Population Pharmacokinetic Analysis Data (ADPPK) Programming in `{admiral}` and the Pharmaverse

Jeffrey Dickinson, Navitas Data Sciences

Table of contents

0.1 Abstract

Population Pharmacokinetic modeling is an important tool for drug development. The CDISC ADaM Population PK Implementation Guide was released on October 6, 2023. Population PK models generally make use of nonlinear mixed effects models that require numeric variables. The data used in the models will include both dosing and concentration records, relative time variables, and numeric covariate variables. {admiral} is an open-source R package for creating CDISC ADaM data. It can be used effectively to create Population PK analysis data (ADPPK). Additional tools from other Pharmaverse packages such as {metacore}, {metatools} and {xportr} can be used to simplify the workflow. I will discuss some of the challenges of Population Pharmacokinetic analysis data programming and show some of the solutions developed in {admiral} and the Pharmaverse.

0.2 Introduction

Pharmacokinetics considers the effect of the body on a drug. Typically samples are drawn at set time intervals after dose administration as illustrated in the schematic above. The resulting concentration profiles can be analyzed. With population pharmacokinetic models, variations within and between populations can be assessed. The CDISC ADaM Population PK Analysis Data Implementation Guide was released on October 6, 2023. Population PK models generally make use of nonlinear mixed effects models that require numeric variables. The data used in the models will include both dosing and concentration records, relative time variables, and numeric covariate variables. A DV or dependent variable is often expected, typically the concentration. This is equivalent to the ADaM AVAL variable and will be included in addition to AVAL for ADPPK. The relative time variables are listed in the table below. Also below are the expected variables unique to ADPPK and the numeric covariates.

The Population PK Analysis Data (ADPPK) follows the CDISC Implementation Guide (https://www.cdisc.org/standards/foundational/adam/basic-data-structure-adam-poppk-implementation-guide-v1-0). Population PK models generally make use of nonlinear mixed effects models that require numeric variables. The data used in the models will include both dosing and concentration records, relative time variables, and numeric covariate variables. A `DV` or dependent variable is often expected. This is equivalent to the ADaM `AVAL` variable and will be included in addition to `AVAL` for ADPPK.

0.3 `{admiral}`

{admiral} is an open-source R package for creating CDISC analysis datasets. It is modular and consists of a set of functions for many of the operations required for dataset construction. There are template programs available for most CDISC ADaM datasets including for PK (i.e. ADPPK, ADNCA and ADPP). The table to the right shows the functions used in the creation of the ADPPK dataset. Follow the link in the QR code below to see an ADPPK template example and run the code in Posit Cloud.

Figure 1: Pharmaverse Examples

The following steps are included in the programming workflow.

- Load packages
- Load Specs with {metacore}
- Derive PC Dates
- Expand Dosing Records
- Find First Dose
- Find Previous Dose
- Find Previous Nominal Dose
- Derive Covariates Using {metacore}
- {metacore} Checks
- {xportr} Steps

0.4 First Load Packages

First we will load the packages required for our project. We will use `{admiral}` for the creation of analysis data. `{admiral}` requires `{dplyr}`, `{lubridate}` and `{stringr}`. We will use `{metacore}` and `{metatools}` to store and manipulate metadata from our specifications. We will use `{xportr}` to perform checks on the final data and export to a transport file.

The source SDTM data will come from the CDISC pilot study data stored in `{pharmaversesdtm}`.

```r
# Load Packages
library(admiral)
library(dplyr)
library(lubridate)
library(stringr)
library(metacore)
library(metatools)
library(xportr)
library(readr)
library(pharmaversesdtm)
library(pharmaverseadam)
```

0.5 Next Load Specifications for {metacore}

We have saved our specifications in an Excel file and will load them into {metacore} with the metacore::spec_to_metacore() function. The spec file can be found here

```r
# ---- Load Specs for Metacore ----
metacore <- spec_to_metacore("pk_spec.xlsx") %>%
  select_dataset("ADPPK")
```

0.6 Load Source Datasets

We will load are SDTM data from {pharmaversesdtm}. The main components of this will be exposure data from EX and pharmacokinetic concentration data from PC. We will use ADSL for baseline characteristics and we will derive additional baselines from vital signs VS and laboratory data LB.

```r
# ---- Load source datasets ----
# Load PC, EX, VS, LB and ADSL
data("pc")
data("ex")
data("vs")
data("lb")

data("adsl")

ex <- convert_blanks_to_na(ex)
pc <- convert_blanks_to_na(pc)
```

```
vs <- convert_blanks_to_na(vs)
lb <- convert_blanks_to_na(lb)
```

0.7 Derivations

0.7.1 Derive PC Dates

At this step, it may be useful to join `ADSL` to your `PC` and `EX` domains as well. Only the `ADSL` variables used for derivations are selected at this step. The rest of the relevant `ADSL` variables will be added later.

In this case we will keep `TRTSDT`/`TRTSDTM` for day derivation and `TRT01P`/`TRT01A` for planned and actual treatments.

In this segment we will use `derive_vars_merged()` to join the `ADSL` variables and the following `{admiral}` functions to derive analysis dates, times and days:

- `derive_vars_dtm()`
- `derive_vars_dtm_to_dt()`
- `derive_vars_dtm_to_tm()`
- `derive_vars_dy()`

We will also create `NFRLT` for `PC` data based on `PCTPTNUM`. We will create an event ID (`EVID`) of 0 for concentration records and 1 for dosing records.

```
# ---- Derivations ----

# Get list of ADSL vars required for derivations
adsl_vars <- exprs(TRTSDT, TRTSDTM, TRT01P, TRT01A)

pc_dates <- pc %>%
  # Join ADSL with PC (need TRTSDT for ADY derivation)
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = adsl_vars,
    by_vars = exprs(STUDYID, USUBJID)
  ) %>%
  # Derive analysis date/time
  # Impute missing time to 00:00:00
  derive_vars_dtm(
    new_vars_prefix = "A",
    dtc = PCDTC,
    time_imputation = "00:00:00"
  ) %>%
  # Derive dates and times from date/times
  derive_vars_dtm_to_dt(exprs(ADTM)) %>%
  derive_vars_dtm_to_tm(exprs(ADTM)) %>%
  # Derive event ID and nominal relative time from first dose (NFRLT)
  mutate(
    EVID = 0,
    DRUG = PCTEST,
    NFRLT = if_else(PCTPTNUM < 0, 0, PCTPTNUM), .after = USUBJID
  )
```

0.7.2 Get Dosing Information

Next we will also join `ADSL` data with `EX` and derive dates/times. This section uses the `{admiral}` functions `derive_vars_merged()`, `derive_vars_dtm()`, and `derive_vars_dtm_to_dt()`. Time is imputed to 00:00:00 here for reasons specific to the sample data. Other imputation times may be used based on study details. Here we create `NFRLT` for `EX` data based on `VISITDY` using the formula `(VISITDY - 1) * 24` using `dplyr::mutate`.

```
# ---- Get dosing information ----

ex_dates <- ex %>%
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = adsl_vars,
    by_vars = exprs(STUDYID, USUBJID)
  ) %>%
  # Keep records with nonzero dose
  filter(EXDOSE > 0) %>%
  # Add time and set missing end date to start date
  # Impute missing time to 00:00:00
  # Note all times are missing for dosing records in this example data
  # Derive Analysis Start and End Dates
  derive_vars_dtm(
    new_vars_prefix = "AST",
    dtc = EXSTDTC,
    time_imputation = "00:00:00"
  ) %>%
  derive_vars_dtm(
    new_vars_prefix = "AEN",
    dtc = EXENDTC,
    time_imputation = "00:00:00"
  ) %>%
  # Derive event ID and nominal relative time from first dose (NFRLT)
  mutate(
    EVID = 1,
    NFRLT = 24 * (VISITDY - 1), .after = USUBJID
  ) %>%
  # Set missing end dates to start date
  mutate(AENDTM = case_when(
    is.na(AENDTM) ~ ASTDTM,
    TRUE ~ AENDTM
  )) %>%
  # Derive dates from date/times
  derive_vars_dtm_to_dt(exprs(ASTDTM)) %>%
  derive_vars_dtm_to_dt(exprs(AENDTM))
```

0.7.3 Expand Dosing Records

The `{admiral}` function `create_single_dose_dataset()` will be used to expand dosing records between the start date and end date. The nominal time will also be expanded based on the values of `EXDOSFRQ`, for example “QD” will result in nominal time being incremented by 24 hours and “BID” will result in nominal time being incremented by 12 hours.

```
# ---- Expand dosing records between start and end dates ----
# Updated function includes nominal_time parameter

ex_exp <- ex_dates %>%
  create_single_dose_dataset(
    dose_freq = EXDOSFRQ,
    start_date = ASTDT,
    start_datetime = ASTDTM,
    end_date = AENDT,
    end_datetime = AENDTM,
    nominal_time = NFRLT,
    lookup_table = dose_freq_lookup,
    lookup_column = CDISC_VALUE,
    keep_source_vars = exprs(
      STUDYID, USUBJID, EVID, EXDOSFRQ, EXDOSFRM,
      NFRLT, EXDOSE, EXDOSU, EXTRT, ASTDT, ASTDTM, AENDT, AENDTM,
      VISIT, VISITNUM, VISITDY,
      TRT01A, TRT01P, DOMAIN, EXSEQ, !!!adsl_vars
    )
  ) %>%
  # Derive AVISIT based on nominal relative time
  # Derive AVISITN to nominal time in whole days using integer division
  # Define AVISIT based on nominal day
  mutate(
    AVISITN = NFRLT %/% 24 + 1,
    AVISIT = paste("Day", AVISITN),
    ADTM = ASTDTM,
    DRUG = EXTRT
  ) %>%
  # Derive dates and times from datetimes
  derive_vars_dtm_to_dt(exprs(ADTM)) %>%
  derive_vars_dtm_to_tm(exprs(ADTM)) %>%
  derive_vars_dtm_to_tm(exprs(ASTDTM)) %>%
  derive_vars_dtm_to_tm(exprs(AENDTM))
```

0.7.4 Find First Dose

We find the first dose for the concentration records using the `{admiral}` function `derive_vars_merged()`

```
# ---- Find first dose per treatment per subject ----
# ---- Join with ADPPK data and keep only subjects with dosing ----

adppk_first_dose <- pc_dates %>%
  derive_vars_merged(
    dataset_add = ex_exp,
    filter_add = (!is.na(ADTM)),
    new_vars = exprs(FANLDTM = ADTM, EXDOSE_first = EXDOSE),
    order = exprs(ADTM, EXSEQ),
    mode = "first",
    by_vars = exprs(STUDYID, USUBJID, DRUG)
  ) %>%
  filter(!is.na(FANLDTM)) %>%
  # Derive AVISIT based on nominal relative time
  # Derive AVISITN to nominal time in whole days using integer division
```

```
    # Define AVISIT based on nominal day
    mutate(
      AVISITN = NFRLT %/% 24 + 1,
      AVISIT = paste("Day", AVISITN),
    )
```

0.7.5 Find Previous Dose

For ADPPK we will find the previous dose with respect to actual time and nominal time. We will use ‘derive_vars_joined().

```
# ---- Find previous dose  ----

adppk_prev <- adppk_first_dose %>%
  derive_vars_joined(
    dataset_add = ex_exp,
    by_vars = exprs(USUBJID),
    order = exprs(ADTM),
    new_vars = exprs(
      ADTM_prev = ADTM, EXDOSE_prev = EXDOSE, AVISIT_prev = AVISIT,
      AENDTM_prev = AENDTM
    ),
    join_vars = exprs(ADTM),
    join_type = "all",
    filter_add = NULL,
    filter_join = ADTM > ADTM.join,
    mode = "last",
    check_type = "none"
  )
```

0.7.6 Find Previous Nominal Dose

```
# ---- Find previous nominal dose ----

adppk_nom_prev <- adppk_prev %>%
  derive_vars_joined(
    dataset_add = ex_exp,
    by_vars = exprs(USUBJID),
    order = exprs(NFRLT),
    new_vars = exprs(NFRLT_prev = NFRLT),
    join_type = "all",
    join_vars = exprs(NFRLT),
    filter_add = NULL,
    filter_join = NFRLT > NFRLT.join,
    mode = "last",
    check_type = "none"
  )
```

0.7.7 Combine PC and EX Data

Here we combine `PC` and `EX` records. We will derive the relative time variables `AFRLT` (Actual Relative Time from First Dose), `APRLT` (Actual Relative Time from Previous Dose), and `NPRLT` (Nominal Relative Time from Previous Dose). Use `derive_vars_duration()` to derive `AFRLT` and `APRLT`. Note we defined `EVID` above with values of 0 for observation records and 1 for dosing records.

```
# ---- Combine ADPPK and EX data ----
# Derive Relative Time Variables

adppk_aprlt <- bind_rows(adppk_nom_prev, ex_exp) %>%
  group_by(USUBJID, DRUG) %>%
  mutate(
    FANLDTM = min(FANLDTM, na.rm = TRUE),
    min_NFRLT = min(NFRLT, na.rm = TRUE),
    maxdate = max(ADT[EVID == 0], na.rm = TRUE), .after = USUBJID
  ) %>%
  arrange(USUBJID, ADTM) %>%
  ungroup() %>%
  filter(ADT <= maxdate) %>%
  # Derive Actual Relative Time from First Dose (AFRLT)
  derive_vars_duration(
    new_var = AFRLT,
    start_date = FANLDTM,
    end_date = ADTM,
    out_unit = "hours",
    floor_in = FALSE,
    add_one = FALSE
  ) %>%
  # Derive Actual Relative Time from Reference Dose (APRLT)
  derive_vars_duration(
    new_var = APRLT,
    start_date = ADTM_prev,
    end_date = ADTM,
    out_unit = "hours",
    floor_in = FALSE,
    add_one = FALSE
  ) %>%
  # Derive APRLT
  mutate(
    APRLT = case_when(
      EVID == 1 ~ 0,
      is.na(APRLT) ~ AFRLT,
      TRUE ~ APRLT
    ),
    NPRLT = case_when(
      EVID == 1 ~ 0,
      is.na(NFRLT_prev) ~ NFRLT - min_NFRLT,
      TRUE ~ NFRLT - NFRLT_prev
    )
  )
```

0.7.8 Derive Analysis Variables

The expected analysis variable for ADPPK is DV or dependent variable. For this example DV is set to the numeric concentration value PCSTRESN. We will also include AVAL equivalent to DV for consistency with CDISC ADaM standards. MDV missing dependent variable will also be included.

```
# ---- Derive Analysis Variables ----
# Derive actual dose DOSEA and planned dose DOSEP,
# Derive AVAL and DV

adppk_aval <- adppk_aprlt %>%
  mutate(
    # Derive Actual Dose
    DOSEA = case_when(
      EVID == 1 ~ EXDOSE,
      is.na(EXDOSE_prev) ~ EXDOSE_first,
      TRUE ~ EXDOSE_prev
    ),
    # Derive Planned Dose
    DOSEP = case_when(
      TRT01P == "Xanomeline High Dose" ~ 81,
      TRT01P == "Xanomeline Low Dose" ~ 54,
      TRT01P == "Placebo" ~ 0
    ),
    # Derive PARAMCD
    PARAMCD = case_when(
      EVID == 1 ~ "DOSE",
      TRUE ~ PCTESTCD
    ),
    ALLOQ = PCLLOQ,
    # Derive CMT
    CMT = case_when(
      EVID == 1 ~ 1,
      TRUE ~ 2
    ),
    # Derive BLQFL/BLQFN
    BLQFL = case_when(
      PCSTRESC == "<BLQ" ~ "Y",
      TRUE ~ "N"
    ),
    BLQFN = case_when(
      PCSTRESC == "<BLQ" ~ 1,
      TRUE ~ 0
    ),
    AMT = case_when(
      EVID == 1 ~ EXDOSE,
      TRUE ~ NA_real_
    ),
    # Derive DV and AVAL
    DV = PCSTRESN,
    DVID = PCTESTCD,
    AVAL = DV,
    DVL = case_when(
      DV != 0 ~ log(DV),
      TRUE ~ NA_real_
    ),
    # Derive MDV
    MDV = case_when(
      EVID == 1 ~ 1,
      is.na(DV) ~ 1,
      TRUE ~ 0
    ),
    AVALU = case_when(
      EVID == 1 ~ NA_character_,
      TRUE ~ PCSTRESU
    ),
    RLTU = "h",
    USTRESC = PCSTRESC,
    UDTC = format_ISO8601(ADTM),
    II = if_else(EVID == 1, 1, 0),
    SS = if_else(EVID == 1, 1, 0),
    ADDL = 0,
    OCC = 1,
  )
```

0.7.9 Add ASEQ

We add a sequence variable using the `{admiral}` function `derive_var_obs_number()`.

```
# ---- Add ASEQ ----

adppk_aseq <- adppk_aval %>%
  # Calculate ASEQ
  derive_var_obs_number(
    new_var = ASEQ,
    by_vars = exprs(STUDYID, USUBJID),
    order = exprs(AFRLT, EVID),
    check_type = "error"
  ) %>%
  mutate(
    PROJID = DRUG,
    PROJIDN = 1,
    PART = 1,
  )
```

0.8 Derive Covariates Using Metacore

In this step we will create our numeric covariates using the `create_var_from_codelist()` function from `{metatools}`.

```r
#---- Derive Covariates ----
# Include numeric values for STUDYIDN, USUBJIDN, SEXN, RACEN etc.

covar <- adsl %>%
  create_var_from_codelist(metacore, input_var = STUDYID, out_var = STUDYIDN) %>%
  create_var_from_codelist(metacore, input_var = SEX, out_var = SEXN) %>%
  create_var_from_codelist(metacore, input_var = RACE, out_var = RACEN) %>%
  create_var_from_codelist(metacore, input_var = ETHNIC, out_var = AETHNIC) %>%
  create_var_from_codelist(metacore, input_var = AETHNIC, out_var = AETHNICN) %>%
  create_var_from_codelist(metacore, input_var = ARMCD, out_var = COHORT) %>%
  create_var_from_codelist(metacore, input_var = ARMCD, out_var = COHORTC) %>%
  create_var_from_codelist(metacore, input_var = COUNTRY, out_var = COUNTRYN) %>%
  create_var_from_codelist(metacore, input_var = COUNTRY, out_var = COUNTRYL) %>%
  mutate(
    STUDYIDN = as.numeric(word(USUBJID, 1, sep = fixed("-"))),
    SITEIDN = as.numeric(word(USUBJID, 2, sep = fixed("-"))),
    USUBJIDN = as.numeric(word(USUBJID, 3, sep = fixed("-"))),
    SUBJIDN = as.numeric(SUBJID),
    ROUTE = unique(ex$EXROUTE),
    FORM = unique(ex$EXDOSFRM),
    REGION1 = COUNTRY,
    REGION1N = COUNTRYN,
    SUBJTYPC = "Volunteer",
  ) %>%
  create_var_from_codelist(metacore, input_var = FORM, out_var = FORMN) %>%
  create_var_from_codelist(metacore, input_var = ROUTE, out_var = ROUTEN) %>%
  create_var_from_codelist(metacore, input_var = SUBJTYPC, out_var = SUBJTYP)
```

0.8.1 Derive Additional Baselines

Next we add additional baselines from vital signs and laboratory data. We will use the `{admiral}` functions `derive_vars_merged()` and `derive_vars_transposed()` to add these.

```r
#---- Derive additional baselines from VS and LB ----

labsbl <- lb %>%
  filter(LBBLFL == "Y" & LBTESTCD %in% c("CREAT", "ALT", "AST", "BILI")) %>%
  mutate(LBTESTCDB = paste0(LBTESTCD, "BL")) %>%
  select(STUDYID, USUBJID, LBTESTCDB, LBSTRESN)

covar_vslb <- covar %>%
  derive_vars_merged(
    dataset_add = vs,
    filter_add = VSTESTCD == "HEIGHT",
    by_vars = exprs(STUDYID, USUBJID),
    new_vars = exprs(HTBL = VSSTRESN)
  ) %>%
  derive_vars_merged(
    dataset_add = vs,
    filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
    by_vars = exprs(STUDYID, USUBJID),
    new_vars = exprs(WTBL = VSSTRESN)
  ) %>%
  derive_vars_transposed(
    dataset_merge = labsbl,
    by_vars = exprs(STUDYID, USUBJID),
    key_var = LBTESTCDB,
    value_var = LBSTRESN
  ) %>%
  mutate(
    BMIBL = compute_bmi(height = HTBL, weight = WTBL),
    BSABL = compute_bsa(
      height = HTBL,
      weight = HTBL,
      method = "Mosteller"
    ),
    CRCLBL = compute_egfr(
      creat = CREATBL, creatu = "SI", age = AGE, weight = WTBL, sex = SEX,
      method = "CRCL"
    ),
    EGFRBL = compute_egfr(
      creat = CREATBL, creatu = "SI", age = AGE, weight = WTBL, sex = SEX,
      method = "CKD-EPI"
    )
  ) %>%
  rename(TBILBL = BILIBL)
```

0.8.2 Combine with Covariates

We combine our covariates with the rest of the data

```r
# Combine covariates with APPPK data

adppk_prefinal <- adppk_aseq %>%
  derive_vars_merged(
    dataset_add = select(covar_vslb, !!!negate_vars(adsl_vars)),
    by_vars = exprs(STUDYID, USUBJID)
  ) %>%
  arrange(STUDYIDN, USUBJIDN, AFRLT, EVID) %>%
  # Add RECSEQ
  # Exclude records if needed
  mutate(
    RECSEQ = row_number(),
    EXCLFCOM = "None"
  ) %>%
  create_var_from_codelist(metacore, input_var = DVID, out_var = DVIDN) %>%
  create_var_from_codelist(metacore, input_var = EXCLFCOM, out_var = EXCLF)
```

0.9 Check Data With Metacore

We use `{metacore}` to perform a number of checks on the data. We will drop variables not in the specs and make sure all the variables from the specs are included.

```r
# Final Steps, Select final variables and Add labels
# This process will be based on your metadata, no example given for this reason
# ...
dir <- "./output"

# Apply metadata and perform associated checks ----
# uses {metatools}

adppk <- adppk_prefinal %>%
  drop_unspec_vars(metacore) %>% # Drop unspecified variables from specs
  check_variables(metacore) %>% # Check all variables specified are present and no more
  check_ct_data(metacore) %>% # Checks all variables with CT only contain values within th
  order_cols(metacore) %>% # Orders the columns according to the spec
  sort_by_key(metacore) # Sorts the rows by the sort keys
```

0.10 Apply Labels and Formats with xportr

Using {xportr} we check variable type, assign variable lenght, add variable labels, add variable formats, and save a transport file.

```
adppk_xpt <- adppk %>%
  xportr_type(metacore) %>% # Coerce variable type to match spec
  xportr_length(metacore) %>% # Assigns SAS length from a variable level metadata
  xportr_label(metacore) %>% # Assigns variable label from metacore specifications
  xportr_format(metacore) %>% # Assigns variable format from metacore specifications
  xportr_df_label(metacore) %>% # Assigns dataset label from metacore specifications
  xportr_write(file.path(dir, "adppk.xpt")) # Write xpt v5 transport file
```

0.11 Save Final Output

Finally we save the final output. We will also create a `CSV` file for the modeler.

```
# ---- Save output ----
saveRDS(adppk, file = file.path(dir, "adppk.rds"), compress = "bzip2")

# Write CSV
write_csv(adppk_xpt, "./output/adppk.csv")
```

1 Example Scripts

ADaM	Sample Code
ADPPK	ad_adppk_spec.R

2 Spec File

pk_spec.xlsx