МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему: «Валідація даних використовуючи SAS Macro»

Виконав: студент 2 курсу, групи МФ-61
спеціальність 122 «Комп'ютерні науки»
освітньо-наукова програма
«Інформатика»

Аксьонов Д.С.
(прізвище та ініціали)

Керівник _____
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Харків – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADaM – Analysis Data Model;
AE – Adverse Events;
CDASH – Clinical Data Acquisition Standards Harmonization;
CDISC – Clinical Data Interchange Standards Consortium;
CRF – Case Report Form;
CRO – Contract Research Organization;
eCRF – Electronic Case Report Form;
EMA – European Medicines Agency;
EOF – End of File;
FDA – U.S. Food and Drug Administration;
GPP – Good Programing Practices;
IG – Implementation Guide;
MedDRA – Medical Dictionary for Regulatory Activities;
PDV – Program Data Vector;
PMDA – Pharmaceuticals and Medical Devices Agency;
QC – Quality Control;
SAP – Statistical Analysis Plan;
SAS – Statistical Analysis System;
SDTM – Study Data Tabulation Model;
TLF – Tables, Listings, Figures;
WHODD – World Health Organization Drug Dictionary.

ВСТУП

1.1 Область дослідження та використання

Якість та тривалість людського життя є одним з найпріоритетніших напрямків досліджень у всьому світі, завдяки чому було сформовано більшість існуючих на даний час знань. Є очевидним, що навколо таких досліджень було сформовано багато інституцій різної направленості та спеціалізації – від дослідницьких інститутів, будь то окремі організації чи структурні підрозділи фармацевтичних компаній, до контролюючих органів (наприклад, FDA, EMA та PMDA). Відповідальними за вивчення, розвиток та впровадження нових лікарських засобів, технологій, тощо, є клінічні дослідження.

Клінічними дослідженнями є такі дослідження, що вивчають нові тести та методи лікування та оцінюють їхній вплив на стан здоров’я людини. Участь у клінічних дослідженнях є винятково добровільною, а предметом вивчення є будь-які медичні втручання, серед яких ліки, біологічні продукти, хірургічні та радіологічні процедури, пристрої, поведінкові методи лікування та профілактики [1].

Клінічні дослідження проводяться у декілька етапів (рис. 1.1), які називаються фазами. В залежності від фази, на якій знаходиться клінічне дослідження, пріоритетними напрямки є визначення побічних ефектів, безпечного рівня дозування та ефективності. Так, на першій фазі дослідження зазвичай вперше випробовують нові ліки на невеликій групі людей, щоб оцінити

безпечний діапазон дозування та виявити побічні ефекти. На другій фазі тестуються методи лікування, які були визнані безпечними на першій фазі, але потребують більшої групи людей для моніторингу будь-яких несприятливих наслідків (adverse effects). Дослідження третьої фази проводяться на більшій вибірці (кількості людей), в різних регіонах і країнах, і часто є кроком безпосередньо перед затвердженням нового методу лікування чи ліків. Дослідження четвертої фази проводяться після схвалення регуляторним органом відповідної країни та потребують подальшого тестування на ще більшій вибірці та протягом більш тривалого періоду часу [1].

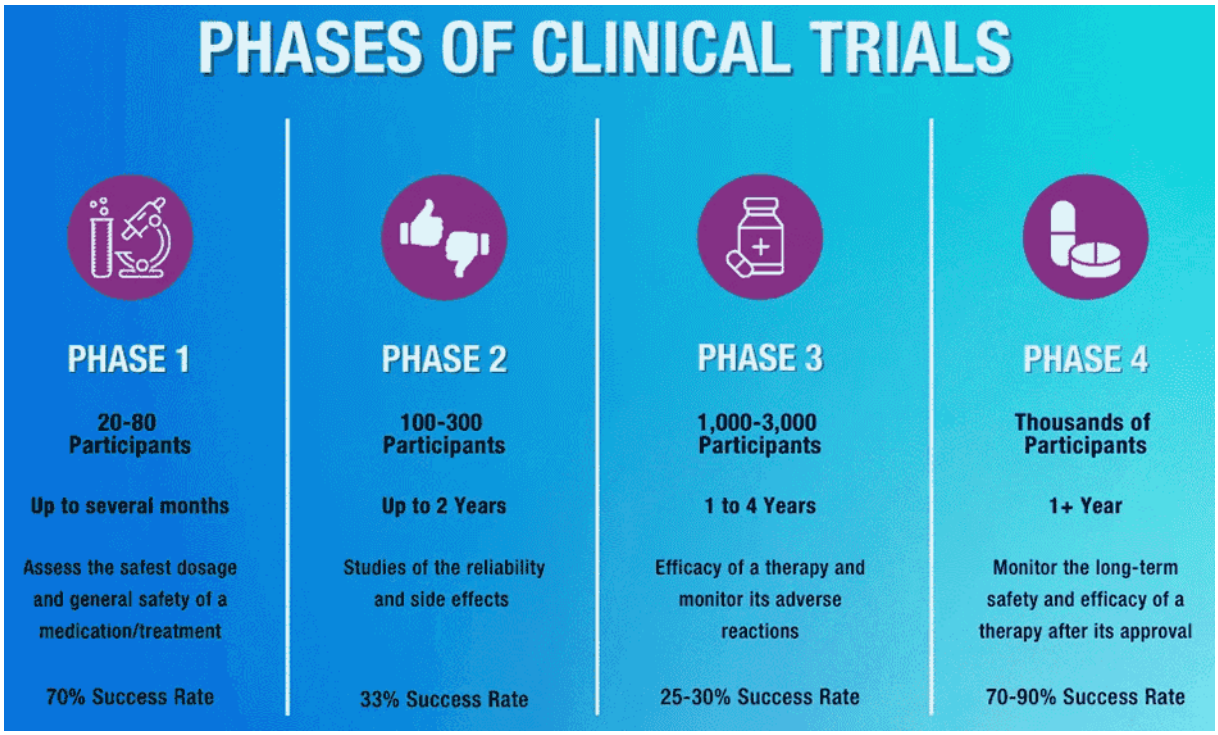

Рисунок 1.1 – Фази клінічних досліджень [2]

Цілком зрозуміло, що для оцінки впливу досліджуваного препарату (наприклад), необхідно зібрати неймовірно велику кількість даних про сам суб’єкт (наприклад, демографічні дані та

фізіологічні характеристики), про процеси, що відбуваються з цим суб’єктом, а також про досліджуваний препарат, що приймає суб’єкт (фармакокінетика, фармакодинаміка, лабораторні тести, фізіологічні показники, тощо). У відповідності до рекомендацій FDA термін «суб'єкт» використовується для загального позначення учасників клінічних досліджень – як пацієнтів, так і здорових добровольців.

1.2 Дані клінічних досліджень

Необхідні дані збираються за допомогою CRF (Case Report Form), зазвичай в електронному вигляді, та зберігаються у вигляді «сирих» датасетів (відповідно до стандарту CDASH). CDASH встановлює стандартний спосіб послідовного збору даних між дослідженнями та спонсорами таким чином, щоб формати та структури збору даних забезпечували чітке відстеження даних, поданих до SDTM, забезпечуючи більшу прозорість для регуляторних органів та інших осіб, які проводять аналіз даних [10]. Таким чином дані сирих датасетів обробляються та зберігаються у вигляді SDTM-датасетів, які фактично є таблицями, стандартизованими за певними вимогами (CDISC). Пізніше, відповідно до вимог аналізу конкретного клінічного дослідження, виконується обробка даних цих SDTM-датасетів, яка фактично являє собою підготовку наявних даних для безпосередньо аналізу. Результатом такої підготовки є створення інших датасетів (ADaM), які також повинні відповідати вимогам CDISC. Сутність процесу зазначеної підготовки, зазвичай, полягає у виконанні певних розрахунків (трансформації даних),

наприклад, визначення тривалості якоїсь події, зміни в лабораторних покажчиках по мірі застосування досліджуваного препарату, час до настання певної події за наявності різних умов, взаємодія з медикаментами, що приймаються паралельно. Результати аналізу представляються у вигляді TLF – таблиць, лістингів, фігур. Один із можливих сценаріїв потоку даних аналізу наведено на рис. 1.2.

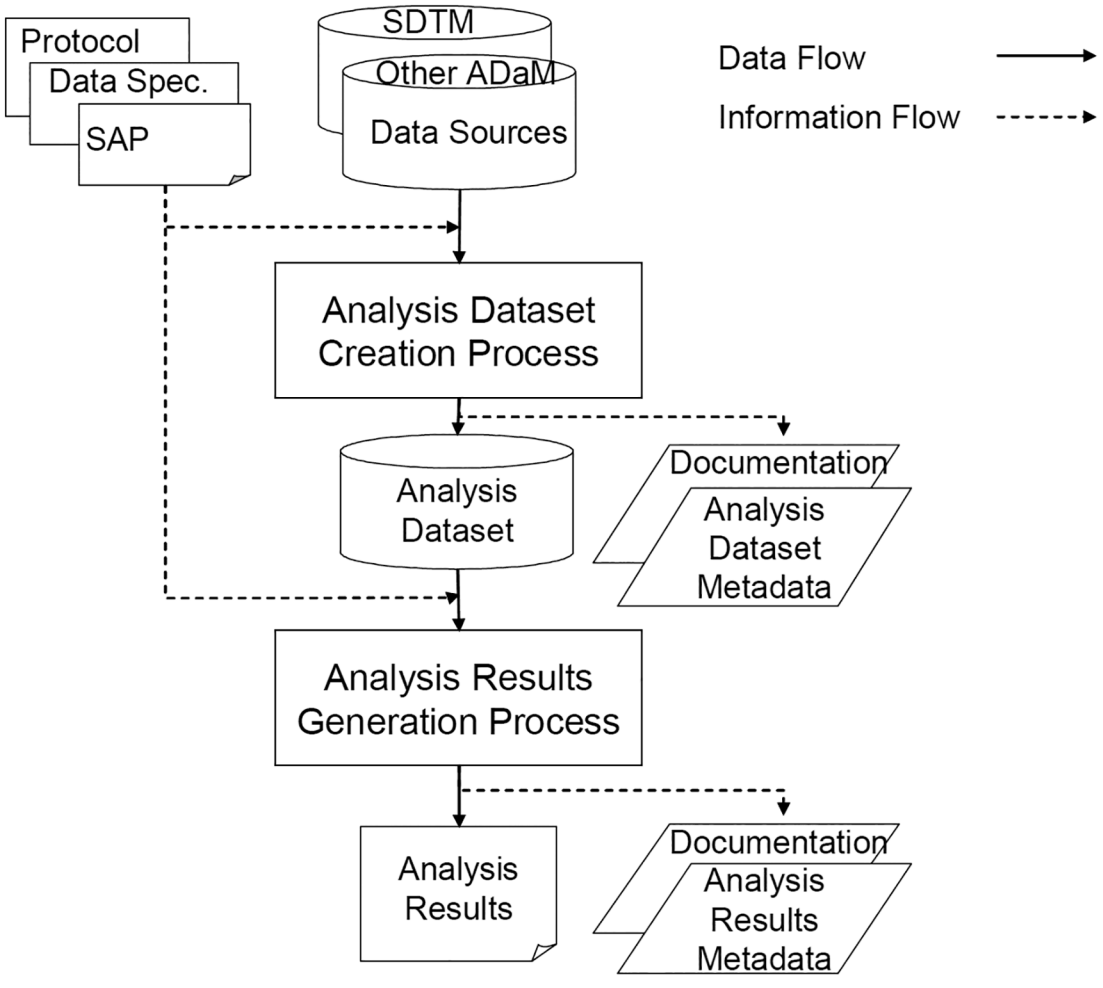

Рисунок 1.2 – Діаграма потоку даних для аналізу, що показує один з можливих сценаріїв потоку даних та інформації [11]

Обробка та трансформація даних, вочевидь, виконується наявними засобами програмування. Принципового значення яка

саме мова програмування використовується немає, але індустріальним стандартом фактично є SAS (див. розд. 2.1).

Ключовим моментом програмування (вищезазначеного перетворення даних) є наявність декількох етапів. В більшості випадків програмування виконується двічі різними програмістами, а результат – отримані датасети – порівнюються між собою [3]. Метою такого підходу є мінімізація впливу людського фактору, хоча також зазначається, що виключити його повністю таким чином не є можливим [4]. Результат програмування вважається достовірним у випадку, якщо датасети, отримані на обох етапах програмування, є ідентичними.

1.3 Обробка даних клінічних досліджень

На протязі свого «життєвого циклу» дані клінічних досліджень проходять декілька етапів. Першим етапом є безпосередньо збір необхідних даних шляхом заповнення спеціальних форм – CRF. Обсяг та тип даних визначається та суворо обмежується вимогами самого клінічного дослідження, які зазначені в його протоколі (є необхідною складовою документації будь-якого дослідження). Наприклад, якщо терапевтичною галуззю дослідження є гепатологія, тести, пов’язані з дослідженням очного дна, вочевидь, не є потрібними, а більшість даних будуть так або інакше пов’язані з печінковими пробами. Але також є і обов’язкові до збору дані, до них відносяться демографічні характеристики суб’єктів, що прийматимуть участь у клінічних дослідженнях, такі як стать, вік, раса, етнічне походження, та дані, що пов’язані з безпекою суб’єктів –

наприклад, спостережувані побічні ефекти (незалежно від того чи пов’язані вони із самим дослідженням, чи ні).

Зібрані в CRF дані можуть містити безліч неточностей, помилок та одруківок, форматів (наприклад формат дат), а також декілька мов. Перехід від паперової форми CRF до електронної (eCRF) хоч і суттєво зменшує кількість зазначених неточностей, не гарантує відсутність помилок, пов’язаних з людським фактором, а отже попередня обробка даних є обов’язковою. Приклад зовнішнього вигляду eCRF наведено на рис. 1.3.

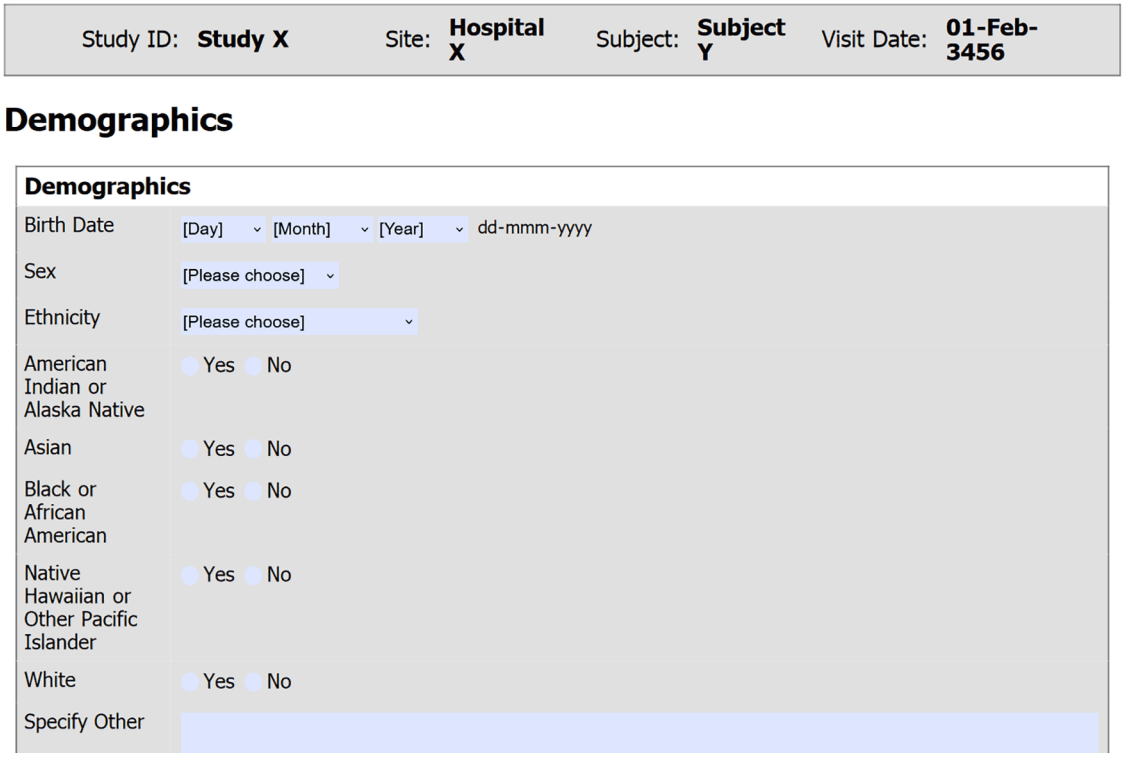

Рисунок 1.3 – Приклад зовнішнього вигляду незаповненої неанотованої eCRF, сторінка демографічних характеристик [12]

Після попередньої обробки, дані підлягають кодуванню (у багатьох випадках) та структуризації відповідно до обов’язкових стандартів. Такий підхід гарантує, що будь-хто знайомий з цими стандартами знає де знаходяться дані у датасеті, який вигляд

мають та звідки походять. За приклад можна взяти перелік медикаментів, що суб'єкт приймає впродовж клінічного дослідження (окрім ліків, що є безпосередньо предметом дослідження): діюча речовина може бути присутня у безлічі препаратів як у вигляду основної речовини, так і вигляді допоміжної. Тому одночасно з назвою медичного засобу, важливо знати його фармакологічний склад, кожен з компонентів має відповідний код, за яким, власне, і можна однозначно ідентифікувати ту чи іншу діючу речовину. Присвоєння (співставлення) коду діючій речовині відповідно до стандартного словника (наприклад, WHODD) і є процесом кодування. Таким чином, після цього етапу всі дані отримані у процесі клінічного дослідження містяться у SDTM-датасетах (рис. 1.4), структуровані відповідно до CDISC-стандартів, та закодовані (де необхідно) у відповідності до словників (WHODD, MedDRA, тощо).

STUDYID	DOMAIN	USUBJID	SUBJID	RFSTDTC	RFENDTC	SITEID	INVNAM	BRTHDTC	AGE	AGEU	SEX	RACE	ETHNIC	ARMCD
ABC123	DM	ABC12301001	01001	2006-01-12	2006-03-10	01	JOHNSON, M	1948-12-13	57	YEARS	M	WHITE	HISPANIC OR LATINO	A
ABC123	DM	ABC12301002	01002	2006-01-15	2006-02-28	01	JOHNSON, M	1955-03-22	50	YEARS	M	WHITE	NOT HISPANIC OR LATINO	P
ABC123	DM	ABC12301003	01003	2006-01-16	2006-03-19	01	JOHNSON, M	1938-01-19	68	YEARS	F	BLACK OR AFRICAN AMERICAN	NOT HISPANIC OR LATINO	P
ABC123	DM	ABC12301004	01004			01	JOHNSON, M	1941-07-02			M	ASIAN	NOT HISPANIC OR LATINO	
ABC123	DM	ABC12302001	02001	2006-02-02	2006-03-31	02	GONZALEZ, E	1950-06-23	55	YEARS	F	AMERICAN INDIAN OR ALASKA NATIVE	NOT HISPANIC OR LATINO	P
ABC123	DM	ABC12302002	02002	2006-02-03	2006-04-05	02	GONZALEZ, E	1956-05-05	49	YEARS	F	NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDERS	NOT HISPANIC OR LATINO	A

Рисунок 1.4 – Приклад фрагменту SDTM-датасету DM (Demographics) [9]

Наступним етапом обробки даних є їхня підготовка до аналізу, відповідно до потреб, зазначених у плані статистичного аналізу (SAP). Прикладом такої підготовки може бути визначення

бейзлайнових значень результатів тестів, зазвичай останнє значення, отримане перед вживанням першої дози досліджуваного медичного препарату або застосування медичного пристрою. Також бейзлайновим може вважатися значення, яке, наприклад, є середнім арифметичним від щонайменше чотирьох тестів, які отримані з інтервалом не менше семи днів, а останній тест зроблено за двадцять днів до першої дози препарату чи використання медпристрою. На додаток, після визначення бейзлайнового значення, зазвичай, потрібно дослідити зміни в результатах певного тесту відносно бейзлайнового значення по мірі проходження суб'єктом клінічного дослідження, тобто в залежності від часу (та сумарної дози) приймання препарату суб'єктом. Перетворені під потреби аналізу дані зберігаються у ADaM-датасетах, які також підлягають стандартизації у відповідності до CDISC-вимог, хоча і з значно меншою жорсткістю вимог ніж SDTM-датасети.

Останнім етапом «життєвого циклу» даних, якщо не враховувати архівацію та зберігання, можна вважати їхнє представлення у таблицях (рис 1.5), лістингах та фігурах (TLF). На основі безперервних даних, зазвичай (щонайменше), розраховуються середнє, медіана та стандартне відхилення, а на основі категоріальних даних розраховуються частоти якихось подій чи результатів тестів, так звані зсуви відносно бейзлайну (наприклад, перехід від «нормального» значення на бейзлайні до «ненормального» на восьмий тиждень приймання досліджуваного препарату) і таке інше.

Table 1.1 Summary of Adverse Events
Safety Population

	Treatment A (N=46) n (%)
Subjects with AE	46 (100.0%)
Subjects with TEAE	45 (97.8%)
Subjects with Treatment-Related TEAE	32 (69.6%)
Subjects with Serious TEAE	20 (43.5%)
Subjects with Serious Treatment-Related TEAE	17 (37.0%)

Percentages are based on the number of subjects in safety population.

Рисунок 1.5 – Приклад таблиці з частотами побічних ефектів (AE, Adverse Events)

1.4 Валідація даних

Оскільки від моменту отримання до моменту використання для аналізу дані клінічних досліджень зазнають велику кількість перетворень, є очевидним, що впродовж цього процесу будуть відбуватися помилки. Наявність помилок обробки є вкрай небажаною, тому, що призводить до викривлення результатів дослідження. Викривлення результатів призводить до хибних висновків щодо ефективності досліджуваних ліків і, що найголовніше, негативно відбивається на безпеці потенційних споживачів. Це означає, що кількість помилок необхідно звести до мінімуму, а сам мінімум визначається ступенем ризику (небезпеки).

Валідація, або контроль якості даних клінічних досліджень, є складовою частиною концепції подвійного програмування. Основним призначенням валідації є запобігання появі помилок в результаті обробки та інтерпретації даних. Валідація відбувається на кожному з описаних вище етапів перетворення даних: при створенні SDTM-датасетів, ADaM-датасетів, TLF-датасетів та самих TLF. Очевидно, що повністю виключити появу помилок неможливо, але можливо мінімізувати ризик їхньої появи.

Основою процесу валідації є перевірка на ідентичність датасетів, отриманих різними програмістами. В ідеалі, для отримання датасетів програмісти повинні використовувати різний код, хоча гарантувати це без попередніх домовленостей між ними доволі складно. До вторинних, але не менш важливих процесів валідації відноситься перевірка на відповідність та дотримання стандартів CDISC, що включає перевірку метаданих – атрибутів змінних (стовпців) у датасетах, унікальності ключів датасетів, сортування датасетів та перевірка логів виконання програм (обох програмістів).

Додаткові складності при порівнянні датасетів виникають коли довжина змінних, що містяться в них є відмінною. В SAS існують тільки два типи змінних – це числові та символьні. Довжина числових фактично однакова (змінювати в певних межах можливо, але зазвичай цим не користуються), а от довжина символьних змінних може розрізнятися суттєво. Така відмінність викликана відсутністю обмежень у стандартах – лише невелика кількість символьних змінних має довжину явно регламентовану стандартами. Обмеження для усіх інших символьних змінних – 200 символів. Специфіка символьних

змінних у SAS полягає ще й у тому, що їхня довжина не залежить від значень, які в них містяться: якщо значення більше за довжину змінної, то воно буде обрізано, а якщо коротше – різницю будуть складати пробіли, додані у кінець значення змінної.

Очевидно, що відсутність домовленостей між програмістами або відсутність якихось внутрішніх стандартів (всередині інституції, де виконується робота) призведе до суттєвих відмінностей у довжинах. Домовленості щодо довжин змінних повинні діяти не тільки в межах певного датасету, а в межах всієї сукупності датасетів, які розробляються в даному клінічному дослідженні. Оскільки різні датасети програмують різні люди, зазначених домовленостей повинно бути досягнено серед усіх програмістів клінічного дослідження, що майже ніколи неможливо з урахуванням витраченого на це часу.

Коли всі датасети запрограмовано та перевірено, вони збираються у пакет, що буде надсилатися регулятору (наприклад, FDA). Одним з етапів приготування цього пакету даних є мінімізація довжин символьних змінних. Тобто для змінних встановлюється довжина, що обумовлена максимально довгим її значенням. З урахуванням вищезазначеного, варто проводити мінімізацію довжин змінних датасету перед його валідацією.

1.5 Мета роботи

Метою даної роботи є створення підпрограм (макросів) для валідації (порівняння та перевірки) датасетів та звітування

знайдених відмінностей за допомогою SAS Macro Facility для використання у програмному пакеті SAS різних модифікацій.

Подібні програми (макроси) є обов’язковою частиною кодів всіх пов’язаних з клінічними дослідженнями інституцій – фармацевтичних компаній та CRO (Contract Research Organization). Але вихідні коди цих макросів є закритими та створеними з урахуванням специфіки роботи цих інституцій, що обумовлює неможливість або складність їхнього використання іншими, не пов’язаними з цими інституціями, третіми особами. Наявні ж у відкритих джерелах макроси, зазвичай, написані з урахуванням якихось специфічних потреб, які занадто ускладнюють процес перевірки або, наприклад, глобалізують його [5] – тобто відсутній модульний підхід, який би давав змогу використовувати тільки певний функціонал в залежності від поточних задач. Необхідно також зазначити, що комерційне використання макросів (наприклад CRO), отриманих із відкритих джерел, не завжди є припустимим у зв’язку з потенційними ризиками, пов’язаними з правами на інтелектуальну власність.

Робота складається зі вступу, однієї глави, висновків, переліку використаних джерел із 13 найменувань та 5 додатків. Повний обсяг магістерської роботи складає 83 сторінки тексту та містить 12 рисунків.

ВАЛІДАЦІЯ ДАНИХ SAS-МАКРОСАМИ

2.1 Середовище розробки SAS

Середовище розробки SAS (абревіатура раніше розшифровувалася як «Statistical Analysis System», наразі є не більш ніж торговою маркою) від початку розроблялася для проведення та аналізу сільськогосподарських експериментів. Але на даний момент є найбільш використовуваною модульною багатофункціональною платформою для виконання статистичних розрахунків; обробки, трансформації та аналізу даних; та звітування результатів цих маніпуляцій з даними [6 – 8]. Середовище розробки SAS складається з наступних компонентів (наведено основні):

– Base SAS – data management and basic procedures;
– SAS/STAT – statistical analysis;
– SAS/GRAPH – presentation quality graphics;
– SAS/OR – Operations research;
– SAS/ETS – Econometrics and Time Series Analysis;
– SAS/IML – interactive matrix language;
– SAS/AF – applications facility (menus and interfaces);
– SAS/QC – quality control.

Середовище розробки SAS має декілька імплементацій, SAS Base, SAS Enterprise Guide, SAS Studio/Viya. Кожна з них має надбудову, так званий макропроцесор (SAS Macro Facility), яка дає змогу змінювати код програми, що підлягає виконанню, в залежності від додаткових умов. Іншими словами, спочатку

виконується програма самого макросу, яка формує або модифікує код основної програми, і тільки після цього відбувається виконання основного коду SAS-програми. Зрозуміло, що такий підхід у програмуванні надає потрібної гнучкості програмі в цілому, та розширює потенціал основної мови програмування SAS.

В даній роботі використання мови програмування та середовища SAS зумовлене декількома факторами, основними є гнучкість за використанням макропроцесору та поширеність застосування мови у сфері клінічних досліджень – де-факто, SAS досі є стандартом в цій сфері.

2.2 Постановка задачі та функціональний опис

З метою пришвидшення процесу валідації датасетів клінічних досліджень, підвищення якості результатів валідації, а також часткової її автоматизації пропонується створення сукупності макросів, написаних за використанням SAS Macro Facility та SAS мови програмування, що забезпечать виконання наступних задач:

– порівняння двох датасетів з точки зору їхньої ідентичності;

– перевірка лейблів (назв) датасетів;

– перевірка однаковості ключів сортування датасетів;

– порівняння порядку розташування змінних (стовпців) всередині датасетів;

– мінімізація довжин змінних датасетів.

Гнучкість застосування планується підвищити шляхом впровадження модульності – необхідний функціонал програми

буде відтворено у сукупності вузькоспеціалізованих макросів, що є незалежними один від одного.

Сукупність макросів повинна мати наступний функціонал:

– порівняння значень всіх спільних змінних датасетів;

– виявлення змінних, що присутні тільки в одному з датасетів;

– виявлення записів (рядків) датасетів, що присутні тільки в одному з датасетів;

– порівняння метаданих змінних датасетів, таких як типи, довжини, лейбли та формати;

– протоколювання виявлених відмінностей у даних та метаданих змінних до логу виконання програми у вигляді помилок;

– перевірка наявності лейблів датасетів та порівняння лейблів датасетів між собою, протоколювання результатів перевірки до логу виконання програми у вигляді помилок;

– протоколювання відмінностей у порядку змінних до логу програми у вигляді помилок;

– протоколювання відмінностей у ключах сортування датасетів до логу програми у вигляді помилок;

– створення датасету, який містить перелік всіх змінних обох датасетів з вказанням їхнього розташування всередині кожного з порівнюваних датасетів, а також індикатор відповідності розташування;

– виявлення мінімально припустимої довжини кожної з символьних змінних, що присутні в датасеті, встановлення якої не призведе до обрізання даних всередині змінної;

– створення датасету, що містить перелік всіх змінних, їхніх максимальних за довжиною значень та відповідних довжин;

– перестворення датасетів з мінімізованими довжинами змінних.

2.3 Валідація даних та метаданих датасетів

Макрос для порівняння ідентичності датасетів та метаданих їхніх змінних розширює функціонал стандартної процедури SAS `proc compare`, має назву `_validate_` та має п’ять вхідних параметрів: `dsin_prod`, `dsin_val`, `dsout_comp`, `id_vars` та `reorder`.

Параметром `dsin_prod` задається перший вхідний датасет для перевірки, зазвичай це датасет який необхідно перевірити. Параметром `dsin_val` задається другий вхідний датасет для перевірки на ідентичність, зазвичай це датасет, яким перевіряють. В більшості випадків валідацію виконує програміст другого датасету, йому важливо знати в якому датасеті помилки для подальших виправлень – він або виправляє власний датасет, або просить першого програміста виправити свій. Але з точки зору реалізації макросу немає різниці в якому з параметрів який датасет вказано. Це також неважливо коли результат перевірки свідчить про ідентичність датасетів.

Параметр `id_vars` визначає ключові змінні датасетів (однакові для обох вхідних) за якими буде проводитися порівняння. Параметр є опціональним, якщо значення не задане, перевірку буде виконано порядково. Тобто значення змінних першого рядка першого датасету буде порівняно зі значеннями змінних першого рядка другого датасету, значення другого рядка

першого датасету – зі значеннями другого рядка другого датасету, і так далі. У випадку коли датасети мають різну кількість записів (рядків) і неспівпадіння саме в цьому, використання параметру id_vars призведе до того, що буде показано саме на зайві рядки і зазначено, що всі інші рядки порівнюваних датасетів ідентичні. В противному випадку (невикористання id_vars), коли процес порівняння дійде до «зайвого» рядка, результатом буде розбіжність по всіх (майже) змінних. Зважаючи на те, що в датасетах можуть бути тисячі рядків, знайти місце виникнення такої розбіжності буде вкрай складно та займе багато часу.

Через значення параметру dsout_comp задається ім’я вихідного датасету, в який буде виведено всі рядки першого датасету та відповідні ним рядки другого (відповідно до порядку порівняння, на який впливає параметр id_vars), в яких знайдено відмінність у значенні хоча б одної з спільних змінних. Змінні ненаявні в одному з датасетів не виводяться до вихідного датасету, а їхня наявність зазначається в звіті. Для демонстрації відмінностей, у датасеті буде додатковий рядок, позначений значенням «DIF» змінної _TYPE_, де розбіжність у значеннях символьних змінних буде позначена символом «X» (посимвольне порівняння), а розбіжність у числових змінних буде позначена у вигляді різниці між значенням змінної першого (рядок, де _TYPE_ = «BASE») та другого (рядок, де _TYPE_ = «COMPARE») датасетів.

Кількість змінних у порівнюваних датасетах може сягати декількох сотень, а порядок в якому вони розташовані у вихідному датасеті (що вказаний у dsout_comp) відповідає порядку першого датасету, тобто того, ім’я якого передано через параметр

dsin_prod. У переважній більшості сценаріїв застосування, таке розташування змінних є дуже не зручним. Якщо відмінність знайдено, наприклад, у сотій змінній, всі попередні змінні необхідно або «промотати» або відключити їхнє відображення. Слід також зазначити, що кількість змінних, що може бути відображеною одночасно, може обмежуватися в залежності від системи, в якій виконується перегляд датасету.

Для зміни методу відображення порядку змінних у вихідному датасеті додано опціональний параметр reorder, можливими значеннями якого є «CHAR» та «NUM». В залежності від його значення, першими після «системних» та ключових змінних буде відповідно виведено або всі символьні змінні що мають відмінності, або числові змінні що мають відмінності.

Слід зазначити, що окрім параметру reorder на послідовність змінних у вихідному датасеті також впливає параметр id_vars. Якщо його значення не пусте, то відразу після «системних» змінних (_TYPE_ та _OBS_) буде виведені змінні, що зазначені в ньому.

Приклад фрагменту датасету з результатами порівняння датасетів (compare-датасет) наведено на рис. 2.1, де рядки з _TYPE_ = «BASE» відповідають першому порівнюваному датасету (заданий у dsin_prod), рядки з _TYPE_ = «COMPARE» – другому порівнюваному датасету (заданий у dsin_val), а рядки з _TYPE_ = «DIF» містять різницю між значеннями змінних порівнюваних датасетів за умови, що такий рядок є присутнім в обох вхідних датасетах. На рис. 2.1 датасет містить інформацію по записам, що присутні тільки в одному з порівнювальних датасетів та дві групи записів, що мають відмінні значення

змінних. Рядок 1 є рядком, що присутній тільки у другому датасеті, а рядок 8 – тільки у першому датасеті. Групи рядків 2 – 4 та 5 – 7 свідчать про те, що у змінних COUNTRY та ACOUNTRY знайдено відмінності, позицію відмінного символу позначено символом «X» у рядках 4 та 7.

	TYPE	_OBS_	USUBJID	COUNTRY	ACOUNTRY	STUDYID	INVID	SITEID	AGE	AGEU
1	COMPARE	3	ABC123.000003	ARG	Argentina	ABC123	4508	999846	65	AGEU
2	BASE	7	ABC123.000008	ARG	Argentina	ABC123	3277	999251	51	AGEU
3	COMPARE	8	ABC123.000008	BRA	Brazil	ABC123	3277	999251	51	AGEU
4	DIF	7	ABC123.000008	X.X	X.XXXXXXX.....	.....	.....	.....	.	.....
5	BASE	11	ABC123.000012	ARG	Argentina	ABC123	9914	999982	31	AGEU
6	COMPARE	12	ABC123.000012	CHN	China	ABC123	9914	999982	31	AGEU
7	DIF	11	ABC123.000012	XXX	XXXXXXXXX.....	.....	.....	.....	.	.....
8	BASE	49	ABC123.000050	CHN	China	ABC123	3233	999802	66	AGEU

Рисунок 2.1 – Фрагмент compare-датасету, що згенерований макросом _validate_

Відмінності у метаданих змінних датасетів виводяться до стандартного звіту, що генерує proc report, а наявність цих відмінностей – у вигляді помилок до логу виконання програми, в якій було здійснено виклик макросу. Приклад записів у лозі програми, що відповідає даним з рис. 2.1 наведено на рис. 2.2. У повідомленні наявні помилки, що свідчать про наявність записів, що присутні тільки в одному з порівнюваних датасетів, відмінності у метаданих змінних, зазначення не рівності датасетів з назвами змінних, в яких знайдено відмінності.

Пояснення щодо ключових фрагментів коду макросу _validate_ наведено нижче, а повний його код можна знайти у додатку 1.

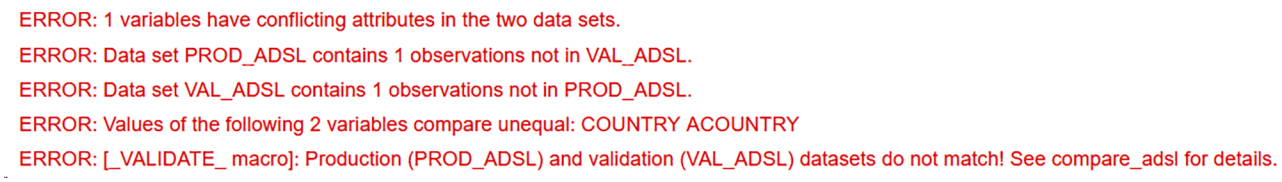

Рисунок 2.2 – Приклад повідомлень з виявленими помилками у лозі програми, що згенерований макросом _validate_

Відповідно до GPP (Good Programing Practices), некоректний виклик програми або підпрограми, в даному випадку – макросу, не повинен призводити до незрозумілих результатів та непрацездатності системи в цілому (нажаль, для SAS це відносно стандартна проблема). Наступна частина коду макросу написана для перевірки значень обов’язкових параметрів, без яких робота макросу не є можливою.

```
%if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
  %put ERR%str(OR:) [_VALIDATE_ macro]: Production and/or validation
datasets were not specified. Exiting macro.;
  %return;
%end;
%if "&dsout_comp." = "" %then %do;
  %put ERR%str(OR:) [_VALIDATE_ macro]: Comparison results dataset was
not specified. Exiting macro.;
  %return;
%end;
```

Якщо виклик макросу здійснюється з відсутнім значенням хоча б одного з параметрів dsin_prod, dsin_val та dsout_comp, роботу макросу буде припинено, а до логу виконання програми буде записана помилка з поясненням. Це дасть змогу користувачеві виправити виклик макросу без читання

документації на макрос. Тут слід відзначити специфічне для SAS камуфлювання слова ERROR. Оскільки SAS за замовчуванням (зазвичай це обов'язкова вимога) записує до логу програми і вихідний код, наявність слова ERROR у вихідному коді призведе до того, що макрос для перевірки цього логу зреагує на це слово як на фактичну помилку – не існує методу розрізнити повідомлення про помилку згенеровану системою SAS, тобто фактичною помилкою, від відповідного слова у лозі програми – в обох випадках буде текстовий рядок з словом ERROR.

Всі використовувані макросом внутрішні змінні потрібно об'являти за допомогою %local. В такому випадку, якщо буде присутня макрозмінна з таким самим ім'ям за межами макросу, її не буде перезаписано, а використовуватиметься змінна з локальної таблиці символів. Хоча такий підхід не є обов'язковим і у багатьох випадках не використовується, його можна також віднести до GPP – робота макросу буде більш передбачуваною щонайменше, оскільки неможливо наперед передбачити в якому середовищі макрос буде викликано.

```
%local comp_nobs usubjid_var dif_cvars dif_nvars;
```

В даному випадку, макрос _validate_ використовує тільки шість локальних макрозмінних.

При порівнянні датасетів за ключовими змінними (параметр id_vars), обов'язковою умовою є наявність сортування даних обох датасетів за цими ключовими змінними. Для випадків «звичайного» порівняння – така умова відсутня, сортування

датасетів лишається на розсуд користувача і тому не входить до макросу, для цього присутній умовний оператор.

```
%if "&id_vars." ^= "" %then %do;
  proc sort data = &dsin_prod. out = work.&dsin_prod._id;
    by &id_vars.;
  run;
  proc sort data = &dsin_val. out = work.&dsin_val._id;
    by &id_vars.;
  run;
  %let dsin_prod = &dsin_prod._id;
  %let dsin_val  = &dsin_val._id;
%end;
```

Додатково виконується оновлення значень параметрів, що спрощує подальший код, а також гарантує, що відсортовані датасети не перезапишуть вхідні датасети. Останнє, у більшості випадків, неприпустимо.

Наступна частина макросу викликає стандартну процедуру proc compare з необхідними налаштуваннями.

```
proc compare base = &dsin_prod. compare = &dsin_val. out = &dsout_comp.
listobs listvars %str(err)or outbase outcomp outnoequal outdif novalues;
  %if "&id_vars." ^= "" %then %do;
    id &id_vars.;
  %end;
run;
```

Оператор id викликається тільки для випадків, коли через параметр макросу id_vars задані ключові змінні для порівняння. Опції outbase та outcomp процедури proc compare гарантують

виведення до вихідного датасету записів, що присутні тільки в одному з порівнюваних датасетів. Завдяки опції outdif до вихідного датасету буде додано додатковий рядок для кожної пари записів вхідних датасетів, які співпадають за значеннями ключових змінних. Опція novalues пригнічує виведення до звіту порівняння значень кожної з наявних змінних – зазвичай не має користі завдяки великій кількості записів та змінних, але займає дуже великих об'єм у звіті. Опції listobs та listvars гарантують що у звіті будуть наведені змінні та записи, що є відсутніми в одному з датасетів.

Подальша обробка даних в макросі а також звітування залежить від того, чи виявлені рядки, в яких данні змінних відрізняються. Якщо такі рядки виявлено, у вихідному датасеті буде присутнім хоча б один запис.

```
data _null_;
  call symputx("comp_nobs", NOBS);
  if 0 then set &dsout_comp. nobs = NOBS;
  stop;
run;
```

Кількість записів у вихідному датасеті записується до макрозмінної comp_nobs. Фактичного читання датасету не виконується задля пришвидшення роботи макросу, а значення до змінної NOBS, яке потім записується до макрозмінної comp_nobs, фактично отримується на етапі компіляції. Саме тому читання датасету не є необхідним – достатньо прочитати його метадані на етапі компіляції. Оператор stop не є обов'язковим, але його наявність (в даній імплементації) запобігає появі попереджень від

SAS про нескінченний цикл читання датасету, що вказано в операторі set. Відбувається це саме тому, що фактичного читання датасету не відбувається, а отже помітка EOF (end of file) також прочитана не буде. Захист SAS це сприймає як потенційну проблему, тому буде записувати відповідні попередження до логу виконання програми.

За виявлення спільних змінних порівнюваних датасетів, що мають різні значення, відповідає наступна частина коду. Виконується ця частина тільки у випадку, коли такі значення наявні – тобто коли вихідний датасет має хоча б один запис.

```
data _null_;
  set &dsout_comp. (where = (_TYPE_ = "DIF")) end = EOF;
  array CVARS[*] _character_;
  array NVARS[*] _numeric_;
  length DIF_CVARS DIF_NVARS $3200;
  retain DIF_CVARS DIF_NVARS;
  do i = 1 to hbound(CVARS);
    if _N_ <= 1 and vname(CVARS[i]) = "USUBJID" then call
symputx("usubjid_var", "USUBJID");
    if index(CVARS[i], "X") and ^index(DIF_CVARS, vname(CVARS[i])) and
vname(CVARS[i]) ^= "_TYPE_" then DIF_CVARS = catx(" ", DIF_CVARS,
vname(CVARS[i]));
  end;
  do i = 1 to hbound(NVARS);
    if ^missing(NVARS[i]) and ^index(DIF_NVARS, vname(NVARS[i])) and
vname(NVARS[i]) ^= "_OBS_" then DIF_NVARS = catx(" ", DIF_NVARS,
vname(NVARS[i]));
  end;
  if EOF then do;
    call symputx("dif_cvars", DIF_CVARS);
    call symputx("dif_nvars", DIF_NVARS);
```

```
    put "NOTE: [_VALIDATE_ macro]: List of unequal character variables: "
DIF_CVARS;
    put "NOTE: [_VALIDATE_ macro]: List of unequal numeric variables: "
DIF_NVARS;
  end;
run;
```

Виконується читання рядків вихідного датасету, в яких змінна _TYPE_ має значення «DIF». Значення всіх символьних змінних «записується» до масиву CVARS, а числових – до масиву NVARS. Слід зазначити, що в SAS масиви являють собою посилання на змінні (дещо подібне до pointer типу в інших мовах програмування) і при створені масивів фактичного копіювання даних не відбувається.

Значення комірок масивів ітеративно перевіряються на наявність значення «X» для символьних змінних та будь-якого значення для числових змінних. Наявність таких значень свідчить про відмінність у відповідних змінних порівнюваних датасетів. Якщо результат пошуку позитивний, ім’я такої змінної буде збережене у змінних DIF_CVARS та DIF_NVARS (залежно від типу змінної), значення яких буде записано відповідно до макрозмінних dif_cvars та dif_nvars для подальшої обробки.

Запис до макрозмінних відбувається тільки коли оброблено увесь вихідний датасет. В цей момент також робиться повідомлення до логу програми виконання з переліком знайдених змінних.

Окремо слід зазначити пошук змінної USUBJID. Ця змінна є обов’язково присутньою майже у кожному з датасетів клінічних досліджень та містить унікальний ідентифікатор суб’єкта

дослідження. Якщо макрос _validate_ викликано з пустим параметром id_vars та у вхідних датасетах знайдено змінну USUBJID, то у вихідному датасеті змінну USUBJID буде розташовано в тому місці, де повинні бути ключові змінні у випадку виклику макросу з непустим параметром id_vars.

Застосування оператору retain до змінних DIF_CVARS та DIF_NVARS зумовлено специфікою роботи SAS. Як тільки відбувається читання наступного запису датасету, що слідує за оператором set, значення всіх змінних спустошуються (очищення PDV – Program Data Vector). Після цього, відбувається оновлення значень змінних, що присутні у датасеті, який читається. Якщо змінної нема в цьому датасеті (в даному випадку це DIF_CVARS та DIF_NVARS), то змінна лишається пустою. Таким чином, по мірі читання датасету всі попередньо знайдені змінні з відмінностями, що зберігалися у DIF_CVARS та DIF_NVARS, буде втрачено. Використання оператору retain запобігає очищенню змінних, але не відміняє читання в них нових значень з датасету. Оскільки змінні DIF_CVARS та DIF_NVARS відсутні у оброблюваному датасеті, їхні дані не будуть перезаписуватися, а завдяки оператору retain вони також не будуть очищуватися, а отже отримане кумулятивне значення (конкатенація) буде доступним до кінця виконання data-кроку з урахуванням всіх необхідних змін.

Зміна порядку змінних у вихідному датасеті досягається за допомогою оператору format за умови його виклику перед оператором set як показано нижче.

```
data &dsout_comp.;
  format _TYPE_ _OBS_
```

```
    %if "&id_vars." ^= "" %then %do;
      &id_vars.
    %end;
    %else %if "&usubjid_var." ^= "" %then %do;
      &usubjid_var.
    %end;
    %if "%upcase(&reorder.)" = "CHAR" %then %do;
      &dif_cvars.
    %end;
    %else %if "%upcase(&reorder.)" = "NUM" %then %do;
      &dif_nvars.
    %end;
  ;
  set &dsout_comp.;
run;
```

Варіативність порядку змінних в залежності від значення параметру reorder, наявності ключових змінних в параметрі id_vars та наявності змінної USUBJID реалізовано за допомогою умовного макрооператора %if-%then-%else. Відмінність від звичайного умовного оператору if-then-else полягає тільки в тому, що вони виконуються на етапі макрообробки, тобто перед виконанням даного data-кроку. Фактично, відбувається так звана макропідстановка значень в залежності від умов, яка модифікую код, що буде виконуватися. Наприклад, якщо макрос було викликано зі значенням «STUDYID USUBJID AVISIT» (без лапок) у параметрі id_vars, значенням «work.comp_adeg» (без лапок) у параметрі dsout_comp, значенням «CHAR» параметру reorder, та в процесі роботи макросу виявлено, що порівнювані датасети

мають відмінності у змінних PARAMCD, AVALC та DTYPE, код, що буде виконано виглядатиме наступним чином:

```
data work.comp_adeg;
  format _TYPE_ _OBS_ STUDYID USUBJID AVISIT PARAMCD AVALC DTYPE;
  set work.comp_adeg;
run;
```

Фінальною частиною макросу є звітування до логу виконання програми щодо результатів перевірки.

```
%if &comp_nobs. ^= 0 %then %put ERR%str(OR:) [_VALIDATE_ macro]:
Production (&dsin_prod.) and validation (&dsin_val.) datasets are not
equal! See &dsout_comp. for details.;
%else %put NOTE: [_VALIDATE_ macro]: Production (&dsin_prod.) and
validation (&dsin_val.) datasets match by common variables.;
```

Якщо відмінності знайдено, то до логу програми буде записано помилку з відповідним текстом пояснення, в протилежному випадку – примітку (NOTE), що дані датасетів є ідентичними з урахуванням спільних наявних змінних. Присутність змінних тільки у одному з двох порівнюваних датасетів та відмінності у метаданих змінних звітуються окремо.

2.4 Перевірка лейблів датасетів

Всі датасети (SDTM та ADaM) клінічних досліджень повинні мати лейбл. Лейбл є частиною метаданих датасету і фактично являє собою його назву. Всі передбачені CDISC-стандартами SDTM-датасети має визначені цим стандартом лейбли, в той час

як для більшості ADaM-датасетів лейбл можна змінювати на розсуд програміста або відповідно до вимог внутрішніх стандартів інституції, що займається їхнім програмуванням.

Оскільки наявність лейблу в датасеті є обов'язковою вимогою у переважній більшості випадків, перевірка їхньої наявності та правильності є частиною роботи QC-програміста (QC – Quality Control), тобто програміста, який виконує порівняння власного датасету з датасетом, що створений іншим програмістом.

Перевірку лейблів можна виконувати за допомогою стандартної процедури SAS `proc compare` чи макросом на її основі, наприклад таким, що наведено у розділі 2.3. Недоліком такого підходу є те, що відмінність у лейблі буде показано тільки у звіті вищезазначеної процедури. Це, в свою чергу, вимагає виконання перевірки лейблів очами, що, вочевидь, не є практичним.

З метою автоматизації перевірки лейблів датасетів, а також зменшення впливу людського фактору (перевірка очима) розроблено простий макрос `_label_check_`, що приймає на вхід два параметри – `dsin_prod` та `dsin_val`, через які до макрообробки передаються назви потрібних датасетів. Повний код макросу наведено в додатку 2. Далі буде наведено скорочений опис його дії.

```
%if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
  %put ERR%str(OR:) [_LABEL_CHECK_ macro]: Production and/or validation
datasets were not specified. Exiting macro.;
  %return;
%end;

%local prod_lbl val_lbl;
```

Як і випадку попереднього макросу (_validate_), відповідно до GPP виконується перевірка значень параметрів з якими було виконано виклик макросу та ініціалізація внутрішніх макрозмінних у локальній таблиці символів.

Метадані датасетів зчитуються за допомогою SAS-процедури proc contents.

```
proc contents data = &dsin_prod. out = prod_metadata (keep = MEMLABEL)
noprint;
run;

proc contents data = &dsin_val. out = val_metadata (keep = MEMLABEL)
noprint;
run;
```

Виклик процедури відбувається із зазначенням опції noprint, що пригнічує вивід метаданих датасетів до стандартного звіту. В даному випадку в цьому звіті немає потреби, а на додаток його вміст можна перевірити тільки вручну (без застосування програмних засобів). Тому результати запиту записуються до тимчасових датасетів (prod_metadata та val_metadata), ім’я яких вказане в out =. Зайві відомості про метадані датасету прибираються фільтром keep =.

```
proc sql noprint;
  select distinct MEMLABEL
    into :prod_lbl trimmed
    from prod_metadata
  ;
```

```
  select distinct MEMLABEL
    into :val_lbl trimmed
    from val_metadata
  ;
  drop table prod_metadata, val_metadata;
quit;
```

Лейбли датасетів, що були записані до тимчасових датасетів prod_metadata та val_metadata, переносяться до макрозмінних prod_lbl та val_lbl, після чого датасети видаляються. З технічної точки зору, можна було б об’єднати датасети prod_metadata та val_metadata у один та порівняти змінні, що містять лейбли датасетів. Однак при написанні макросу малася на увазі ідея можливості розширення його функціоналу та використання отриманих лейблів за межами коду макросу з метою, відмінною від просто перевірки. Тому реалізацію виконано за використанням макрозмінних.

Кінцевим етапом перевірки є порівняння лейблів, що записані у макрозмінні prod_lbl та val_lbl між собою

```
%if "&prod_lbl." ^= "&val_lbl." %then %do;
  %put ERR%str(OR:) [_LABEL_CHECK_ macro]: Production and validation
datasets have different labels.;
  %put ERR%str(OR-) [_LABEL_CHECK_ macro]: Production dataset label is
"&prod_lbl.".;
  %put ERR%str(OR-) [_LABEL_CHECK_ macro]: Validation dataset label is
"&val_lbl.".;
%end;
%else %put NOTE: [_LABEL_CHECK_ macro]: Production and validation
datasets have same labels.;
```

В залежності від результату порівняння, до логу виконання програми буде виведено або помилку, або примітку (NOTE) з описом цього результату. Якщо лейбли порівнюваних датасетів відмінні, то повідомлення у лозі буде мати формат помилки, також буде наведено фактичні лейбли датасетів. Приклад повідомлень до логу наведено на рис. 2.3.

```
ERROR: [_LABEL_CHECK_ macro]: Production and validation dataset labels are not equal.
       [_LABEL_CHECK_ macro]: Production dataset label is "".
       [_LABEL_CHECK_ macro]: Validation dataset label is "Subject-Level Analysis Dataset".
```

Рисунок 2.3 – Приклад повідомлень з виявленими помилками у лозі програми, що згенерований макросом `_label_check_`

2.5 Перевірка сортування датасетів

Кожен датасет клінічних досліджень сортується за унікальним «натуральним» ключем. Ключ складається з переліку змінних датасету, за значеннями яких можна однозначно визначити запис у датасеті [9]. Іншими словами, комбінація значень ключових змінних властива тільки одному єдиному запису датасету. Перевірити за якими змінними відсортовано датасет можна використовуючи процедуру `proc contents`. Недоліком такого підходу є те, що результатом виклику процедури буде звіт, перевіряти який програміст вимушений «очима». Це, щонайменше, є незручним, призводить до додаткових витрат часу, а також зазнає впливу людського фактору. Автоматизація цього процесу нівелює перераховані недоліки.

Макрос для перевірки та порівняння сортування датасетів, як і у попередніх випадках (макроси _validate_ та _label_check_) приймає два параметри, dsin_prod та dsin_val, через які до макросу передаються назви порівнюваних датасетів. Назва макросу – _sort_check_, а повний код наведено у додатку 3.

На початку виконання макросу виконується перевірка значень, що передано через параметри та ініціалізація внутрішніх макрозмінних в локальній таблиці символів (детальніше у розд. 2.3 та 2.4).

```
%if "&dsin_prod." = "" or "&val_ds." = "" %then %do;
  %put ERR%str(OR:) [_SORT_CHECK_ macro]: Production and/or validation
datasets were not specified. Exiting macro. Exiting macro.;
  %return;
%end;

%local prod_is_sorted val_is_sorted prod_sort_order val_sort_order
_dummy_;
```

Аналогічним до макросу _label_check_ (розд. 2.4) чином виконується зчитування метаданих датасетів за допомогою процедури proc contents. Результат читання зберігається до тимчасових датасетів prod_metadata та val_metadata. Різниця полягає тільки в тому, що в даному випадку необхідними даними для подальшої обробки є не лейбл датасету, а його ім’я, перелік змінних сортування та чи є цей перелік валідованим системою SAS.

```
proc contents data = &dsin_prod. out = prod_metadata (keep = NAME SORTED
SORTEDBY) noprint;
run;

proc contents data = &dsin_val. out = val_metadata (keep = NAME SORTED
SORTEDBY) noprint;
run;
```

Додатковою перевагою використання макросу є те, що він виконує перевірку не тільки змінних, за якими датасет відсортовано, а й те, чи був він фактично відсортований за вказаними змінними, тобто валідацію сортування. З точки зору розміщення даних різниці не буде, якщо тільки не було спроб навмисно ввести в оману.

Пояснення щодо різниці, чому це є важливим наступне. Датасет можна створити багатьма способами. Наприклад, виводити записи до нього вручну (явне використання оператору output) і таким чином можна досягти того, що записи в датасетів будуть розташовані відповідно до ключа без фактичного сортування датасету. Після цього, можна використати опцію data-кроку sortedby = та вказати перелік змінних ключа. В такому випадку процедура proc contents покаже, що записи датасету відсортовано за зазначеними в sortedby = змінними.

При цьому може статися, наприклад, помилка програмування, та фактичний порядок записів не буде відповідати переліченим змінним ключа. І якщо такий датасет потім використовується у процедурах, що передбачають обов’язкове попереднє сортування, з’являться помилки виконання, але вже у програмах користувачів датасету, а це

неприпустимо. Власне для цього й існує вимога сортування фінальних версій датасетів, а до запропонованого макросу додано відповідну перевірку.

Як слідує з коду нижче, дані про валідність сортування перевіряються за допомогою SAS-процедури `proc sql` та зберігаються у внутрішніх макрозмінних `prod_is_sorted` та `val_is_sorted`. Запит переліку змінних ключа сортування відбувається також за допомогою `proc sql`, але тільки у випадку, якщо є підтвердження того, що сортування є валідним. Тимчасові датасети з метаданими в кінці цього фрагменту знищуються за ненадібністю.

```
proc sql noprint;
  select distinct SORTED
    into :prod_is_sorted trimmed
    from prod_metadata
  ;
  select distinct SORTED
    into :val_is_sorted trimmed
    from val_metadata
  ;

  %if &prod_is_sorted. = 1 %then %do;
    select distinct NAME, SORTEDBY
      into :prod_sort_order separated by ', ', :_dummy_ separated by ', '
      from prod_metadata
      where ^missing(SORTEDBY)
      order by SORTEDBY
    ;
  %end;
  %if &val_is_sorted. = 1 %then %do;
```

```
    select distinct NAME, SORTEDBY
      into :val_sort_order separated by ', ', :_dummy_ separated by ', '
      from val_metadata
      where ^missing(SORTEDBY)
      order by SORTEDBY
    ;
  %end;

  drop table prod_metadata, val_metadata;
quit;
```

Імена змінних ключа у датасетах згенерованих `proc contents` розташовані всередині однієї змінної `NAME` тому відбувається конкатенація всіх значень цієї змінної до значення макрозмінної `prod_sort_order` (або `val_sort_order`), де роздільником є пробіл. Слід також зазначити, що послідовність визначається значенням змінної `SORTEDBY`, відповідно до якого й формується `prod_sort_order` або `val_sort_order`.

Останній фрагмент макросу відповідає за звітування результатів перевірки, що виконано макросом. Якщо виявлено, що порядок сортування двох порівнюваних датасетів однаковий, а також валідний, до логу виконання програми буде виведено примітку (NOTE). В протилежному випадку буде виведено помилку, а також буде наведено за якими змінними відсортовано кожен з порівнюваних датасетів. Приклад повідомлень до логу наведено на рис. 2.4.

```
ERROR: [_SORT_CHECK_ macro]: Production and validation datasets are sorted differently:
       [_SORT_CHECK_ macro]: Production dataset is sorted by STUDYID, USUBJID.
       [_SORT_CHECK_ macro]: Validation dataset is not sorted.
```

Рисунок 2.4 – Приклад повідомлень з виявленими помилками у лозі програми, що згенерований макросом _sort_check_

```
%if "&prod_sort_order." = "&val_sort_order." %then %do;
  %put NOTE: [_SORT_CHECK_ macro]: Production and validation datasets
have the same sorting order.;
%end;
%else %do;
  %put ERR%str(OR:) [_SORT_CHECK_ macro]: Production and validation
datasets are sorted differently:;
  %if &prod_is_sorted. = 1 %then %put ERR%str(OR-) [_SORT_CHECK_ macro]:
Production dataset is sorted by &prod_sort_order..;
  %else %put ERR%str(OR-) [_SORT_CHECK_ macro]: Production dataset is not
sorted.;
  %if &val_is_sorted.  = 1 %then %put ERR%str(OR-) [_SORT_CHECK_ macro]:
Validation dataset is sorted by &val_sort_order..;
  %else %put ERR%str(OR-) [_SORT_CHECK_ macro]: Validation dataset is not
sorted.;
%end;
```

2.6 Порівняння порядку розташування змінних всередині датасетів

Порядок змінних чітко не регламентується стандартами і технічно може бути майже будь-яким. Але в посібнику (IG, Implementation Guide) надаються певні рекомендації. По-перше, бажано, щоб порядок змінних відповідав порядку, порядку змінних в IG, а по-друге, треба щоб послідовність змінних була

однаковою для всіх датасетів всередині одного клінічного дослідження. Останнє, втім, можна вважати вимогою.

Для порівняння послідовностей змінних у двох датасетів створено макрос _var_order_check_ (повний код див. у додатку 4) який має три параметри. Як і у випадку макросу _validate_ (розд. 2.3) це dsin_prod, dsin_val та dsout_comp. Перші два параметри передають макросу назви датасетів для порівняння, а третій параметр вказує ім’я датасету, до якого буде виведено детальну інформацію щодо результатів порівняння.

Як і у випадку попередньо описаних макросів, з метою дотримання принципів GPP, спершу виконується перевірка значень параметрів та локальна ініціалізація макрозмінних. Детальніше див. розд. 2.3.

```
%if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
  %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]: Production and/or
validation datasets were not specified. Exiting macro.;
  %return;
%end;
%if "&dsout_comp." = "" %then %do;
  %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]: Comparison results dataset
was not specified. Exiting macro.;
  %return;
%end;

%local prod_var_cnt val_var_cnt sort_var diff_st_name diff_st_pos
diff_found;
```

Метадані датасетів, до яких входить порядок змінних, зчитується за допомогою процедури proc contents у тимчасові датасети prod_metadata та val_metadata.

```
proc contents data = &dsin_prod. out = prod_metadata (keep = NAME VARNUM
LABEL) noprint;
run;

proc contents data = &dsin_val. out = val_metadata (keep = NAME VARNUM
LABEL) noprint;
run;
```

Отримані за допомогою proc contents датасети мають кількість записів, що дорівнює кількості змінних порівнюваних датасетів. Кількість записів зчитується data-кроком та записується до макрозмінних prod_var_cnt та val_var_cnt.

```
data _null_;
  call symputx("prod_var_cnt", NOBS);
  if 0 then set prod_metadata nobs = NOBS;
  stop;
run;

data _null_;
  call symputx("val_var_cnt", NOBS);
  if 0 then set val_metadata nobs = NOBS;
  stop;
run;
```

Якщо виявляється, що хоча б один з порівнюваних датасетів не має змінних (кількість записів тимчасових датасетів

prod_metadata або val_metadata дорівнює нулю), подальше виконання макросу не має сенсу, бо відсутні дані для порівняння. Тому до логу виконання програми виводиться повідомлення-помилка із зазначенням причини помилки, а робота макросу зупиняється. Код макросу, що є відповідальним за це, наведено нижче.

```
%if &prod_var_cnt. = 0 %then %do;
  %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]: Production dataset has no variables. Exiting macro.;
  %return;
%end;
%if &val_var_cnt. = 0 %then %do;
  %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]: Validation dataset has no variables. Exiting macro.;
  %return;
%end;
```

Для подальшої обробки та створення вихідного датасету-звіту необхідно об’єднати тимчасові датасети, що створені при виконанні процедури proc contents. Ключом об’єднання буде змінна NAME, що містить назви змінних порівнюваних датасетів. Об’єднання буде виконано за допомогою оператора merge data-кроку, але для цього є необхідним щоб обидва об’єднувані датасети були відсортовані по цій змінній NAME.

```
proc sort data = prod_metadata;
  by NAME;
run;
```

```
proc sort data = val_metadata;
  by NAME;
run;
```

В результаті об’єднання датасетів prod_metatdata та val_metadata за ключем NAME (оператор by) створюється попередня версія вихідного датасету. Оскільки вхідні датасети мають однакові назви змінних (VARNUM та LABEL), перед об’єднанням ці змінні необхідно перейменувати. В такому випадку у вихідному датасеті буде чотири змінних (VORD_P, VLABEL_P, VORD_V та VLABEL_V). Якщо перейменування не буде виконане, вихідний датасет матиме тільки дві (окрім ключової NAME) змінні VARNUM та LABEL. При чому значення змінних з датасету prod_metatdata буде перезаписане значеннями відповідних змінних з датасету val_metadata (за умови співпадіння записів за ключем).

З метою визначення чи є присутнім запис в кожному з вхідних датасетів використовується опція in = оператору merge. В результаті створюються тимчасові (автоматично будуть видалені з вихідного датасету) змінні in_PROD та in_VAL. Якщо запис присутній у обох вхідних датасетах, обидві змінні матимуть значення 1 для цього запису. Якщо запис присутній тільки в одному з датасетів – тільки одній з цих змінних буде присвоєне значення 1, друга лишиться пустою (missing в імплементації SAS, для числових змінних це символ крапки).

Результат перевірки походження записів записується до символьної змінної PRES (від presence): «BOTH» – запис є в обох датасетах, «PROD» – запис є тільки в першому датасеті, «VAL» – запис є тільки в другому (валідаційному) датасеті. Значення

«MACRO ERROR» є потенційно неможливим. Створене для відстежування помилок виконання макросу. Порядок змінних створюваного вихідного датасету визначається переліком змінних, що слідують за оператором format, що передує оператору set.

```
data &dsout_comp.;
  format PRES NAME VORD_P VORD_V VLABEL_P VLABEL_V;
  length PRES $20;
  merge prod_metadata (in = in_PROD rename = (VARNUM = VORD_P LABEL =
VLABEL_P))
        val_metadata  (in = in_VAL  rename = (VARNUM = VORD_V LABEL =
VLABEL_V));
  by NAME;
  if in_PROD and in_VAL then PRES = 'BOTH';
  else if in_PROD       then PRES = 'PROD';
  else if in_VAL        then PRES = 'VAL';
                        else PRES = 'MACRO ERROR';
run;
```

Вихідний датасет з результатами порівняння буде відсортовано за порядком змінних одного з вхідних датасетів. Логічним сортуванням було б таке, що відповідає порядку змінних у датасеті який перевіряється. Але можлива ситуація, коли цей датасет матиме меншу кількість змінних ніж валідаційний датасет. В такому випадку у змінній VORD_P будуть відсутні значення для записів, джерелом яких є валідаційний датасет, тобто відсутній порядковий номер змінної, якщо немає самої змінної. Сортування по VORD_P призведе до того, що строки з назвами змінних, які є відсутніми в датасеті, що перевіряються будуть завжди першими

(для SAS відсутність значення, або missing, є завжди меншим ніж будь-яке непусте значення). Таке сортування не є логічним. Тому виконується перевірка на кількість змінних в кожному з порівнюваних датасетів. Якщо їхня кількість однакова або датасет, що перевіряється має більшу кількість змінних, то вихідний датасет буде відсортовано відповідно до порядку змінних саме в цьому датасеті (за змінною VORD_P). Інакше вихідний датасет матиме сортування, що відповідає порядку змінних у валідаційному датасеті (за змінною VORD_V).

Ім’я змінної за якою буде відсортовано вихідний датасет записується до макрозмінної sort_var після перевірки умовним макрооператором %if-%then-%else.

```
proc sql noprint;
  select max(VARNUM)
    into :prod_var_cnt trimmed
    from prod_metadata
  ;
  select max(VARNUM)
    into :val_var_cnt trimmed
    from val_metadata
  ;
quit;

%if &prod_var_cnt. >= &val_var_cnt. %then %let sort_var = VORD_P;
                                    %else %let sort_var = VORD_V;
```

Вихідний датасет створюється на основі попередньо створеного за використанням data-кроку – назва вхідного (після оператору set) та вихідного (в шапці data-кроку) датасетів тут

однакові та відповідають тому, що записано до макрозмінної dsout_comp). Результат порівняння порядкового номеру змінної у вхідних датасетах (змінні VORD_P та VORD_V) записується до змінної ORD_MATCH: 1 коли порядок співпадає, 0 – коли ні.

На цьому data-кроці порядок змінних вихідного датасету задається за допомогою оператора format, довжини та типи змінних датасету ініціалізуються оператором length (наявність символу «$» перед довжиною ініціалізує змінну як символьну), а лейбли змінних задаються, відповідно, оператором label.

```
proc sort data = &dsout_comp.;
  by &sort_var.;
run;

data &dsout_comp. (drop = prev_: DIFF_FOUND);
  format PRES ORD_MATCH NAME VORD_P VORD_V VLABEL_P VLABEL_V;
  length ORD_MATCH prev_VORD DIFF_FOUND 8 prev_VNAME $200;
  set &dsout_comp.;
  retain ORD_MATCH DIFF_FOUND 0 prev_VNAME prev_VORD;
  label PRES      = 'Presence in Datasets';
  label ORD_MATCH = 'Variable Order Is Matching';
  label NAME      = 'Variable Name';
  label VORD_P    = 'Variable Order in Production Dataset';
  label VORD_V    = 'Variable Order in Validation Dataset';
  label VLABEL_P  = 'Variable Label in Production Dataset';
  label VLABEL_V  = 'Variable Label in Validation Dataset';
  if VORD_P = VORD_V then ORD_MATCH = 1;
                     else ORD_MATCH = 0;
  if ORD_MATCH and ^DIFF_FOUND then do;
    prev_VNAME = NAME;
    prev_VORD  = &sort_var.;
    call symputx("diff_st_name", NAME);
```

```
    call symputx("diff_st_pos",  &sort_var.);
  end;
  else do;
    DIFF_FOUND = 1;
    call symputx("diff_found", 1);
  end;
run;
```

На вищезазначеному data-кроці, окрім створення вихідного датасету для звітності, також виконується розрахунок макрозмінних, які необхідні для виведення повідомлень до логу виконання програми. Коли знайдено невідповідність у порядку змінних, до логу буде виводитися ім’я останньої змінної, порядок якої все ще співпадає в порівнювальних датасетах. Очевидно, що ім’я цієї змінної знаходиться у записі, який передує поточному (тому, в якому вперше знайдено невідповідність). Оскільки механізм читання записів датасетів у SAS послідовний – при читанні даних поточного запису дані з попередніх стають недоступними – використано оператор `retain` (детальніше див. розд. 2.3), таким чином порядковий номер змінної та її назва з попереднього запису зберігається у змінних `prev_VNAME` та `prev_VORD`, значення яких оновлюються тільки до тих пір, поки не буде знайдено невідповідність.

Дані щодо останньої змінної де її положення однакове в обох вхідних датасетах зберігаються у макрозмінних `diff_st_name` та `diff_st_pos`, а факт того, що знайдено невідповідність – у макрозмінній-індикаторі `diff_found`. Ці макрозмінні використовуються для формування та виведення повідомлень до логу програми, код чого наведено нижче.

```
%if &diff_found. ^= 1 %then %put NOTE: [_VAR_ORDER_CHECK_ macro]:
Production and validation datasets have the same variable order.;
%else %do;
  %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]: Production and validation
datasets have different variable order.;
  %put ERR%str(OR-) [_VAR_ORDER_CHECK_ macro]: The difference starts from
the variable &diff_st_name. at position &diff_st_pos..;
%end;

proc sql noprint;
  drop table prod_metadata, val_metadata;
quit;
```

Датасети, створені у результаті виклику процедури proc contents знищуються на фінальному етапі роботи макросу за ненадібністю (та для вивільнення ресурсів).

У випадках коли відмінностей у порядку змінних порівнюваних датасетів не знайдено, до логу програми виводиться примітка (NOTE), в противному випадку – помилка із зазначенням назви та положення змінної, що співпадає. Приклад повідомлень до логу наведено на рис. 2.5.

```
ERROR: [QC_RU_CHK_VORD macro]: Production and validation datasets have different variable order.
       [QC_RU_CHK_VORD macro]: The difference starts from the variable SITEID at position 3.
```

Рисунок 2.5 – Приклад повідомлень з виявленими помилками у лозі програми, що згенерований макросом _var_order_check_

Приклад результату роботи макросу наведено на рис. 2.6. Видно, що останньою змінною, розташування якої є однаковим в порівнювальних датасетах, є SITEID, як і зазначено у лозі на

рис. 2.5, а відмінності починаються зі змінної AGE, про що свідчить значення змінної датасету ORD_MATCH = 0. Порядковий номер змінної AGE у першому з порівнюваних датасетів VORD_P = 4, а у другому – VORD_V = 7.

	PRES	ORD_MATCH	NAME	VORD_P	VORD_V	VLABEL_P	VLABEL_V
1	BOTH	1	STUDYID	1	1	Study Identifier	Study Identifier
2	BOTH	1	INVID	2	2	Investigator Identifier	Investigator Identifier
3	BOTH	1	SITEID	3	3	Study Site Identifier	Study Site Identifier
4	BOTH	0	AGE	4	7	Age	Age
5	BOTH	0	AGEU	5	8	Age Units	Age Units
6	BOTH	0	USUBJID	6	4	Unique Subject Identifier	'
7	BOTH	0	COUNTRY	7	5	Country	Country
8	BOTH	0	ACOUNTRY	8	6	Analysis Country	Analysis Country
9	BOTH	1	AGEGR1	9	9	Pooled Age Group 1	Pooled Age Group 1
10	BOTH	1	AGEGR1N	10	10	Pooled Age Group 1 (N)	Pooled Age Group 1 (N)
11	BOTH	1	SEX	11	11	Sex	Sex
12	BOTH	1	ETHNIC	12	12	Ethnicity	Ethnicity
13	BOTH	1	RACE	13	13	Race	Race
14	BOTH	1	SCRNFL	14	14	Screening Population Flag	Screening Population Flag
15	BOTH	1	ENRLFL	15	15	Enrolled Population Flag	Enrolled Population Flag
16	BOTH	1	ITTFL	16	16	Intent-To-Treat Population Flag	Intent-To-Treat Population Flag
17	BOTH	1	SAFFL	17	17	Safety Population Flag	Safety Population Flag
18	BOTH	1	ARM	18	18	Description of Planned Arm	Description of Planned Arm
19	BOTH	1	ARMCD	19	19	Planned Arm Code	Planned Arm Code
20	BOTH	1	ACTARM	20	20	Description of Actual Arm	Description of Actual Arm
21	BOTH	1	ACTARMCD	21	21	Actual Arm Code	Actual Arm Code

Рисунок 2.6 – Фрагмент датасету із порівнянням порядку змінних, що згенерований макросом _var_order_check_

2.7 Мінімізація довжин символьних змінних

Довжини змінних мають декілька обмежень. Першим є обмеження, що накладається SAS. Так, для числових змінних можливий діапазон довжин є від 3 до 8 байт, а для символьних – від 1 до 32767 байт. Додатковим обмеженням є використання так званого транспортного протоколу SAS версії 5, використання якого є обов’язковим в області клінічних досліджень. Протокол використовується для обміну даними між інституцією що їх

створила (фармацевтична компанія, CRO тощо) та регуляторним органом (FDA, PMDA тощо). Транспортний протокол обмежує максимальну довжину змінних до 200 байт, а деяких змінних (--TESTCD чи IDVAR в SDTM-датасетах) до 40 байт, бо значення цих змінних буде потенційно використане в якості лейблів змінних при транспонуванні датасетів [9, 13]. Останнє не є важливим з точки зору реалізації описуваного макросу, тому що перевірка значень таких змінних відбувається іншим чином окремо.

Мінімізація довжин символьних змінних є необхідною, а подекуди обов'язковою, коли дані клінічних досліджень зібрані та оброблені та надсилаються до регулятора. В такому випадку процедура виконується один раз в самому кінці розробки датасетів.

Однак складнощі виникають в процесі валідації датасетів, коли програмісти не дотримуються якихось певних домовленостей щодо довжин змінних або, що є справедливим для більшості випадків, таких домовленостей не існує. Наприклад, для значення довжиною 100 символів первинний програміст використовує довжину змінної 200, а програміст-валідатор – якусь іншу, відмінну від 200. Без домовленостей, обидві реалізації є коректними, але порівняння датасетів за допомогою стандартної SAS-процедури proc compare покаже, що датасети не є однаковими (з точки зору метаданих). В такому випадку, використання макросу, що пропонується, обома програмістами вирішує проблему необхідності домовленостей щодо довжин змінних – програмісти можуть не замислюватися щодо встановлення довжин змінних та використовувати таку, що є зручною для них. Слід відмітити, що вищезазначене, а саме вибір

довжини одним з програмістів, є вірним тільки якщо встановлена довжина більшою за максимальне значення змінної. В противному випадку це буде помилкою.

Розроблений макрос `_len_minify_` може використовуватися не тільки на боці валідації, але й на боці первинного програміста. Проте, це не буде коректним з точки зору концепції подвійного програмування – макрос слід використовувати тільки одним з програмістів, інший програміст повинен використовувати інший макрос, аналогічний за функціональністю розробленому.

Макрос `_len_minify_` має чотири вхідних параметри: `dsin`, `dsout`, `debug_dsout` та `debug`. Через параметри `dsin` та `dsout` вказуються відповідно назви вхідного датасету, довжини змінних якого треба мінімізувати, та вихідного датасету – це той самий датасет, але з вже мінімізованими довжинами. Параметр `dsin`, вочевидь, є обов’язковим. За відсутності значення в цьому параметрі, макрос виведе помилку до логу виконання програми та припинить подальше своє виконання. У випадку відсутності значення у параметрі `dsout` макрос не буде створювати додатковий датасет з мінімізованими змінними, а перезапише вхідний датасет новим, тобто оновить датасет, що вказано в параметрі `dsin`.

Через параметр `debug_dsout` визначається назва датасету, що міститиме дані з аргументацією прийнятих макросом рішень, такі як назва символьної змінної, її нова, мінімізована, довжина та максимально довге знайдене значення відповідної змінної, за яким власне й було встановлено нову довжину. Створення зазначеного датасету відбувається тільки у випадку, коли через параметр `debug` передане будь-яке непусте значення, наприклад

коли debug = Y. Необхідно зазначити, що такий датасет створюється в будь-якому випадку, він є необхідним для внутрішньої обробки, тому фактично в залежності від значення параметру debug датасет стирається або лишається.

Робота макросу _len_minify_, як і у випадку попередньо описаних макросів, починається з перевірки значень вхідних параметрів на наявність значень (тільки dsin) та ініціалізації макрозмінних у локальній таблиці символів. Параметри dsout та debug_dsout в даному випадку функціонують як локальні макрозмінні. Якщо значення для параметру dsout не задано, то в нього буде записане значення з dsin який є обов'язковим і тому бути пустим не може. У разі відсутності значення для параметру debug_dsout до нього буде також записане значення з параметру dsin , але з суфіксом _varlen. Тобто якщо вхідний датасет має назву input_data, то створений «діагностичний» датасет матиме назву input_data_varlen і буде розташований в тій самій бібліотеці, що і датасет input_data.

```
%if "&dsin." = "" %then %do;
  %put ERR%str(OR:) [_LEN_MINIFY_ macro]: Input dataset was not
specified. Exiting macro.;
  %return;
%end;

%local curr_lib curr_ds src_var_list var_cnt curr_var out_var_names
out_var_lengths;

%if "&dsout."       = "" %then %let dsout       = &dsin.;
%if "&debug_dsout." = "" %then %let debug_dsout = &dsin._varlen;
```

Після перевірки вхідних значень параметрів та ініціалізації макрозмінних відбувається перевірка назви бібліотеки, в якій розташовано вхідний датасет, що необхідно для подальшої обробки. В SAS назва бібліотеки та датасету розділяються крапкою, тобто mylib.mydata означає, що датасет з назвою mydata розташований в бібліотеці mylib. Необхідно вточнити, що якщо назва бібліотеки (токен перед крапкою) не вказаний, то бібліотекою розташування датасету є внутрішня SAS-бібліотека work. Таким чином записи maydata та work.mydata є рівнозначними.

```
%if %sysfunc(countw(&dsin., %str(.))) = 2 %then %do;
  %let curr_lib = %upcase(%scan(&dsin., 1, %str(.)));
  %let curr_ds  = %upcase(%scan(&dsin., 2, %str(.)));
%end;
%else %do;
  %let curr_lib = WORK;
  %let curr_ds  = %upcase(&dsin.);
%end;
```

Для розбиття значення параметру dsin на назву бібліотеки та датасету використовується оператор countw, що викликаний через макрооператор %sysfunc (необхідно в зв’язку відсутністю макроаналога оператора countw). Оператор countw рахує кількість «слів» у рядку, що розділено певним роздільником. В даному випадку роздільником є крапка. Якщо цих слів у параметрі dsin два, тобто датасет знаходиться не в бібліотеці work або використано формат work.mydata, то за використанням макрооператора %scan перше слово (бібліотека) із значення параметру dsin записується до макрозмінної curr_lib, а друге

(датасет) – до макрозмінної curr_ds. Якщо слово у параметрі dsin лише одне, що вказує на неявне посилання на бібліотеку work, до макрозмінних curr_lib та curr_ds відповідно записуються значення work та те, що міститься у параметрі dsin. Значення, що записуються до curr_lib та curr_ds переводяться до верхнього регістру, це необхідно для подальшої обробки, де є чутливість до регістру.

```
proc sql noprint;
  select NAME
    into :src_var_list separated by ' '
    from dictionary.columns
    where upcase(LIBNAME) = "&curr_lib." and upcase(MEMTYPE) = 'DATA' and
upcase(MEMNAME) = "&curr_ds." and upcase(TYPE) = 'CHAR'
  ;
quit;

%let var_cnt = %sysfunc(countw(&src_var_list., %str( )));
```

Подальша обробка довжин змінних відбуватиметься всередині %do-%end циклічного макрооператора, тому перед цим необхідно зробити перелік всіх символьних змінних вхідного датасету та кількість таких змінних. Метадані всіх датасетів та змінних, що всередині них, які знаходяться в підключених SAS-бібліотеках датасетів (визначення посилань на бібліотеки не є частиною роботи макросу) знаходяться в так званих SAS-словниках. Ці словники фактично є спеціалізованою бібліотекою, що має назву dictionary. Таким чином, одним з найпростіших засобів отримання необхідних кількості та переліку символьних змінних певного датасету є читання даних

відповідних датасетів бібліотеки dictionary, в даному випадку це датасет columns.

Безпосередньо читання назв змінних виконується процедурою proc sql, а їхній перелік записується до макрозмінної src_var_list. У якості роздільника між елементами переліку використовується пробіл (опція separated by оператора into). При читанні, дані фільтруються за допомогою оператора where – вказується назва бібліотеки, назва датасету та тип змінних. Коли перелік збережено до макрозмінної src_var_list, кількість його елементів отримується оператором countw, що викликаний через макрооператор %sysfunc, а результат записується до макрозмінної var_cnt.

Подальший код для простоти пояснення розбито на частини – окремі SQL-запити, але всі ці запити розташовані всередині одного виклику SAS-процедури proc sql (первинний вигляд зазначеного фрагменту коду можна подивитися у додатку 5).

Наведений нижче SQL-запит порядково створює датасет з назвою за значенням у debug_dsout (далі для спрощення просто датасет debug_dsout) шляхом перебору переліку оброблюваних символьних змінних, що міститься у макрозмінній src_var_list. Назва поточної оброблюваної змінної визначається макрооператором %scan та записується до макрозмінної curr_var. Це не є обов’язковим, але спрощує подальший код, бо ім’я отриманої в такий спосіб змінної використовуватиметься всередині %do-%end циклу декілька разів.

Кожен рядок створюваного датасету матиме три змінні, це VAR_NAM, VAR_LEN та MAX_VAL з лейблами «Variable Name», «Maximum Length» та «The Longest Content» відповідно. До лейблу першої

змінної також додається ім’я вхідного датасету (параметр dsin). Рядки додаються до створюваного датасету оператором outer union до тих пір, поки не буде пройдено весь перелік символьних змінних, що міститься у макрозмінній src_var_list. За це відповідає макрооператор %if-%then, що перевіряє умову, чи не стало значення лічильника i рівним або більшим за значення макрозмінної var_cnt яка, в свою чергу, містить кількість елементів переліку у src_var_list.

Вибір максимального за довжиною значення оброблюваної змінної виконується шляхом використання оператору having з відповідною умовою (функція max). Таким чином, у створюваному датасеті debug_dsout кількість записів буде дорівнювати кількості символьних змінних у вхідному датасеті, що переданий через параметр dsin. У випадках коли змінна має декілька однакових за довжиною значень, обирається перше, що зустрілося. Таку унікальність вибору записів гарантує поєднання використання функцій monotonic та min в умові оператора having.

```
create table &debug_dsout. as
  %do i = 1 %to &var_cnt.;
    %let curr_var = %scan(&src_var_list., &i., %str( ));
    select distinct
      "&curr_var."       as VAR_NAM label = "&dsin. Variable Name",
      length(&curr_var.) as MAX_LEN label = 'Maximum Length',
      &curr_var.         as MAX_VAL label = 'The Longest Content'
    from &dsin.
    having MAX_LEN = max(MAX_LEN) and monotonic() = min(monotonic())
    %if &i. < &var_cnt. %then %str(outer union corr);
  %end;
  order by VAR_NAM
```

```
;
```

Отримані пари значень «назва змінної» у VAR_NAM та «максимально довге її значення» у MAX_LEN читаються з датасету debug_dsout та у вигляді переліків (роздільник пробіл) записуються до макрозмінних out_var_names та out_var_lengths. Ці переліки необхідні для подальшого перебору всіх змінних та оновлення їхніх довжин всередині %do-%end циклу. На цьому етапі, за умови наявності значення вхідного параметру dsout, також відбувається створення вихідного датасету, що міститиме копію датасету з параметру dsin, але з оновленими довжинами змінних.

```
select VAR_NAM, MAX_LEN
  into :out_var_names   separated by ' ',
       :out_var_lengths separated by ' '
  from &debug_dsout.
;

%if &dsout. ^= &dsin. %then %do;
  create table &dsout. as
    select *
    from &dsin.
  ;
%end;
```

Фінальним етапом роботи макросу _len_minify_ є зміна довжин символьних змінних у датасеті, що є результатом роботи макросу в залежності від комбінації параметрів dsin та dsout, як описано раніше. Зміна довжини змінних виконується за допомогою оператору modify SQL-запиту alter, як показано

нижче. Тут обробка відбувається шляхом перебору створених вище переліків, що містять назви та нові довжини змінних (макрозмінні out_var_names та out_var_lengths), всередині %do-%end циклу. Елементи переліків вибираються, як і раніше, макрооператором %scan. Тобто на кожній ітерації %do-%end циклу беруться i-елементи кожного з переліків до тих пір, доки не будуть перебрані всі елементи, кількість яких зазначена у макрозмінній var_cnt.

Датасет debug_dsout знищується в залежності від значення вхідного параметру debug: стирання датасету відбувається тільки у випадку пустого значення параметру debug.

```
alter table &dsout.
  %do i = 1 %to &var_cnt.;
    modify %scan(&out_var_names., &i., %str( ))
char(%scan(&out_var_lengths., &i., %str( )))
  %end;
;
%if "&debug." = "" %then %do;
  drop table &debug_dsout.;
%end;
```

Використання SQL-запиту alter замість перестворення датасету з новими довжинами на data-кроці SAS, а також використання бібліотеки dictionary суттєво пришвидшує обробку даних макросом. Цей момент є принциповим при обробці, наприклад, датасетів клінічних досліджень, що містять лабораторні дані – кількість записів таких датасетів зазвичай становить десятки та сотні тисяч.

Приклад датасету, що містить дані, на основі яких макрос _len_minify_ оновлював довжини символьних змінних наведено на рис. 2.7. У змінній MAX_VAL міститься найдовше знайдене значення змінної, з назвою вказаною у VAR_NAM, для якої встановлено нову довжину, зазначену у MAX_LEN.

	VAR_NAM	MAX_LEN	MAX_VAL
1	ACOUNTRY	9	Argentina
2	ACTARM	7	Placebo
3	ACTARMCD	4	PLCB
4	AGEGR1	5	18-49
5	AGEU	4	AGEU
6	ARM	7	Placebo
7	ARMCD	4	PLCB
8	COUNTRY	3	ARG
9	ENRLFL	1	N
10	ETHNIC	22	NOT HISPANIC OR LATINO
11	INVID	4	1981
12	ITTFL	1	N
13	RACE	25	BLACK OR AFRICAN AMERICAN
14	SAFFL	1	N
15	SCRNFL	1	N
16	SEX	1	M
17	SITEID	6	999253
18	STUDYID	6	ABC123
19	USUBJID	13	ABC123.000001

Рисунок 2.7 – Фрагмент датасету-звіту, що згенерований макросом _len_minify_

2.8 Перевірка роботи розроблених SAS-макросів

Перевірка роботи розроблених SAS-макросів _validate_, _label_check_, _sort_check_, _var_order_check_ та _len_minify_ на

предмет вірності створюваних результатів відбувалася шляхом подвійного програмування, де другим ступенем були самі макроси. Тобто результати, які генерують макроси порівнювалися з отриманими шляхом звичайного програмування (open-code programming), яке виконано незалежним програмістом. Результати отримані незалежним програмістом не підлягали програмній обробці, тому порівняння відбувалося третьою особою-оглядачем (reviewer). Розбіжностей між результатами, що отримані незалежним програмістом та розробленими макросами не виявлено.

Макрос `_len_minify_` також перевірявся з точки зору швидкодії, де порівнювалася його робота із аналогом (аналог з точки зору кінцевого результату). В результаті перевірки виявлено, що розроблений макрос `_len_minify_` виконує ту ж саму задачу у 12 разів швидше, не зважаючи на можливість створення ним додаткових датасетів, призначенням яких є звітування роботи макросу та контроль виконання (прийняття рішень).

ВИСНОВКИ

Розроблено сукупність SAS-макросів, які дають змогу ефективно виявляти відмінності між даними та метаданими двох датасетів, що суттєво пришвидшує процес валідації даних клінічних досліджень на етапах створення SDTM-датасетів, ADaM-датасетів та датасетів що містять аналітичні дані TLF.

Автоматизовано процес валідації метаданих датасетів та звітування знайдених відмінностей у такому вигляді, що дає змогу обробляти ці звіти програмно, таким чином суттєво підвищуючи ефективність процесу валідації. Валідація метаданих включає в себе порівняння лейблів датасетів, ключів сортування, порядку змінних всередині датасетів, їхніх типів, довжин та лейблів.

Результат знайдених невідповідностей між двома датасетами виводиться до окремого датасету, що включає в себе записи, які є наявними тільки в одному з них, а також пари записів, що містять хоча б одну змінну, значення якої не співпадає. Впроваджено можливість виводу до датасету, в першу чергу, тих змінних, що мають відмінності. Сукупність перелічених підходів робить роботу програміста-валідатора значно простішою, та в рази пришвидшує процес пошуку невідповідностей між двома датасетами.

Необхідність мінімізації довжин символьних змінних, як у процесі валідації так і на фінальному етапі розробки датасетів клінічних досліджень, досягається завдяки розробленому SAS-макросу, який аналізує вміст кожної символьної змінної, на базі чого робить висновок щодо необхідного змінення її довжини,

а також створює датасет, що містить найдовші знайдені значення та відповідні до них довжини. Останнє забезпечує надійний контроль над процесом виконання обробки та прийняття рішень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Clinical Trials – Overview. URL: https://www.who.int/health-topics/clinical-trials (дата звернення 22.02.2024).
2. BioPharma Services Inc., How to Conduct Human Clinical Trials. URL: https://www.biopharmaservices.com (дата звернення 22.02.2024).
3. A. Randall, B. Coar. Risk-based Validation in Clinical Trial Reporting: Focus on What Matters Most. // *PharmaSUG*, Paper SI-02, 2018, 10 p.
4. Dipl.- Dok. Ch. Ziegler, F. Hoffmann. Risk Based Approach Applied to the Validation of Report Objects. // *PhUSE*, Paper RA02, 2008, 8 p.
5. Z. Zeng. A SAS® Macro to Create Validation Summary of Dataset Report. // *PharmaSUG*, Paper *EP-25*, 2018, 6 p.
6. URL: https://www.sas.com (дата звернення 23.02.2024).
7. J. Shostak. *SAS® Programming in the Pharmaceutical Industry*. Cary, NC: SAS Institute Inc., 2005, 355 p.
8. Ph. Spector. *An Introduction to the SAS System*. Berkeley, University of California, 2013, 168 p.
9. Clinical Data Interchange Standards Consortium, Inc. *Study Data Tabulation Model Implementation Guide: Human Clinical Trials*. Version 3.4, 2021, 461 p.
10. URL: https://www.cdisc.org (дата звернення 23.02.2024).
11. Clinical Data Interchange Standards Consortium, Inc. *Analysis Data Model (ADaM)*. Version 2.1, 2009, 41 p.

12. CDASH Data Definitions and CRF Examples. URL: https://www.cdisc.org/standards/foundational/cdash.
13. Clinical Data Interchange Standards Consortium, Inc. *Analysis Data Model Implementation Guide*. Version 1.2, 2019, 110 p.

ДОДАТОК 1 Код макросу _validate_

```
%macro _validate_(dsin_prod =, dsin_val =, dsout_comp =,
id_vars =, reorder =);
    %if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
        %put ERR%str(OR:) [_VALIDATE_ macro]: Production
and/or validation datasets were not specified. Exiting macro.;
        %return;
    %end;
    %if "&dsout_comp." = "" %then %do;
        %put ERR%str(OR:) [_VALIDATE_ macro]: Comparison
results dataset was not specified. Exiting macro.;
        %return;
    %end;

    %local comp_nobs usubjid_var dif_cvars dif_nvars;

    %if "&id_vars." ^= "" %then %do;
        proc sort data = &dsin_prod. out =
work.&dsin_prod._id;
            by &id_vars.;
        run;

        proc sort data = &dsin_val. out = work.&dsin_val._id;
            by &id_vars.;
        run;

        %let dsin_prod = &dsin_prod._id;
```

```
        %let dsin_val  = &dsin_val._id;
    %end;

    proc compare base = &dsin_prod. compare = &dsin_val. out =
&dsout_comp. listobs listvars %str(err)or outbase outcomp
outnoequal outdif novalues;
        %if "&id_vars." ^= "" %then %do;
            id &id_vars.;
        %end;
    run;

    data _null_;
        call symputx("comp_nobs", NOBS);
        if 0 then set &dsout_comp. nobs = NOBS;
        stop;
    run;

    %if &comp_nobs. ^= 0 %then %do;
        data _null_;
            set &dsout_comp. (where = (_TYPE_ = "DIF")) end =
EOF;
            array CVARS[*] _character_;
            array NVARS[*] _numeric_;

            length DIF_CVARS DIF_NVARS $3200;
            retain DIF_CVARS DIF_NVARS;

            do i = 1 to hbound(CVARS);
```

```
                if _N_ <= 1 and vname(CVARS[i]) = "USUBJID"
then call symputx("usubjid_var", "USUBJID");
                if index(CVARS[i], "X") and ^index(DIF_CVARS,
vname(CVARS[i])) and vname(CVARS[i]) ^= "_TYPE_" then
DIF_CVARS = catx(" ", DIF_CVARS, vname(CVARS[i]));
            end;

            do i = 1 to hbound(NVARS);
                if ^missing(NVARS[i]) and ^index(DIF_NVARS,
vname(NVARS[i])) and vname(NVARS[i]) ^= "_OBS_" then DIF_NVARS
= catx(" ", DIF_NVARS, vname(NVARS[i]));
            end;

            if EOF then do;
                call symputx("dif_cvars", DIF_CVARS);
                call symputx("dif_nvars", DIF_NVARS);
                put "NOTE: [_VALIDATE_ macro]: List of unequal
character variables: " DIF_CVARS;
                put "NOTE: [_VALIDATE_ macro]: List of unequal
numeric variables: "   DIF_NVARS;
            end;
        run;

        data &dsout_comp.;
            format _TYPE_ _OBS_
                %if "&id_vars." ^= "" %then %do;
                    &id_vars.
                %end;
                %else %if "&usubjid_var." ^= "" %then %do;
```

```
                    &usubjid_var.
                %end;
                %if "%upcase(&reorder.)" = "CHAR" %then %do;
                    &dif_cvars.
                %end;
                %else %if "%upcase(&reorder.)" = "NUM" %then
%do;
                    &dif_nvars.
                %end;
            ;
            set &dsout_comp.;
        run;
    %end;

    %if &comp_nobs. ^= 0 %then %put ERR%str(OR:) [_VALIDATE_
macro]: Production (&dsin_prod.) and validation (&dsin_val.)
datasets are not equal! See &dsout_comp. for details.;
                        %else %put NOTE: [_VALIDATE_ macro]:
Production (&dsin_prod.) and validation (&dsin_val.) datasets
match by common variables.;
%mend _validate_;
```

ДОДАТОК 2 Код макросу _label_check_

```
%macro _label_check_(dsin_prod =, dsin_val =);
    %if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
        %put ERR%str(OR:) [_LABEL_CHECK_ macro]: Production
and/or validation datasets were not specified. Exiting macro.;
        %return;
    %end;

    %local prod_lbl val_lbl;

    proc contents data = &dsin_prod. out = prod_metadata (keep
= MEMLABEL) noprint;
    run;

    proc contents data = &dsin_val. out = val_metadata (keep =
MEMLABEL) noprint;
    run;

    proc sql noprint;
        select distinct MEMLABEL
            into :prod_lbl trimmed
            from prod_metadata
        ;
        select distinct MEMLABEL
            into :val_lbl trimmed
            from val_metadata
        ;
```

```
        drop table prod_metadata, val_metadata;
    quit;

    %if "&prod_lbl." ^= "&val_lbl." %then %do;
        %put ERR%str(OR:) [_LABEL_CHECK_ macro]: Production
and validation datasets have different labels.;
        %put ERR%str(OR-) [_LABEL_CHECK_ macro]: Production
dataset label is "&prod_lbl.".;
        %put ERR%str(OR-) [_LABEL_CHECK_ macro]: Validation
dataset label is "&val_lbl.".;
    %end;
    %else %put NOTE: [_LABEL_CHECK_ macro]: Production and
validation datasets have same labels.;
%mend _label_check_;
```

ДОДАТОК З Код макросу _sort_check_

```
%macro _sort_check_(dsin_prod =, dsin_val =);
    %if "&dsin_prod." = "" or "&val_ds." = "" %then %do;
        %put ERR%str(OR:) [_SORT_CHECK_ macro]: Production
and/or validation dataset names were not provided. Exiting
macro.;
        %return;
    %end;

    %local prod_is_sorted val_is_sorted prod_sort_order
val_sort_order __dummy__;

    proc contents data = &dsin_prod. out = prod_metadata (keep
= NAME SORTED SORTEDBY) noprint;
    run;

    proc contents data = &dsin_val. out = val_metadata (keep =
NAME SORTED SORTEDBY) noprint;
    run;

    proc sql noprint;
        select distinct SORTED
            into :prod_is_sorted trimmed
            from prod_metadata
        ;
        select distinct SORTED
            into :val_is_sorted trimmed
```

```
            from val_metadata
        ;

        %if &prod_is_sorted. = 1 %then %do;
            select distinct NAME, SORTEDBY
                into :prod_sort_order separated by ', ',
:__dummy__ separated by ', '
                from prod_metadata
                where ^missing(SORTEDBY)
                order by SORTEDBY
            ;
        %end;

        %if &val_is_sorted. = 1 %then %do;
            select distinct NAME, SORTEDBY
                into :val_sort_order separated by ', ',
:__dummy__ separated by ', '
                from val_metadata
                where ^missing(SORTEDBY)
                order by SORTEDBY
            ;
        %end;

        drop table prod_metadata, val_metadata;
    quit;

    %if "&prod_sort_order." = "&val_sort_order." %then %do;
        %put NOTE: [_SORT_CHECK_ macro]: Production and
validation datasets have the same sorting order.;
```

```
    %end;
    %else %do;
        %put ERR%str(OR:) [_SORT_CHECK_ macro]: Production and
validation datasets are sorted differently:;
        %if &prod_is_sorted. = 1 %then %put ERR%str(OR-)
[_SORT_CHECK_ macro]: Production dataset is sorted by
&prod_sort_order..;
                                %else %put ERR%str(OR-)
[_SORT_CHECK_ macro]: Production dataset is not sorted.;
        %if &val_is_sorted.  = 1 %then %put ERR%str(OR-)
[_SORT_CHECK_ macro]: Validation dataset is sorted by
&val_sort_order..;
                                %else %put ERR%str(OR-)
[_SORT_CHECK_ macro]: Validation dataset is not sorted.;
    %end;
%mend _sort_check_;
```

ДОДАТОК 4 Код макросу _var_order_check_

```
%macro _var_order_check_(dsin_prod =, dsin_val =, dsout_comp
=);
    %if "&dsin_prod." = "" or "&dsin_val." = "" %then %do;
        %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]:
Production and/or validation datasets were not specified.
Exiting macro.;
        %return;
    %end;
    %if "&dsout_comp." = "" %then %do;
        %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]:
Comparison results dataset was not specified. Exiting macro.;
        %return;
    %end;

    %local prod_var_cnt val_var_cnt sort_var diff_st_name
diff_st_pos diff_found;

    proc contents data = &dsin_prod. out = prod_metadata (keep
= NAME VARNUM LABEL) noprint;
    run;

    proc contents data = &dsin_val. out = val_metadata (keep =
NAME VARNUM LABEL) noprint;
    run;

    data _null_;
```

```
        call symputx("prod_var_cnt", NOBS);
        if 0 then set prod_metadata nobs = NOBS;
        stop;
    run;

    data _null_;
        call symputx("val_var_cnt", NOBS);
        if 0 then set val_metadata nobs = NOBS;
        stop;
    run;

    %if &prod_var_cnt. = 0 %then %do;
        %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]:
Production dataset has no variables. Exiting macro.;
        %return;
    %end;

    %if &val_var_cnt. = 0 %then %do;
        %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]:
Validation dataset has no variables. Exiting macro.;
        %return;
    %end;

    proc sort data = prod_metadata;
        by NAME;
    run;

    proc sort data = val_metadata;
        by NAME;
```

```
    run;

    data &dsout_comp.;
        format PRES NAME VORD_P VORD_V VLABEL_P VLABEL_V;
        length PRES $20;
        merge prod_metadata (in = in_PROD rename = (VARNUM =
VORD_P LABEL = VLABEL_P))
              val_metadata  (in = in_VAL  rename = (VARNUM =
VORD_V LABEL = VLABEL_V));
        by NAME;

        if in_PROD and in_VAL then PRES = 'BOTH';
        else if in_PROD       then PRES = 'PROD';
        else if in_VAL        then PRES = 'VAL';
                              else PRES = 'MACRO ERROR';
    run;

    proc sql noprint;
        select max(VARNUM)
            into :prod_var_cnt trimmed
            from prod_metadata
        ;
        select max(VARNUM)
            into :val_var_cnt trimmed
            from val_metadata
        ;
    quit;
```

```
    %if &prod_var_cnt. >= &val_var_cnt. %then %let sort_var =
VORD_P;
                                        %else %let sort_var =
VORD_V;

    proc sort data = &dsout_comp.;
        by &sort_var.;
    run;

    data &dsout_comp. (drop = prev_: DIFF_FOUND);
        format PRES ORD_MATCH NAME VORD_P VORD_V VLABEL_P
VLABEL_V;
        length ORD_MATCH prev_VORD DIFF_FOUND 8 prev_VNAME
$200;
        set &dsout_comp.;

        retain ORD_MATCH DIFF_FOUND 0 prev_VNAME prev_VORD;

        label PRES      = 'Presence in Datasets';
        label ORD_MATCH = 'Variable Order Is Matching';
        label NAME      = 'Variable Name';
        label VORD_P    = 'Variable Order in Production
Dataset';
        label VORD_V    = 'Variable Order in Validation
Dataset';
        label VLABEL_P  = 'Variable Label in Production
Dataset';
        label VLABEL_V  = 'Variable Label in Validation
Dataset';
```

```
        if VORD_P = VORD_V then ORD_MATCH = 1;
                            else ORD_MATCH = 0;

        if ORD_MATCH and ^DIFF_FOUND then do;
            prev_VNAME = NAME;
            prev_VORD  = &sort_var.;
            call symputx("diff_st_name", NAME);
            call symputx("diff_st_pos",  &sort_var.);
        end;
        else do;
            DIFF_FOUND = 1;
            call symputx("diff_found", 1);
        end;
    run;

    %if &diff_found. ^= 1 %then %put NOTE: [_VAR_ORDER_CHECK_
macro]: Production and validation datasets have the same
variable order.;
    %else %do;
        %put ERR%str(OR:) [_VAR_ORDER_CHECK_ macro]:
Production and validation datasets have different variable
order.;
        %put ERR%str(OR-) [_VAR_ORDER_CHECK_ macro]: The
difference starts from the variable &diff_st_name. at position
&diff_st_pos..;
    %end;

    proc sql noprint;
```

```
        drop table prod_metadata, val_metadata;
    quit;

%mend _var_order_check_;
```

ДОДАТОК 5 Код макросу _len_minify_

```
%macro _len_minify_(dsin =, dsout =, debug_dsout =, debug =
Y);
    %if "&dsin." = "" %then %do;
        %put ERR%str(OR:) [_LEN_MINIFY_ macro]: Input dataset
was not specified. Exiting macro.;
        %return;
    %end;

     %local curr_lib curr_ds src_var_list var_cnt curr_var
out_var_names out_var_lengths;

    %if "&dsout."       = "" %then %let dsout       = &dsin.;
    %if "&debug_dsout." = "" %then %let debug_dsout =
&dsin._varlen;

    %if %sysfunc(countw(&dsin., %str(.))) = 2 %then %do;
        %let curr_lib = %upcase(%scan(&dsin., 1, %str(.)));
        %let curr_ds  = %upcase(%scan(&dsin., 2, %str(.)));
    %end;
    %else %do;
        %let curr_lib = WORK;
        %let curr_ds  = %upcase(&dsin.);
    %end;

    proc sql noprint;
        select NAME
```

```
            into :src_var_list separated by ' '
            from dictionary.columns
            where upcase(LIBNAME) = "&curr_lib." and
upcase(MEMTYPE) = 'DATA' and upcase(MEMNAME) = "&curr_ds." and
upcase(TYPE) = 'CHAR'
        ;
    quit;

    %let var_cnt = %sysfunc(countw(&src_var_list., %str( )));

    proc sql noprint;
        create table &debug_dsout. as
            %do i = 1 %to &var_cnt.;
                %let curr_var = %scan(&src_var_list., &i.,
%str( ));
                select distinct "&curr_var." as VAR_NAM label
= "&dsin. Variable Name",
                       length(&curr_var.)    as MAX_LEN label
= 'Maximum Length',
                       &curr_var.            as MAX_VAL label
= 'The Longest Content'
                from &dsin.
                having MAX_LEN = max(MAX_LEN) and monotonic()
= min(monotonic())
                %if &i. < &var_cnt. %then %str(outer union
corr);
            %end;
            order by VAR_NAM
        ;
```

```
        select VAR_NAM, MAX_LEN
            into :out_var_names separated by ' ',
:out_var_lengths separated by ' '
            from &debug_dsout.
        ;
        %if &dsout. ^= &dsin. %then %do;
            create table &dsout. as
                select *
                from &dsin.
            ;
        %end;
        alter table &dsout.
            %do i = 1 %to &var_cnt.;
                modify %scan(&out_var_names., &i., %str( ))
char(%scan(&out_var_lengths., &i., %str( )))
            %end;
        ;
        %if "&debug." = "" %then %do;
            drop table &debug_dsout.;
        %end;
    quit;
%mend _len_minify_;
```