/2025 10:11:04 26.44 -0.16
12/05/2025 10:11:05 26.45 -0.16
12/05/2025 10:11:06 26.46 -0.15
12/05/2025 10:11:07 26.47 -0.16
12/05/2025 10:11:08 26.49 -0.15

For Starmon, the pressure needs to be calculated from the depth using a function in the R library(gsw).

```
# === 4. Starmon data processing ===

## Define functions:
read_starmon_simple <- function(file_path) {
  cat("Reading file:", basename(file_path), "\n")
  df <- read_excel(file_path, sheet = 1)
  names_lower <- tolower(names(df))
   required_cols <- c("date & time", "temp(°c)", "depth(m)")
  if (!all(required_cols %in% names_lower)) stop("Missing required columns in ", basename(
file_path))

  date_col <- names(df)[which(names_lower == "date & time")]
  temp_col <- names(df)[which(names_lower == "temp(°c)")]
  depth_col <- names(df)[which(names_lower == "depth(m)")]

  df <- df %>%
    mutate(
      datetime_parsed = parse_date_time(.data[[date_col]], orders = c("dmy HMS", "dmy HM",
"ymd HMS", "ymd HM")),
      datetime_CEST = force_tz(datetime_parsed, "Europe/Berlin"),
      temperature_C = as.numeric(.data[[temp_col]]),
      depth_m = as.numeric(.data[[depth_col]])
    ) %>%
    filter(!is.na(datetime_CEST)) %>%
    mutate(datetime_UTC = with_tz(datetime_CEST, "UTC"),

#Pressure calculation from depth: gsw requires 'z' altitude (negative under sea)
#change latitude according to test location
      pressure_dbar = gsw::gsw_p_from_z(z = -depth_m, latitude = 43)
   )%>%
    select(datetime_UTC, temperature_C, depth_m, pressure_dbar)
  cat(" - Read", nrow(df), "rows\n")
  df
}

combine_starmon_simple <- function(folder_path, output_csv = NULL) {
  if (!dir.exists(folder_path)) stop("Folder does not exist: ", folder_path)

  xlsx_files <- list.files(folder_path, pattern = "^[^~].*\\.xlsx$", full.names = TRUE, ig
nore.case = TRUE)
  if (length(xlsx_files) == 0) stop("No Excel files found in folder: ", folder_path)
```

```
  data_list <- lapply(xlsx_files, function(f) {
    tryCatch(read_starmon_simple(f), error = function(e) {
      warning("Error reading ", basename(f), ": ", e$message)
      NULL
    })
  }) %>% Filter(Negate(is.null), .)

  if (length(data_list) == 0) stop("No valid data found in folder: ", folder_path)
  combined <- bind_rows(data_list) %>% arrange(datetime_UTC)
    if (!is.null(output_csv)) {
    write.csv(combined, output_csv, row.names = FALSE)
    cat("Combined Starmon CSV saved to:", output_csv, "\n")
  }
  combined
}

plot_time_vs_depth_starmon_simple <- function(data_or_csv) {
  df <- if (is.character(data_or_csv)) {
    read.csv(data_or_csv, stringsAsFactors = FALSE) %>% mutate(datetime_UTC = as.POSIXct(d
atetime_UTC, tz = "UTC"))
  } else {
    data_or_csv
  }
  ylim <- rev(range(df$depth_m, na.rm = TRUE))
  plot(df$datetime_UTC, df$depth_m, type = "l", col = "forestgreen",
       xlab = "Time (UTC)", ylab = "Depth (m)", ylim = ylim,
       main = "Time vs Depth (Starmon Data)")
}
## Use functions:
combine_starmon_simple(sensor_folders$Starmon_a, output_csv = file.path(sensor_folders$Sta
rmon_a, "combined_data_Starmon_a.csv")) %>%
 plot_time_vs_depth_starmon_simple()
combine_starmon_simple(sensor_folders$Starmon_b, output_csv = file.path(sensor_folders$Sta
rmon_b, "combined_data_Starmon_b.csv")) %>%
 plot_time_vs_depth_starmon_simple()
```

9 Tegaru JFE Advantech Data Processing

Tegaru data files are comma separated csv with a custom header and data format that requires parsing for key variables and and pressure conversion. This step combine all CSV files, processes and visualizes accordingly.

Tegaru data file header and column names example:

// Smart ACT
// Version 1.02
// File Date:2025/05/12 08:51:41

[Head]
SondeName=ACTDf-BT
SondeNo=0BG1008
SensorType=D2T2C1C2B0

SensorType2=0402020002
SensorType3=66660
Channel=5
MeasMode=0
PreHeat=950
BurstTime=0
BurstCnt=0
Interval=1
DepthZero=114438
SampleCnt=26966
StartTime=2025/05/12 08:02:59
EndTime=2025/05/12 08:47:56
StopTime=2025/05/12 08:51:28
DepAdjRho=1.025
Status=01000
Position=43.8288662,13.7124303
CoefDate=2024/07/25
Ch1=-3.25703e-01,+1.82218e-04,+8.84756e-
11,+0.00000e+00,+0.00000e+00,+0.00000e+00,+0.00000e+00,+0.00000e+00,
Ch2=-1.191836e+01,+1.308083e-03,-1.344373e-08,+1.539756e-
13,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,
Ch3=-4.214672e-
02,+3.520517e+01,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.0
00000e+00,
Ch4=+0.000000e+00,+1.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0
.000000e+00,+0.000000e+00,
Ch5=+4.022560E-02,+1.607745E-
03,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,+0.000000e+00,

[Item]
Date,Depth[m],Press.[MPa],Temp.[degC],Salinity,Cond.[mS/cm],Battery[V],Status,
2025/05/12 08:02:59,2.79,0.0281,19.38,36.76,49.39,4.14,0,
2025/05/12 08:03:00,2.84,0.0285,19.35,36.82,49.45,4.14,0,
2025/05/12 08:03:00,2.91,0.0292,19.32,36.92,49.54,4.14,0,
2025/05/12 08:03:00,2.98,0.0299,19.30,36.98,49.59,4.14,0,
2025/05/12 08:03:00,3.01,0.0302,19.29,36.94,49.52,4.14,0,
2025/05/12 08:03:00,2.97,0.0299,19.30,36.68,49.23,4.14,0,
2025/05/12 08:03:00,2.92,0.0293,19.34,36.66,49.24,4.14,0,
2025/05/12 08:03:00,2.90,0.0291,19.33,36.83,49.44,4.14,0,

```
# === 5. Tegaru data processing ===

# If strange characters are in the header, replace it; check columns consistency in every
data file
# The following additional library might be needed to handle Japanese characters in the he
ader
#library(stringi)

## Define functions:
```

```r
read_tegaru_simple <- function(file_path) {
  cat("Reading Tegaru file:", basename(file_path), "\n")
  lines <- readLines(file_path, warn = FALSE)
# in case of Japanese characters read lines assuming Latin1 encoding (adjust if you know a
different encoding)
  #raw_lines <- readLines(file_path, encoding = "latin1", warn = FALSE)
# in case of Japanese characters convert lines to UTF-8 to fix invalid byte sequences
  #lines <- stri_enc_toutf8(raw_lines)
  item_line_idx <- which(str_detect(lines, fixed("[Item]")))
  if (length(item_line_idx) == 0) stop("[Item] line not found in file: ", file_path)
  header_line_idx <- item_line_idx + 1
  header_line <- lines[header_line_idx]
  sep <- ifelse(str_detect(header_line, ";"), ";", ",")

  col_names <- make.names(str_trim(str_split(header_line, sep)[[1]]), unique = TRUE)
# in case of Japanese characters clean column names with ASCII transliteration to remove s
trange characters
  #col_names <- iconv(col_names, from = "", to = "ASCII//TRANSLIT")
  data_lines <- lines[(header_line_idx + 1):length(lines)]
  if (length(data_lines) == 0) {
    cat(" - No data lines after header\n")
    return(tibble())
  }

  data_strings <- unlist(str_split(data_lines, sep)) %>% str_trim()
  data_strings <- data_strings[data_strings != ""]
  n_cols <- length(col_names)
  n_rows <- floor(length(data_strings) / n_cols)
  if (n_rows == 0) {
    cat(" - No complete data rows\n")
    return(tibble())
  }

  mat <- matrix(data_strings[1:(n_rows * n_cols)], ncol = n_cols, byrow = TRUE)
  df <- as_tibble(as.data.frame(mat, stringsAsFactors = FALSE))
  names(df) <- col_names
  if (!"Date" %in% col_names) stop("Date column missing in file: ", file_path)
  df <- df %>%
    mutate(datetime_CEST = as.POSIXct(Date, format = "%Y/%m/%d %H:%M:%S", tz = "Europe/Rom
e")) %>%
    filter(!is.na(datetime_CEST)) %>%
    mutate(datetime_UTC = with_tz(datetime_CEST, "UTC")) %>%
    #mutate(datetime_JST = as.POSIXct(Date, format = "%Y/%m/%d %H:%M:%S", tz = "Etc/GMT-9"
)) %>%
    #filter(!is.na(datetime_JST)) %>%
    #mutate(datetime_UTC = with_tz(datetime_JST, "UTC")) %>%
    select(-Date)
  depth_col <- grep("depth", names(df), ignore.case = TRUE, value = TRUE)[1]
  if (is.na(depth_col)) stop("Depth column missing in file: ", file_path)
  df <- df %>% mutate(depth_m = as.numeric(.data[[depth_col]])) %>% select(-all_of(depth_c
ol))
  pressure_cols_mpa <- grep("Press.*MPa", names(df), ignore.case = TRUE, value = TRUE)
  if (length(pressure_cols_mpa) > 0) {
    pcol <- pressure_cols_mpa[1]
    df <- df %>% mutate(pressure_dbar = as.numeric(.data[[pcol]]) * 100) %>% select(-all_o
f(pcol))
```

```
  } else {
    pressure_any <- grep("press", names(df), ignore.case = TRUE, value = TRUE)
    if (length(pressure_any) > 0) {
      pcol <- pressure_any[1]
      df <- df %>% mutate(pressure_dbar = as.numeric(.data[[pcol]])) %>% select(-all_of(pc
ol))
    } else {
      df <- df %>% mutate(pressure_dbar = NA_real_)
    }
  }
#in case of Japanese characters substitute the pressure_col function with the following:
  #pressure_cols_mpa <- grep("Press.*MPa", names(df), ignore.case = TRUE, value = TRUE)
    #if (length(pressure_cols_mpa) > 0) {
    #pcol <- pressure_cols_mpa[1]
    # Clean and convert pressure column safely
    #pressure_raw <- str_trim(df[[pcol]])
    # Remove all characters except digits, dot, minus sign
    #pressure_clean <- as.numeric(str_replace_all(pressure_raw, "[^0-9.-]", ""))
    # Warn if non-numeric coercion happens
    #if (any(is.na(pressure_clean) & !is.na(pressure_raw))) {
      #warning(sprintf("Non-numeric values found in pressure column '%s' in file %s, coerc
ed to NA", pcol, basename(file_path)))
    #}
   #df <- df %>%
      #mutate(pressure_dbar = pressure_clean * 100) %>%
      #select(-all_of(pcol))
  #} else {
    #pressure_any <- grep("press", names(df), ignore.case = TRUE, value = TRUE)
    #if (length(pressure_any) > 0) {
      #pcol <- pressure_any[1]
      #pressure_raw <- str_trim(df[[pcol]])
      #pressure_clean <- as.numeric(str_replace_all(pressure_raw, "[^0-9.-]", ""))
      #if (any(is.na(pressure_clean) & !is.na(pressure_raw))) {
        #warning(sprintf("Non-numeric values found in pressure column '%s' in file %s, coe
rced to NA", pcol, basename(file_path)))
      #}
      #df <- df %>%
        #mutate(pressure_dbar = pressure_clean) %>%
        #select(-all_of(pcol))
    #} else {
      #df <- df %>% mutate(pressure_dbar = NA_real_)
    #}
  #}
  other_cols <- setdiff(names(df), c("datetime_UTC", "depth_m", "pressure_dbar"))
  for (col in other_cols) {
    df[[col]] <- suppressWarnings(as.numeric(df[[col]]))
  }
  df <- df %>% arrange(datetime_UTC)
  cat(" - Read", nrow(df), "rows with pressure in dbar\n")
  df
}

combine_tegaru_simple <- function(folder_path, output_csv = NULL) {
  if (!dir.exists(folder_path)) stop("Folder does not exist: ", folder_path)
  files <- list.files(folder_path, pattern = "\\.csv$", full.names = TRUE)
  if (length(files) == 0) stop("No CSV files found in folder: ", folder_path)
```

```
  data_list <- lapply(files, function(f) {
    tryCatch(read_tegaru_simple(f), error = function(e) {
      warning("Error reading ", basename(f), ": ", e$message)
      NULL
    })
  }) %>% Filter(Negate(is.null), .)
  if (length(data_list) == 0) stop("No valid Tegaru data found in folder.")
    combined <- bind_rows(data_list) %>%
    arrange(datetime_UTC)
    if (!is.null(output_csv)) {
    write.csv(combined, output_csv, row.names = FALSE)
    cat("Combined Tegaru data saved to:", output_csv, "\n")
  }
  combined
}

plot_time_vs_depth_tegaru_simple <- function(df) {
  stopifnot(all(c("datetime_UTC", "depth_m") %in% names(df)))
  if (nrow(df) == 0) stop("Empty dataframe.")

  ggplot(df, aes(x = datetime_UTC, y = depth_m)) +
    geom_line(color = "lightblue") + scale_y_reverse() +
    labs(title = "Time vs Depth (Tegaru Data)", x = "Time (UTC)", y = "Depth (m)") +
    theme_minimal()
}

## Use functions:

combine_tegaru_simple(sensor_folders$Tegaru_a, output_csv = file.path(sensor_folders$Tegar
u_a, "combined_data_Tegaru_a.csv")) %>%
plot_time_vs_depth_tegaru_simple()

combine_tegaru_simple(sensor_folders$Tegaru_b, output_csv = file.path(sensor_folders$Tegar
u_b, "combined_data_Tegaru_b.csv")) %>%
plot_time_vs_depth_tegaru_simple()
```

10 Merge All Data and Overview Plots

Now that individual sensor datasets are ready, align them by timestamp and create comparative plots.

```
# === 6. Align all combined files and plot ===

# ---- Utility functions ----
clean_names <- function(x) {
  x <- gsub("[^A-Za-z0-9_]", "_", x)
  x <- gsub("_+", "_", x)
  tolower(x)
}
standardize_var <- function(colname) {
  cn <- tolower(colname)
  if (grepl("depth", cn)) return("depth")
  if (grepl("press|pres|pressure", cn)) return("pressure")
  if (grepl("temp", cn)) return("temperature")
```

```r
  if (grepl("sal", cn)) return("salinity")
  NA_character_
}
get_prefix_from_col <- function(colname) {
  parts <- unlist(strsplit(colname, "_", fixed = TRUE))
  parts[1]
}

# ---- Robust datetime parser with tz handling ----
parse_datetime_vec <- function(x, tz_in = NULL) {
  if (inherits(x, "POSIXt")) {
    out <- as.POSIXct(x, tz = "UTC")
    return(out)
  }
  if (all(is.na(x))) return(as.POSIXct(NA_real_, origin = "1970-01-01", tz = "UTC"))
  if (is.numeric(x)) {
    if (all(x > 1e9, na.rm = TRUE)) {
      return(as.POSIXct(x, origin = "1970-01-01", tz = "UTC"))
    } else {
      return(as.POSIXct((x - 25569) * 86400, origin = "1970-01-01", tz = "UTC"))
    }
  }
  s <- as.character(x)
  orders <- c("Ymd HMS", "Ymd HM", "Ymd", "dmy HMS", "dmy HM", "dmy",
              "mdy HMS", "mdy HM", "mdy", "Y-m-d H:M:S", "Y-m-d H:M", "Y-m-d")
  if (!is.null(tz_in)) {
    parsed <- parse_date_time(s, orders = orders, tz = tz_in, exact = FALSE)
    parsed <- with_tz(parsed, tzone = "UTC")
  } else {
    parsed <- parse_date_time(s, orders = orders, tz = "UTC", exact = FALSE)
    if (all(is.na(parsed))) parsed <- suppressWarnings(ymd_hms(s, tz = "UTC"))
  }
  parsed
}

# ---- Preferred datetime detection (prefer explicit UTC) ----
choose_datetime_info <- function(colnames_vec) {
  cn <- tolower(colnames_vec)
  if ("datetime_utc" %in% cn) return(list(col = colnames_vec[which(cn == "datetime_utc")[1
]], tz_in = "UTC"))
  if ("datetime_cest" %in% cn) return(list(col = colnames_vec[which(cn == "datetime_cest")
[1]], tz_in = "Europe/Paris"))
  # fallback any datetime-like
  dt_idx <- grep("datetime|time", cn, ignore.case = TRUE)
  if (length(dt_idx)) return(list(col = colnames_vec[dt_idx[1]], tz_in = "UTC"))
  return(list(col = NA_character_, tz_in = NULL))
}

# ---- Read & join combined files (general) ----
read_all_combined_files <- function(base_folder) {
  combined_files <- list.files(base_folder, pattern = "^combined.*\\.csv$", recursive = TR
UE, full.names = TRUE)
  if (length(combined_files) == 0) stop("No combined*.csv files found.")
  lapply(combined_files, function(f) {
    df <- read_csv(f, show_col_types = FALSE)
    names(df) <- clean_names(names(df))
```

```r
    dt_info <- choose_datetime_info(names(df))
    if (!is.na(dt_info$col)) {
      df$datetime_UTC <- parse_datetime_vec(df[[dt_info$col]], tz_in = dt_info$tz_in)
    } else {
      df$datetime_UTC <- NA_real_
    }
    df$folder_name <- basename(dirname(f))
    df$source_file <- basename(f)
    df
  })
}
join_all_combined_files <- function(all_data_list) {
  renamed <- lapply(all_data_list, function(df) {
    folder <- unique(df$folder_name)
    stopifnot(length(folder) == 1)
    var_cols <- setdiff(names(df), c("datetime_utc", "datetime_UTC", "folder_name", "sourc
e_file"))
    new_names <- paste0(folder, "_", var_cols)
    names(df)[names(df) %in% var_cols] <- new_names
    df %>% select(datetime_UTC, all_of(new_names))
  })
  reduce(renamed, full_join, by = "datetime_UTC") %>% arrange(datetime_UTC)
}

# ---- Plots ----
plot_all_combined_time_vs_depth <- function(df_all) {
  depth_cols <- grep("_depth_m$|\\bdepth_m$|_depth$|\\bdepth\\b", names(df_all), ignore.ca
se = TRUE, value = TRUE)
  if (length(depth_cols) == 0) return(NULL)
  plot_data <- df_all %>% select(datetime_UTC, all_of(depth_cols)) %>% pivot_longer(-datet
ime_UTC, names_to = "file", values_to = "depth") %>% filter(!is.na(depth))
  ggplot(plot_data, aes(x = datetime_UTC, y = depth, color = file)) + geom_point(alpha = 0
.6, size = 0.7) + scale_y_reverse() + labs(title = "Time vs Depth (All sensors)", x = "Tim
e (UTC)", y = "Depth (m)", color = "Sensor") + theme_minimal()
}
plot_combined_pressure_vs_time <- function(df_all) {
  pressure_cols <- grep("press|pressure", names(df_all), ignore.case = TRUE, value = TRUE)
  if (length(pressure_cols) == 0) return(NULL)
  plot_data <- df_all %>% select(datetime_UTC, all_of(pressure_cols)) %>% pivot_longer(-da
tetime_UTC, names_to = "sensor", values_to = "pressure") %>% filter(!is.na(pressure))
  ggplot(plot_data, aes(x = datetime_UTC, y = pressure, color = sensor)) + geom_point(alph
a = 0.6, size = 0.7) + labs(title = "Pressure vs Time (All sensors)", x = "Time (UTC)", y
= "Pressure (dbar)", color = "Sensor") + theme_minimal()
}
plot_pressure_vs_temp <- function(df_all) {
  pressure_cols <- grep("press|pressure", names(df_all), ignore.case = TRUE, value = TRUE)
  temp_cols <- grep("temp|temperature", names(df_all), ignore.case = TRUE, value = TRUE)
  if (length(pressure_cols) == 0 || length(temp_cols) == 0) return(NULL)
  out <- tibble()
  for (p in pressure_cols) {
    prefix <- get_prefix_from_col(p)
    tcols <- temp_cols[startsWith(temp_cols, paste0(prefix, "_"))]
    if (length(tcols) == 0) next
    for (t in tcols) {
      tmp <- df_all %>% select(all_of(c(p, t))) %>% filter(!is.na(.data[[p]]), !is.na(.dat
a[[t]])) %>% rename(pressure = !!p, temperature = !!t) %>% mutate(sensor = prefix)
```

```r
      out <- bind_rows(out, tmp)
    }
  }
  if (nrow(out) == 0) return(NULL)
  ggplot(out, aes(x = temperature, y = pressure, color = sensor)) + geom_point(alpha = 0.6
, size = 0.7) + scale_y_reverse() + labs(title = "Pressure vs Temperature (All sensors)",
x = "Temperature", y = "Pressure (dbar)", color = "Sensor") + theme_minimal()
}
plot_depth_vs_temp_sal <- function(df_all, which = c("temperature", "salinity")) {
  which <- match.arg(which)
  depth_cols <- grep("_depth_m$|\\bdepth_m$|_depth$|\\bdepth\\b", names(df_all), ignore.ca
se = TRUE, value = TRUE)
  var_pattern <- if (which == "temperature") "temp|temperature" else "sal|salinity"
  var_cols <- grep(var_pattern, names(df_all), ignore.case = TRUE, value = TRUE)
  if (length(depth_cols) == 0 || length(var_cols) == 0) return(NULL)
  out <- tibble()
  for (d in depth_cols) {
    prefix <- get_prefix_from_col(d)
    vcols <- var_cols[startsWith(var_cols, paste0(prefix, "_"))]
    if (length(vcols) == 0) next
    for (v in vcols) {
      tmp <- df_all %>% select(all_of(c(d, v))) %>% filter(!is.na(.data[[d]]), !is.na(.dat
a[[v]])) %>% rename(depth = !!d, var = !!v) %>% mutate(sensor = prefix)
      out <- bind_rows(out, tmp)
    }
  }
  if (nrow(out) == 0) return(NULL)
  ggplot(out, aes(x = var, y = depth, color = sensor)) + geom_point(alpha = 0.6, size = 0.
7) + scale_y_reverse() + labs(title = paste("Depth vs", which, "(All sensors)"), x = which
, y = "Depth (m)", color = "Sensor") + theme_minimal()
}

# ---- Initial combined read & quick plots ----
all_data_list <- read_all_combined_files(base_folder)
joined_df <- join_all_combined_files(all_data_list)

# if a selection based on time is needed use the following function:
#joined_df <- joined_df %>%
  #filter(datetime_UTC >= as.POSIXct("2025-06-11 10:00:00", tz = "UTC")&
           #datetime_UTC <= as.POSIXct("2025-06-11 11:00:00", tz = "UTC"))

print(plot_combined_pressure_vs_time(joined_df))
print(plot_all_combined_time_vs_depth(joined_df))
print(plot_pressure_vs_temp(joined_df))
print(plot_depth_vs_temp_sal(joined_df, "temperature"))
print(plot_depth_vs_temp_sal(joined_df, "salinity"))
```

11 Stable depth intervals, Comparative Plots and Basic Statistics

According to selected conditions (e.g. depth > 1 m with consistent - depth variation = 0.1 m- record for > 60s, and excludes a few seconds at the beginning and at the end to let the sensor measurement stabilize and/or to avoid any problems with the sensor time stamp alignment where needed), this part of the script identifies "stable" depth intervals based on CTD data and assign consecutive stable group IDs (a `stable_depth_intervals.csv` file is produced).
The depth_tol (depth tolerance) parameter used defines the maximum depth variation between consecutive records allowed for the probe to be considered "still" or "at stable depth." In fact, even when the vessel is stationary, the probes may experience small depth variations due to waves or the vessel's rolling. If depth_tol = 0.1, the algorithm accepts oscillations of up to 10 centimeters.

Then the script joins sensor records with the identified stable group IDs and produce `combined_stable_depth_<sensor>.csv` files. Diagnostic plots are then generated using these files: Depth vs Time (points belonging to stable depth intervals only), Depth vs mean Temperature per sensor/stable depth ID (with 95% CI), Depth vs mean Salinity per sensor/stable depth ID (with 95% CI).

The last part of this section produce the file `stable_depth_sensor_statistics.csv` containing for each sensor/stable_group/variable (*temperature/salinity/depth/pressure)* combination, the number of observations (n), mean, standard deviation (sd) , standard error (se), and other information (i.e. t_crit, ci_lower, ci_upper, min, max, datetime_min, datetime_max). This file is the base for subsequent analyses.

Note the required inputs expected to be set or produced earlier in the script/environment:
- sensor_folders: named list of folder paths keyed by sensor (must include CTD),
- CTD combined file: <CTD folder>/combined_CTD_data.csv,
- Helper functions used by this section: clean_names(), choose_datetime_info(), parse_datetime_vec(), standardize_var(), get_prefix_from_col(),
- packages loaded (dplyr, data.table, ggplot2, etc.).

```
# --- 7. Stable depth intervals based on CTD depth > 1 m with consistent record for > 60s, exclude first XXs and last XXs, assign unique consecutive stable group IDs; we remove a few seconds at the beginning to let the sensor measurement stabilize and at the beginning and end to avoid any problems with the sensor time stamp alignment

# Load CTD data and identify stable depth intervals
ctd_folder <- sensor_folders$CTD
combined_CTD_path <- file.path(ctd_folder, "combined_CTD_data.csv")
if (!file.exists(combined_CTD_path)) stop("CTD combined file not found: ", combined_CTD_path)
combined_CTD_data <- read_csv(combined_CTD_path, show_col_types = FALSE)
names(combined_CTD_data) <- clean_names(names(combined_CTD_data))
dt_info_ctd <- choose_datetime_info(names(combined_CTD_data))
if (!is.na(dt_info_ctd$col)) {
  combined_CTD_data$datetime_UTC <- parse_datetime_vec(combined_CTD_data[[dt_info_ctd$col]], tz_in = dt_info_ctd$tz_in)
} else combined_CTD_data$datetime_UTC <- NA_real_
```

```r
ctd_depth_col <- grep("depth_m$|\\bdepth\\b", names(combined_CTD_data), ignore.case = TRUE
, value = TRUE)[1]
if (is.na(ctd_depth_col)) stop("No depth column found in CTD data.")
combined_CTD_data <- combined_CTD_data %>% filter(!is.na(datetime_UTC))

# add the following function in case a selection based on time UTC is needed (e.g. to spli
t a time series)
#combined_CTD_data <- combined_CTD_data %>%
  #filter(datetime_UTC >= as.POSIXct("2025-06-25 20:46:00", tz = "UTC") &
           #datetime_UTC <= as.POSIXct("2025-06-25 23:26:00", tz = "UTC"))

find_stable_intervals_ctd <- function(df, depth_col, depth_threshold = 1, min_duration_sec
s = 60,
                                      exclude_start_secs = 40, exclude_end_secs = 5, depth
_tol = 0.1) {
  df_filtered <- df %>% filter(!is.na(.data[[depth_col]]), .data[[depth_col]] > depth_thre
shold) %>% arrange(datetime_UTC)
  if (nrow(df_filtered) == 0) return(tibble())
  df_min <- df_filtered %>% mutate(datetime_min = floor_date(datetime_UTC, "minute")) %>%
group_by(datetime_min) %>% summarize(mean_depth = mean(.data[[depth_col]], na.rm = TRUE),
.groups = "drop") %>% arrange(datetime_min) %>%
    mutate(prev_min = lag(datetime_min), diff_min = as.numeric(difftime(datetime_min, prev
_min, units = "mins")), depth_diff = abs(mean_depth - lag(mean_depth)), new_group = is.na(
diff_min) | diff_min > 1 | depth_diff > depth_tol, stable_group_id = cumsum(new_group)) %>
%
    group_by(stable_group_id) %>%
    summarize(start_time = first(datetime_min) + seconds(exclude_start_secs),
              end_time = last(datetime_min) + minutes(1) - seconds(1 + exclude_end_secs),
              duration_s = as.numeric(difftime(last(datetime_min) + minutes(1), first(date
time_min), units = "secs")) - (exclude_start_secs + exclude_end_secs),
              mean_depth = mean(mean_depth, na.rm = TRUE),
              n_minutes = n(), .groups = "drop") %>%
    filter(duration_s >= min_duration_secs, start_time < end_time) %>%
    arrange(start_time) %>%
    mutate(stable_group = row_number())
  df_min
}

stable_intervals <- find_stable_intervals_ctd(combined_CTD_data, ctd_depth_col)
message("Identified ", nrow(stable_intervals), " stable depth intervals from CTD data.")

stable_depth_table <- stable_intervals %>% select(stable_group, start_time, end_time)
write_csv(stable_depth_table, file.path(ctd_folder, "stable_depth_intervals.csv"))

# ---- Join_sensor_files  ----
join_sensor_file <- function(sensor_df, stable_df, sensor_time_col = "datetime_utc") {
  if (!(sensor_time_col %in% names(sensor_df))) stop("sensor_time_col not found in sensor_
df")
  sensor_df[[sensor_time_col]] <- parse_datetime_vec(sensor_df[[sensor_time_col]], tz_in =
"UTC")
  sensor_dt <- as.data.table(sensor_df)
  stable_dt <- as.data.table(stable_df)
  stable_dt[, `:=`(start_time = as.POSIXct(start_time, tz = "UTC"), end_time = as.POSIXct(
end_time, tz = "UTC"))]
  sensor_dt[, `:=`(start_time_sensor = get(sensor_time_col), end_time_sensor = get(sensor_
time_col))]
```

```r
  setkey(sensor_dt, start_time_sensor, end_time_sensor)
  setkey(stable_dt, start_time, end_time)
  joined_dt <- foverlaps(sensor_dt, stable_dt, type = "within", nomatch = 0L)
  joined_dt[, c("start_time_sensor", "end_time_sensor") := NULL]
  as.data.frame(joined_dt)
}

stable_depth_table <- read_csv(file.path(ctd_folder, "stable_depth_intervals.csv"), show_c
ol_types = FALSE) %>% mutate(start_time = as.POSIXct(start_time, tz = "UTC"), end_time = a
s.POSIXct(end_time, tz = "UTC"))
for (sensor_name in names(sensor_folders)) {
  folder <- sensor_folders[[sensor_name]]
  if (!dir.exists(folder)) next
  sensor_files <- list.files(folder, pattern = "^combined.*\\.csv$", full.names = TRUE)
  message(sprintf("Processing %d files in folder '%s'...", length(sensor_files), sensor_na
me))
  all_joined_data <- list()
  for (file_path in sensor_files) {
    sensor_df <- read_csv(file_path, show_col_types = FALSE)
    names(sensor_df) <- clean_names(names(sensor_df))
    dt_info <- choose_datetime_info(names(sensor_df))
    if (is.na(dt_info$col)) { message("No datetime column in ", basename(file_path)); next
}
# Always prefer existing datetime_utc column when present (choose_datetime_info handles pr
eference)
    sensor_df$datetime_utc <- parse_datetime_vec(sensor_df[[dt_info$col]], tz_in = dt_info
$tz_in)
    joined_df <- tryCatch(join_sensor_file(sensor_df, stable_depth_table, sensor_time_col
= "datetime_utc"),
                          error = function(e) { message("Join error for ", basename(file_p
ath), ": ", e$message); NULL })
    if (!is.null(joined_df) && nrow(joined_df) > 0) all_joined_data[[file_path]] <- joined
_df
  }
  if (length(all_joined_data) > 0) {
    combined_joined_df <- bind_rows(all_joined_data)
    write_csv(combined_joined_df, file.path(folder, paste0("combined_stable_depth_", senso
r_name, ".csv")))
    message("Saved combined_stable_depth_", sensor_name)
  } else message("No stable-depth data for ", sensor_name)
}

# ---- Read combined_stable_depth files and plot with labels ----
combined_stable_files <- unlist(lapply(sensor_folders, function(f) if (!is.null(f) && dir.
exists(f)) list.files(f, pattern = "^combined_stable_depth_.*\\.csv$", full.names = TRUE)
else character(0)))
if (length(combined_stable_files) == 0) stop("No combined_stable_depth_*.csv files found."
)
rows_list <- list()
for (fp in combined_stable_files) {
  df <- read_csv(fp, show_col_types = FALSE)
  names(df) <- clean_names(names(df))
  dt_info <- choose_datetime_info(names(df))
  depth_col <- grep("_depth_m$|^depth_m$|\\bdepth\\b", names(df), ignore.case = TRUE, valu
e = TRUE)[1]
  stable_col <- grep("stable_group|stable.*group.*id", names(df), ignore.case = TRUE, valu
```

```
e = TRUE)[1]
  if (is.na(dt_info$col) || is.na(depth_col) || is.na(stable_col)) next
  sensor_name <- sub("^combined_stable_depth_([^.]+)\\.csv$", "\\1", basename(fp))
  df$datetime_utc <- parse_datetime_vec(df[[dt_info$col]], tz_in = dt_info$tz_in)
  df_sel <- df %>% transmute(datetime_UTC = as.POSIXct(datetime_utc, tz = "UTC"), depth_m
= as.numeric(.data[[depth_col]]), stable_group = .data[[stable_col]]) %>% filter(!is.na(da
tetime_UTC), !is.na(depth_m), !is.na(stable_group)) %>% mutate(sensor = sensor_name)
  if (nrow(df_sel) > 0) rows_list[[fp]] <- df_sel
}
if (length(rows_list) == 0) stop("No usable data in combined_stable_depth files.")
depth_data <- bind_rows(rows_list)
labels_df <- depth_data %>% group_by(sensor, stable_group) %>% summarize(median_time = as.
POSIXct(median(as.numeric(datetime_UTC)), origin = "1970-01-01", tz = "UTC"), median_depth
= median(depth_m, na.rm = TRUE), .groups = "drop")

depth_time <- ggplot(depth_data, aes(x = datetime_UTC, y = depth_m, color = sensor)) + geo
m_point(alpha = 0.7, size = 0.9) + scale_y_reverse() + labs(title = "Depth vs Time (Stable
Depth Groups Only)", x = "Time (UTC)", y = "Depth (m)", color = "Sensor") + theme_minimal(
) + geom_text_repel(data = labels_df, aes(x = median_time, y = median_depth, label = stabl
e_group, color = sensor), size = 3, min.segment.length = 0, box.padding = 0.3, segment.col
or = "grey50", show.legend = FALSE)
print(depth_time)

plot_depth_vs_temperature_from_combined <- function(combined_files = NULL, sensor_folders
= NULL) {
  if (is.null(combined_files)) {
    if (!is.null(sensor_folders)) {
      combined_files <- unlist(lapply(sensor_folders, function(f) {
        if (is.null(f) || !dir.exists(f)) return(character(0))
        list.files(f, pattern = "^combined_stable_depth_.*\\.csv$", full.names = TRUE)
      }))
    } else {
      combined_files <- list.files(pattern = "^combined_stable_depth_.*\\.csv$", full.name
s = TRUE)
    }
  }
  if (length(combined_files) == 0) stop("No combined_stable_depth_*.csv files found.")

  rows <- list()
  for (fp in combined_files) {
    df <- readr::read_csv(fp, show_col_types = FALSE)
    names(df) <- clean_names(names(df))

    temp_col <- grep("temp|temperature", names(df), ignore.case = TRUE, value = TRUE)[1]
    depth_col <- grep("_depth_m$|^depth_m$|\\bdepth\\b", names(df), ignore.case = TRUE, va
lue = TRUE)[1]
    stable_col <- grep("stable_group|stable.*group.*id|stable_group_id", names(df), ignore
.case = TRUE, value = TRUE)[1]
    if (is.na(temp_col) || is.null(temp_col)) {
      message("Skipping ", basename(fp), " (no temperature column detected).")
      next
    }
    if (is.na(depth_col) || is.null(depth_col)) {
      message("Skipping ", basename(fp), " (no depth column detected).")
      next
    }
```

```r
    if (is.na(stable_col) || is.null(stable_col)) {
      message("Skipping ", basename(fp), " (no stable_group column detected).")
      next
    }

    sensor_name <- sub("^combined_stable_depth_([^.]+)\\.csv$", "\\1", basename(fp))
    if (sensor_name == basename(fp)) sensor_name <- get_prefix_from_col(depth_col)

    df_sel <- df %>%
      transmute(
        sensor = sensor_name,
        stable_group = as.character(.data[[stable_col]]),
        temp = as.numeric(.data[[temp_col]]),
        depth = as.numeric(.data[[depth_col]])
      ) %>%
      filter(!is.na(temp) & !is.na(depth) & !is.na(stable_group))
    if (nrow(df_sel) > 0) rows[[fp]] <- df_sel
  }

  if (length(rows) == 0) stop("No usable data across combined files.")
  dat <- dplyr::bind_rows(rows)

  stats <- dat %>%
    dplyr::group_by(sensor, stable_group) %>%
    dplyr::summarize(
      n = sum(!is.na(temp)),
      mean_temp = mean(temp, na.rm = TRUE),
      sd_temp = sd(temp, na.rm = TRUE),
      mean_depth = mean(depth, na.rm = TRUE),
      .groups = "drop"
    ) %>%
    dplyr::mutate(
      se = sd_temp / sqrt(pmax(1, n)),
      t_crit = ifelse(n > 1, qt(0.975, df = n - 1), NA_real_),
      ci_lower = ifelse(!is.na(t_crit), mean_temp - t_crit * se, mean_temp),
      ci_upper = ifelse(!is.na(t_crit), mean_temp + t_crit * se, mean_temp)
    )

  low_n <- stats %>% dplyr::filter(n < 2)
  if (nrow(low_n) > 0) message("Note: ", nrow(low_n), " group(s) have n < 2; CI not availa
ble for those groups.")

  p <- ggplot2::ggplot(stats, ggplot2::aes(x = mean_temp, y = mean_depth, color = sensor))
+
    ggplot2::geom_point(size = 2) +
    ggplot2::geom_errorbarh(ggplot2::aes(xmin = ci_lower, xmax = ci_upper), height = 0.1)
+
    ggplot2::scale_y_reverse() +
    ggplot2::labs(title = "Depth vs Average Temperature with 95% CI", x = "Mean Temperatur
e", y = "Mean Depth (m)", color = "Sensor") +
    ggplot2::theme_minimal() +
    ggrepel::geom_text_repel(ggplot2::aes(label = stable_group), size = 3, show.legend = F
ALSE)
  return(p)
}
```

```r
plot_depth_vs_salinity_from_combined <- function(combined_files = NULL, sensor_folders = N
ULL) {
  if (is.null(combined_files)) {
    if (!is.null(sensor_folders)) {
      combined_files <- unlist(lapply(sensor_folders, function(f) {
        if (is.null(f) || !dir.exists(f)) return(character(0))
        list.files(f, pattern = "^combined_stable_depth_.*\\.csv$", full.names = TRUE)
      }))
    } else {
      combined_files <- list.files(pattern = "^combined_stable_depth_.*\\.csv$", full.name
s = TRUE)
    }
  }
  if (length(combined_files) == 0) stop("No combined_stable_depth_*.csv files found.")

  rows <- list()
  for (fp in combined_files) {
    df <- readr::read_csv(fp, show_col_types = FALSE)
    names(df) <- clean_names(names(df))

    sal_col <- grep("sal|salinity", names(df), ignore.case = TRUE, value = TRUE)[1]
    depth_col <- grep("_depth_m$|^depth_m$|\\bdepth\\b", names(df), ignore.case = TRUE, va
lue = TRUE)[1]
    stable_col <- grep("stable_group|stable.*group.*id|stable_group_id", names(df), ignore
.case = TRUE, value = TRUE)[1]
    if (is.na(sal_col) || is.null(sal_col)) {
      message("Skipping ", basename(fp), " (no salinity column detected).")
      next
    }
    if (is.na(depth_col) || is.null(depth_col)) {
      message("Skipping ", basename(fp), " (no depth column detected).")
      next
    }
    if (is.na(stable_col) || is.null(stable_col)) {
      message("Skipping ", basename(fp), " (no stable_group column detected).")
      next
    }

    sensor_name <- sub("^combined_stable_depth_([^.]+)\\.csv$", "\\1", basename(fp))
    if (sensor_name == basename(fp)) sensor_name <- get_prefix_from_col(depth_col)
    df_sel <- df %>%
      transmute(
        sensor = sensor_name,
        stable_group = as.character(.data[[stable_col]]),
        sal = as.numeric(.data[[sal_col]]),
        depth = as.numeric(.data[[depth_col]])
      ) %>%
      filter(!is.na(sal) & !is.na(depth) & !is.na(stable_group))
   if (nrow(df_sel) > 0) rows[[fp]] <- df_sel
  }
  if (length(rows) == 0) stop("No usable data across combined files.")
  dat <- dplyr::bind_rows(rows)
  stats <- dat %>%
    dplyr::group_by(sensor, stable_group) %>%
    dplyr::summarize(
      n = sum(!is.na(sal)),
```

```
      mean_sal = mean(sal, na.rm = TRUE),
      sd_sal = sd(sal, na.rm = TRUE),
      mean_depth = mean(depth, na.rm = TRUE),
      .groups = "drop"
    ) %>%
    dplyr::mutate(
      se = sd_sal / sqrt(pmax(1, n)),
      t_crit = ifelse(n > 1, qt(0.975, df = n - 1), NA_real_),
      ci_lower = ifelse(!is.na(t_crit), mean_sal - t_crit * se, mean_sal),
      ci_upper = ifelse(!is.na(t_crit), mean_sal + t_crit * se, mean_sal)
    )
  low_n <- stats %>% dplyr::filter(n < 2)
  if (nrow(low_n) > 0) message("Note: ", nrow(low_n), " group(s) have n < 2; CI not availa
ble for those groups.")

  p <- ggplot2::ggplot(stats, ggplot2::aes(x = mean_sal, y = mean_depth, color = sensor))
+
    ggplot2::geom_point(size = 2) +
    ggplot2::geom_errorbarh(ggplot2::aes(xmin = ci_lower, xmax = ci_upper), height = 0.1)
+
    ggplot2::scale_y_reverse() +
    ggplot2::labs(title = "Depth vs Average Salinity with 95% CI", x = "Mean Salinity", y
= "Mean Depth (m)", color = "Sensor") +
    ggplot2::theme_minimal() +
    ggrepel::geom_text_repel(ggplot2::aes(label = stable_group), size = 3, show.legend = F
ALSE)
  return(p)
}

 p_temp <- plot_depth_vs_temperature_from_combined(sensor_folders = sensor_folders)
 print(p_temp)
p_sal  <- plot_depth_vs_salinity_from_combined(sensor_folders = sensor_folders)
print(p_sal)


# ---- Summary stats by stable group (temperature/salinity/depth/pressure) ----
process_sensor_file <- function(file_path) {
  df <- read_csv(file_path, show_col_types = FALSE)
  names(df) <- clean_names(names(df))
  dt_info <- choose_datetime_info(names(df))
  stable_col <- grep("stable_group|stable.*group.*id", names(df), ignore.case = TRUE, valu
e = TRUE)[1]
  if (is.na(dt_info$col) || is.na(stable_col)) return(NULL)
  df$datetime_utc <- parse_datetime_vec(df[[dt_info$col]], tz_in = dt_info$tz_in)
  depth_cols <- names(df)[grepl("depth", names(df))]
  pressure_cols <- names(df)[grepl("press", names(df))]
  temperature_cols <- names(df)[grepl("temp", names(df))]
  salinity_cols <- names(df)[grepl("sal", names(df))]
  num_vars <- c(depth_cols, pressure_cols, temperature_cols, salinity_cols)
  if (length(num_vars) == 0) return(NULL)
  sensor_name <- sub("^combined_stable_depth_([^.]+)\\.csv$", "\\1", basename(file_path))
  long_df <- df %>% pivot_longer(cols = all_of(num_vars), names_to = "orig_variable", valu
es_to = "value") %>% filter(!is.na(value)) %>% mutate(variable = sapply(orig_variable, sta
ndardize_var), sensor = sensor_name) %>% filter(!is.na(variable))
  stats <- long_df %>% group_by(sensor, stable_group = .data[[stable_col]], variable) %>%
summarize(n = sum(!is.na(value)), mean = mean(value, na.rm = TRUE), sd = sd(value, na.rm =
```

```
TRUE), se = sd / sqrt(pmax(n, 1)), t_crit = ifelse(n > 1, qt(0.975, df = n - 1), NA_real_)
, ci_lower = ifelse(!is.na(t_crit), mean - t_crit * se, mean), ci_upper = ifelse(!is.na(t_
crit), mean + t_crit * se, mean), min = min(value, na.rm = TRUE), max = max(value, na.rm =
TRUE), datetime_min = min(datetime_utc, na.rm = TRUE), datetime_max = max(datetime_utc, na
.rm = TRUE), .groups = "drop")
  stats
}

combined_files <- combined_stable_files
stats_list <- map(combined_files, process_sensor_file)
stats_list <- stats_list[!sapply(stats_list, is.null)]
if (length(stats_list) == 0) stop("No valid data processed.")
all_stats <- bind_rows(stats_list)
write_csv(all_stats, file.path(base_folder, "stable_depth_sensor_statistics.csv"))
message("Saved stable_depth_sensor_statistics.csv")
```

12 Offsets computation and exploratory statistics

This section of the script calculates, across stable depth intervals, average offsets for each sensor/variable combination against the chosen reference CTD, along with a range of exploratory statistics (i.e. avg_mean_diff, median_mean_diff, sd_mean_diff, count, se, ci_lower, ci_upper) and produce the `stable_depth_sensor_statistics.csv`. Plots are as well generated to visually highlights how these deviations relate to accuracies declared by the manufacturer, helping evaluate sensor performance and calibration quality.

The second part of the script identifies “depth bins”, namely different depth intervals (defined according to CTD depths grouping stable depth groups) and calculate the previously described statistics for each of them, producing the `sensor_diff_summary_by_fixed_depth_bins.csv` file and related plots.

```
# === 8. Offsets calculation ===
# ---- Differences vs CTD and summaries ----
# set accuracies based on the characteristics of the tested sensors declared by the manufa
cturer
sensor_accuracies <- tibble::tribble(
  ~sensor, ~temp_acc, ~depth_acc,
  "CTD", 0.001, 1.02,
  "Wisens", 0.005, 0.45,
  "Moana", 0.05, 1.0,
  "Starmon_b", 0.025, 0.6,
  "Starmon_a", 0.025, 0.6,
  "Tegaru_a", 0.2, 2.0,
  "Tegaru_b", 0.2, 5.0
)

stable_stats_all <- read_csv(file.path(base_folder, "stable_depth_sensor_statistics.csv"),
show_col_types = FALSE)
ctd_stats <- stable_stats_all %>% filter(sensor == "CTD") %>% select(stable_group, variabl
e, ctd_mean = mean)
stats_diff <- stable_stats_all %>% filter(sensor != "CTD") %>% left_join(ctd_stats, by = c
("variable", "stable_group")) %>% mutate(mean_diff = mean - ctd_mean) %>% select(stable_gr
oup, sensor, variable, mean, ctd_mean, mean_diff, n, sd)
write_csv(stats_diff, file.path(base_folder, "stats_diff.csv"))
```

```r
message("stats_diff.csv")

summary_stats <- stats_diff %>% group_by(sensor, variable) %>% summarise(avg_mean_diff = m
ean(mean_diff, na.rm = TRUE), median_mean_diff = median(mean_diff, na.rm = TRUE), sd_mean_
diff = sd(mean_diff, na.rm = TRUE), count = n(), se = sd_mean_diff / sqrt(count), ci_lower
= avg_mean_diff - qt(0.975, df = pmax(1, count - 1)) * se, ci_upper = avg_mean_diff + qt(0
.975, df = pmax(1, count - 1)) * se, .groups = "drop")
write_csv(summary_stats, file.path(base_folder, "mean_diff_summary_stats.csv"))
message("Saved mean_diff_summary_stats.csv")

p_mean_diff <- ggplot(summary_stats, aes(x = variable, y = avg_mean_diff, color = sensor))
+ geom_point(position = position_dodge(width = 0.7), size = 3) + geom_errorbar(aes(ymin =
ci_lower, ymax = ci_upper), position = position_dodge(width = 0.7), width = 0.2) + labs(ti
tle = "Average Mean Difference with 95% CI", y = "Average Mean Difference", x = "Variable"
, color = "Sensor") + theme_minimal() + theme(axis.text.x = element_text(angle = 45, hjust
= 1))
print(p_mean_diff)

# ---- Compare summary_stats with sensor accuracies & plots ----
# Join accuracy info (safe join; keep unmatched sensors too)
summary_stats <- summary_stats %>%
  left_join(sensor_accuracies %>% rename_all(clean_names), by = c("sensor" = "sensor"))

# Helper to draw sensor-specific +/- accuracy lines centered on each sensor's x position
plot_with_accuracy_lines <- function(df_summary, variable_name, acc_col = NULL, title = NU
LL) {
  df <- df_summary %>%
    filter(variable == variable_name) %>%
    arrange(sensor) %>%
    mutate(sensor = as.character(sensor))
  if (nrow(df) == 0) {
    message("No rows for variable: ", variable_name); return(NULL)
  }
  df$sensor_f <- factor(df$sensor, levels = unique(df$sensor))
  df$xpos <- as.numeric(df$sensor_f)

  # accuracy lines data (only for sensors with non-missing accuracy)
  if (!is.null(acc_col) && acc_col %in% names(df)) {
    acc_df <- df %>% filter(!is.na(.data[[acc_col]])) %>%
      transmute(sensor, xpos,
                acc_pos = .data[[acc_col]],
                acc_neg = -.data[[acc_col]],
                xmin = xpos - 0.3, xmax = xpos + 0.3)
  } else {
    acc_df <- tibble()
  }

  p <- ggplot(df, aes(x = sensor_f, y = avg_mean_diff, color = sensor)) +
    geom_point(position = position_dodge(width = 0.7), size = 3) +
    geom_errorbar(aes(ymin = ci_lower, ymax = ci_upper), position = position_dodge(width =
0.7), width = 0.2) +
    labs(
      title = ifelse(is.null(title), paste0("Average Mean Difference for ", variable_name)
, title),
      y = "Average Mean Difference",
      x = "Sensor",
      color = "Sensor"
```

```r
    ) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))

  # add per-sensor accuracy horizontal short segments (+/-) using linewidth (replacement f
or deprecated size for lines)
  if (nrow(acc_df) > 0) {
    p <- p +
      geom_segment(data = acc_df,
                   aes(x = xmin, xend = xmax, y = acc_pos, yend = acc_pos),
                   inherit.aes = FALSE, linetype = "dashed", alpha = 0.7, linewidth = 0.7,
color = "black") +
      geom_segment(data = acc_df,
                   aes(x = xmin, xend = xmax, y = acc_neg, yend = acc_neg),
                   inherit.aes = FALSE, linetype = "dashed", alpha = 0.7, linewidth = 0.7,
color = "black")
  }
  p
}

# Depth plot (uses depth_acc)
p_depth_diff <- plot_with_accuracy_lines(summary_stats, "depth", acc_col = "depth_acc",
                                         title = "Average Mean Difference for Depth with 9
5% CI and Sensor Accuracy")
if (!is.null(p_depth_diff)) print(p_depth_diff)

# Temperature plot (uses temp_acc)
p_temp_diff <- plot_with_accuracy_lines(summary_stats, "temperature", acc_col = "temp_acc"
,
                                        title = "Average Mean Difference for Temperature w
ith 95% CI and Sensor Accuracy")
if (!is.null(p_temp_diff)) print(p_temp_diff)

# Salinity plot (no formal accuracy shown here)
p_sal_diff <- plot_with_accuracy_lines(summary_stats, "salinity", acc_col = NULL,
                                       title = "Average Mean Difference for Salinity with
95% CI")
if (!is.null(p_sal_diff)) print(p_sal_diff)

# Add a small summary table combining avg offset and accuracy for quick inspection
summary_vs_acc <- summary_stats %>%
  filter(variable %in% c("depth", "temperature", "salinity")) %>%
  select(sensor, variable, avg_mean_diff, sd_mean_diff, ci_lower, ci_upper, temp_a
cc, depth_acc) %>%
  arrange(sensor, variable)

# save a CSV for records
readr::write_csv(summary_vs_acc, file.path(base_folder, "summary_vs_sensor_accurac
y.csv"))
message("Saved summary_vs_sensor_accuracy.csv")

# ---- Fixed depth bins calculations ----
ctd_depth_ranges <- stable_stats_all %>% filter(sensor == "CTD", variable == "depth") %>%
arrange(stable_group) %>% select(stable_group, min_depth = min, max_depth = max) %>% mutat
e(mid_depth = (min_depth + max_depth) / 2)
```

```r
bin_width <- 10
ctd_min_depth <- floor(min(ctd_depth_ranges$min_depth, na.rm = TRUE))
ctd_max_depth <- ceiling(max(ctd_depth_ranges$max_depth, na.rm = TRUE))
break_points <- seq(ctd_min_depth, ctd_max_depth + bin_width, by = bin_width)
bin_labels <- paste0(head(break_points, -1), " - ", tail(break_points, -1), " m")
ctd_depth_ranges <- ctd_depth_ranges %>% mutate(bin_label = cut(mid_depth, breaks = break_
points, labels = bin_labels, include.lowest = TRUE, right = FALSE))
stats_diff_bin <- stats_diff %>% left_join(ctd_depth_ranges %>% select(stable_group, bin_l
abel, mid_depth), by = "stable_group")
summary_stats_bin <- stats_diff_bin %>% filter(!is.na(bin_label)) %>% group_by(sensor, var
iable, bin_label) %>% summarise(avg_mean_diff = mean(mean_diff, na.rm = TRUE), median_mean
_diff = median(mean_diff, na.rm = TRUE), sd_mean_diff = sd(mean_diff, na.rm = TRUE), count
= n(), se = sd_mean_diff / sqrt(pmax(1, count)), ci_lower = avg_mean_diff - qt(0.975, df =
pmax(1, count - 1)) * se, ci_upper = avg_mean_diff + qt(0.975, df = pmax(1, count - 1)) *
se, .groups = "drop")
write_csv(summary_stats_bin, file.path(base_folder, "sensor_diff_summary_by_fixed_depth_bi
ns.csv"))
message("Saved sensor_diff_summary_by_fixed_depth_bins.csv")

avg_mean_depths_diff_bin <- summary_stats_bin %>% filter(variable == "depth")
avg_mean_temp_diff_bin <- summary_stats_bin %>% filter(variable == "temperature")
avg_mean_sal_diff_bin <- summary_stats_bin %>% filter(variable == "salinity")

p_depth_bins <- ggplot(avg_mean_depths_diff_bin, aes(x = bin_label, y = avg_mean_diff, col
or = sensor)) + geom_point(position = position_dodge(width = 0.7), size = 3) + geom_errorb
ar(aes(ymin = ci_lower, ymax = ci_upper), position = position_dodge(width = 0.7), width =
0.2) + labs(title = "Average Mean Difference (Depth) with 95% CI", y = "Average Mean Diffe
rence", x = "Depth bin", color = "Sensor") + theme_minimal() + theme(axis.text.x = element
_text(angle = 45, hjust = 1))
print(p_depth_bins)

p_temp_bins <- ggplot(avg_mean_temp_diff_bin, aes(x = bin_label, y = avg_mean_diff, color
= sensor)) + geom_point(position = position_dodge(width = 0.7), size = 3) + geom_errorbar(
aes(ymin = ci_lower, ymax = ci_upper), position = position_dodge(width = 0.7), width = 0.2
) + labs(title = "Average Mean Difference (Temperature) with 95% CI", y = "Average Mean Di
fference", x = "Depth bin", color = "Sensor") + theme_minimal() + theme(axis.text.x = elem
ent_text(angle = 45, hjust = 1))
print(p_temp_bins)

p_sal_bins <- ggplot(avg_mean_sal_diff_bin, aes(x = bin_label, y = avg_mean_diff, color =
sensor)) + geom_point(position = position_dodge(width = 0.7), size = 3) + geom_errorbar(ae
s(ymin = ci_lower, ymax = ci_upper), position = position_dodge(width = 0.7), width = 0.2)
+ labs(title = "Average Mean Difference (Salinity) with 95% CI", y = "Average Mean Differe
nce", x = "Depth bin", color = "Sensor") + theme_minimal() + theme(axis.text.x = element_t
ext(angle = 45, hjust = 1))
print(p_sal_bins)
```

13 Statistical analyses

This section of the code is dedicated to statistical analyses to robustly test whether measurement offsets of each sensor (e.g. on temperature or salinity) are dependent on depth or temperature. Tests are then performed to verify any differences between downcast and upcast. Related plots help to interpret possible effects.

```
###### STATISTICAL TESTS
# The following script assumes the 'tidyverse' package (for data manipulation with
dplyr, ggplot2, tidyr, purrr) and 'broom' package (for tidying model outputs) are
installed and loaded.
# The following script assumes two data frames, 'stats_diff' and 'stable_stats_all
', are available in the environment,
# if not, use the following lines to read from CSV files
#stats_diff <- read_csv(file.path(base_folder, "stats_diff.csv"), show_col_types =
FALSE)
#stable_stats_all <- read_csv(file.path(base_folder, "stable_depth_sensor_statisti
cs.csv"), show_col_types = FALSE)

# --- Data Preparation: Extracting Reference CTD Data ---
# Get the average depth and temperature recorded by the reference CTD for each 'st
able_group' and add cast_direction.
ctd_depth_temp <- stable_stats_all%>%
  filter(sensor == "CTD", variable %in% c("depth", "temperature")) %>%
  select(stable_group, variable, ctd_value = mean) %>%
  pivot_wider(names_from = variable, values_from = ctd_value) %>%
  rename(depth_CTD = depth, temperature_CTD = temperature)%>%
arrange(stable_group) %>%
  mutate(
     depth_change = depth_CTD - lag(depth_CTD),
    cast_direction = ifelse(depth_change > 0, "downcast", "upcast"),
    cast_direction = ifelse(is.na(depth_change), "downcast", cast_direction)
  )

# --- Data Preparation: Merging and Filtering ---
# Combine the sensor difference data with the reference CTD data for analysis.
regs_data <- stats_diff %>%
  filter(sensor != "CTD") %>%
  left_join(ctd_depth_temp, by = "stable_group") %>%
  filter(!is.na(cast_direction), !is.na(depth_CTD), !is.na(temperature_CTD), !is.n
a(mean_diff))
```

Statistical analyses:

1. **Shapiro_P_Value (X & Y):**
    - Tests the null hypothesis that the data series are normally distributed.
    - **Interpretation:** If $p < 0.05$, the data is likely **not** normally distributed. In this case, rely on the **Spearman_R** correlation rather than Pearson.
2. **Spearman_R:**
    - Assesses how well the relationship between two variables can be described using a **monotonic function**. It is non-parametric and robust to outliers and non-normal distributions.

3. **Pearson_R:**
    - Assesses the strength and direction of the **linear relationship** between two variables. It assumes normality and is sensitive to outliers.
4. **Logic Gate (N_Observations ≥ 3):**
    - The script returns NA if the sample size is too small for calculation. However, results with **N_Observations < 10** should be treated as purely indicative and not statistically robust.

How to Interpret the Output:

Column	Description	Interpretation Guide
sensor	Sensor ID	The specific device being analyzed.
N_Observations	Valid data points	Samples with N<10 are indicative but not statistically robust.
Shapiro_P_Value	Normality Test	If p>0.05, use Pearson_R. If p<0.05, use Spearman_R.
Spearman_R	Rank Correlation	Use if the relationship is non-linear or data is not normal. Varies from -1 to +1.
P_value_Spearman	Correlation Significance	If p<0.05, the monotonic relationship is statistically significant.
Pearson_R	Linear Correlation	Values near 1 or -1 indicate a strong linear relationship.
P_value_Pearson	Correlation Significance	If p<0.05, the linear relationship is statistically significant.
Slope	Regression Slope	Shows how much the offset changes for every unit increase in the predictor.
R_Squared	Goodness of Fit	The proportion of offset variability explained by the predictor (0 to 1).

```
##### 1. NORMALITY, CORRELATION, AND REGRESSION ######
# This script performs a comprehensive analysis of sensor offsets.
# 1. Shapiro-Wilk Test: Verifies the assumption of normality.
# 2. Spearman Correlation: A non-parametric measure for monotonic relationships (r
obust to outliers).
# 3. Pearson Correlation: A parametric measure for linear relationships.
# 4. Linear Regression: Quantifies the systematic drift (Slope and R-Squared).
analyze_sensor_correlation <- function(df, y_var, x_var, target_variable) {

  # Filter data for the target variable and remove missing values
  data_filtered <- df %>%
    filter(variable == target_variable) %>%
    filter(!is.na(.data[[y_var]]), !is.na(.data[[x_var]]))

  # Group by sensor and calculate diagnostics and correlation metrics
  correlation_results <- data_filtered %>%
    group_by(sensor, variable) %>%
    summarize(
      N_Observations = n(),

      # --- 1. Normality Check (Shapiro-Wilk) ---
      # If p < 0.05, the data is likely NOT normal.
      Shapiro_P_Value_Y = if (n() >= 3) shapiro.test(get(y_var))$p.value else NA_r
eal_,
      Shapiro_P_Value_X = if (n() >= 3) shapiro.test(get(x_var))$p.value else NA_r
```

```
eal_,

      # --- 2. Spearman Correlation (Non-Parametric) ---
      # Robust to outliers and non-normal distributions.
      Spearman_R = if (n() >= 3) cor(get(x_var), get(y_var), method = "spearman")
else NA_real_,
      P_value_Spearman = if (n() >= 3) cor.test(get(x_var), get(y_var), method = "
spearman")$p.value else NA_real_,

      # --- 3. Pearson Correlation (Parametric) ---
      # Best for linear relationships in normally distributed data.
      Pearson_R = if (n() >= 3) cor(get(x_var), get(y_var), method = "pearson") el
se NA_real_,
      P_value_Pearson = if (n() >= 3) cor.test(get(x_var), get(y_var), method = "p
earson")$p.value else NA_real_,

      .groups = "drop"
    )

  # --- 4. Regression Analysis (Linear Model) ---
  regression_results <- data_filtered %>%
    group_by(sensor) %>%
    do(
      if (nrow(.) >= 3) {
        model <- lm(reformulate(x_var, response = y_var), data = .)
        tidy_model <- broom::tidy(model)
        glance_model <- broom::glance(model)

        tibble(
          Slope = tidy_model$estimate[tidy_model$term == x_var],
          R_Squared = glance_model$r.squared
        )
      } else {
        tibble(Slope = NA_real_, R_Squared = NA_real_)
      }
    )

  # Join all metrics into a single table
  final_results <- correlation_results %>%
    left_join(regression_results, by = "sensor")

  return(final_results)
}

# --- Execution ---
# Set A: Predictor = Depth (CTD)
results_depth <- regs_data %>%
  group_split(variable) %>%
  map_dfr(~analyze_sensor_correlation(.x, "mean_diff", "depth_CTD", unique(.x$vari
able)))
```

```r
# Set B: Predictor = Temperature (CTD)
results_temp <- regs_data %>%
  group_split(variable) %>%
  map_dfr(~analyze_sensor_correlation(.x, "mean_diff", "temperature_CTD", unique(.
x$variable)))

print("--- Analysis Results: Predictor Depth_CTD ---")

print(results_depth, n = Inf,width = Inf)

print("--- Analysis Results: Predictor Temperature_CTD ---")

print(results_temp, n = Inf,width = Inf)

# Plotting Function for Visualization, Generates a scatter plot with regression li
ne for easy interpretation

plot_sensor_regression <- function(df, var_name, x_var, y_var = "mean_diff") {
  # Filter data similarly to the analysis function
  df_sub <- df %>%
    filter(variable == var_name) %>%
    filter(!is.na(.data[[y_var]]), !is.na(.data[[x_var]]))

  ggplot(df_sub, aes_string(x = x_var, y = y_var, color = "sensor")) +
    geom_point(alpha = 0.6) + # Add transparent scatter points
    geom_smooth(method = "lm", se = TRUE, linewidth = 1) + # Add linear model fit
line with confidence interval (se=TRUE)
    labs(
      title = paste("Offset of", var_name, "vs", tools::toTitleCase(gsub("_", " ",
x_var))),
      x = tools::toTitleCase(gsub("_", " ", x_var)),
      y = paste("Mean", var_name, "Offset"),
      color = "Sensor"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom")
}

print(plot_sensor_regression(df = regs_data, var_name = "depth", x_var = "depth_CT
D"))

print(plot_sensor_regression(df = regs_data, var_name = "temperature", x_var = "de
pth_CTD"))

print(plot_sensor_regression(df = regs_data, var_name = "salinity", x_var = "depth
_CTD"))

print(plot_sensor_regression(df = regs_data, var_name = "temperature", x_var = "te
mperature_CTD"))

print(plot_sensor_regression(df = regs_data, var_name = "salinity", x_var = "tempe
rature_CTD"))
```

```r
##### 2.Differences downcast/upcast
#generate plots

plot_cast_differences <- function(df, var_name) {
  df_sub <- df %>% filter(variable == var_name)

  ggplot(df_sub, aes(x = depth_CTD, y = mean_diff, color = cast_direction)) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm", se = TRUE) +
    facet_wrap(~ sensor) +
    labs(
      title = paste(var_name, "Offset by Cast Direction"),
      x = "Depth (m)",
      y = paste("Mean", var_name, "Offset"),
      color = "Direction"
    ) +
    theme_minimal()
}

print(plot_cast_differences(regs_data, "depth"))

print(plot_cast_differences(regs_data, "temperature"))

print(plot_cast_differences(regs_data, "salinity"))

#Perform T-test between upcast and downcast series
test_cast_significance <- function(df, target_var) {

  df_sub <- df %>%
    filter(variable == target_var, cast_direction %in% c("downcast", "upcast")) %>
%
    filter(!is.na(mean_diff))

  if(nrow(df_sub) == 0) return(NULL)

  results <- df_sub %>%
    group_by(sensor) %>%
    summarize(
      test_results_df = list(
        if (n_distinct(cur_data()$cast_direction) == 2 && n() >= 3) {
          tidy(t.test(mean_diff ~ cast_direction, data = cur_data()))
        } else {
          tibble(estimate = NA_real_, p.value = NA_real_, method = "Insufficient D
ata",
                 estimate1 = NA_real_, estimate2 = NA_real_, statistic = NA_real_,
                 parameter = NA_real_, conf.low = NA_real_, conf.high = NA_real_,
alternative = NA_real_)
        }
      ),
      .groups = "drop"
    ) %>%
```

```r
    unnest(test_results_df) %>%
    select(sensor, p.value, estimate1, estimate2, method)

  return(results)
}

sig_depth <- test_cast_significance(regs_data, "depth")

print("--- Significance of Depth Differences (Upcast vs Downcast) ---")

print(sig_depth)

sig_temp <- test_cast_significance(regs_data, "temperature")
print("--- Significance of Temperature Differences (Upcast vs Downcast) ---")

print(sig_temp)

sig_sal <- test_cast_significance(regs_data, "salinity")
print("--- Significance of Salinity Differences (Upcast vs Downcast) ---")

print(sig_sal)
```

Plots and Summary statistics for depth bins and cast direction

```r
# Bin definition
bin_width <- 10
ctd_min_depth <- floor(min(ctd_depth_temp$depth_CTD, na.rm = TRUE))
ctd_max_depth <- ceiling(max(ctd_depth_temp$depth_CTD, na.rm = TRUE))

# create break point and bin labels
break_points <- seq(ctd_min_depth, ctd_max_depth + bin_width, by = bin_width)
bin_labels <- paste0(head(break_points, -1), " - ", tail(break_points, -1), " m")

# assign bin and calculate stats
ctd_depth_ranges <- ctd_depth_temp  %>%
  mutate(
    bin_label = cut(depth_CTD,
                    breaks = break_points,
                    labels = bin_labels,
                    include.lowest = TRUE,
                    right = FALSE)
  )

stats_diff_bin_dir <- stats_diff %>%
  left_join(ctd_depth_ranges %>%
              select(stable_group, bin_label, depth_CTD, cast_direction), # Includ
i cast_direction
            by = "stable_group")

summary_stats_bin_dir <- stats_diff_bin_dir %>%
  filter(!is.na(bin_label), !is.na(cast_direction)) %>%
  group_by(sensor, variable, bin_label, cast_direction) %>%
```

```r
  summarise(
    avg_mean_diff = mean(mean_diff, na.rm = TRUE),
    median_mean_diff = median(mean_diff, na.rm = TRUE),
    sd_mean_diff = sd(mean_diff, na.rm = TRUE),
    count = n(),
    se = sd_mean_diff / sqrt(pmax(1, count)),
    ci_lower = avg_mean_diff - qt(0.975, df = pmax(1, count - 1)) * se,
    ci_upper = avg_mean_diff + qt(0.975, df = pmax(1, count - 1)) * se,
    .groups = "drop"
  )

write_csv(summary_stats_bin_dir, file.path(base_folder, "sensor_diff_summary_by_fi
xed_depth_bins_direction.csv"))
message("Saved sensor_diff_summary_by_fixed_depth_bins_direction.csv")

# generate plots
plot_binned_stats_combined <- function(data, target_variable, title_text) {
  data %>%
    filter(variable == target_variable) %>%
    ggplot(aes(x = bin_label, y = avg_mean_diff, color = sensor, shape = cast_dire
ction)) +
    geom_point(position = position_dodge(width = 0.7), size = 3) +
    geom_errorbar(aes(ymin = ci_lower, ymax = ci_upper), position = position_dodge
(width = 0.7), width = 0.2) +
    labs(
      title = paste(title_text, "(Upcast vs Downcast)"),
      y = "Average Mean Difference",
      x = "Depth bin (m)",
      color = "Sensor",
      shape = "Direction"
    ) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          legend.position = "bottom")
}

p_depth_combined <- plot_binned_stats_combined(summary_stats_bin_dir, "depth", "De
pth Offset (95% CI)")
print(p_depth_combined)

p_temp_combined <- plot_binned_stats_combined(summary_stats_bin_dir, "temperature"
, "Temperature Offset (95% CI)")
print(p_temp_combined)

p_sal_combined <- plot_binned_stats_combined(summary_stats_bin_dir, "salinity", "S
alinity Offset (95% CI)")
print(p_sal_combined)
```