PharmaSUG China 2024 - Paper DV-10046

Interactive analyses and displays for clinical data by utilizing a complete suite of analysis and reporting (A&R) tools via MedDRAH

Zhiping Yan, Dizal Pharma

ABSTRACT

The establishment of a systematic and well-planned process for conducting Ongoing Aggregate Safety Evaluations (OASEs) is crucial in order to meet the FDA's safety reporting requirements for Investigational New Drugs (INDs). Furthermore, in 2023, the NMPA CDE issued technical guidance on the clinical safety assessment of new drugs. It states that a clinical trial sponsor must report serious unexpected suspected adverse reactions for which there is evidence to suggest a causal relationship between the drug and the adverse event. Over time, there has been an evolution in tools used to view safety data, transitioning from tabular and line-listing formats to graphical representations.

At Dizal, we have developed the Medical Data Review and Analysis Hub (MedDRAH/'med'ra:/), an interactive web-based platform that enables medical and PV physicians, statisticians, and PK scientists to efficiently review, analyze, and generate both tabular and graphical reports using intuitive point-and-click wizards. Additionally, this platform allows users to interact with elements within graphs beyond just navigating through the webpage itself.

This presentation will demonstrate how such a platform can be designed and developed using R and JavaScript.

INTRODUCTION

There has been an increased demand on the proactive and comprehensive evaluation of safety data to ensure subject well-being throughout the clinical trial life cycle. Currently, disparate solutions including manual processes are applied by different key stakeholders for medical data review, analysis, monitoring, and management. Data must be transferred into different locations and may be processed in different ways to meet diverse needs of each stakeholder. With escalating data volumes and intricate review processes, huge pressure is growing for data processing and reviewing, as well as getting the subject data earlier and flawlessly.

To solve pain points and improve review efficiency, we provided a unified end-to-end platform — Medical Data Review and Analysis Hub (MedDRAH /'med'ra:/) — to automate data review and analysis with one common data source. The web application of this platform offers a user-friendly interface which allows all stakeholders to automate data review and analysis by clicking on several intuitive wizards.

OVERVIEW

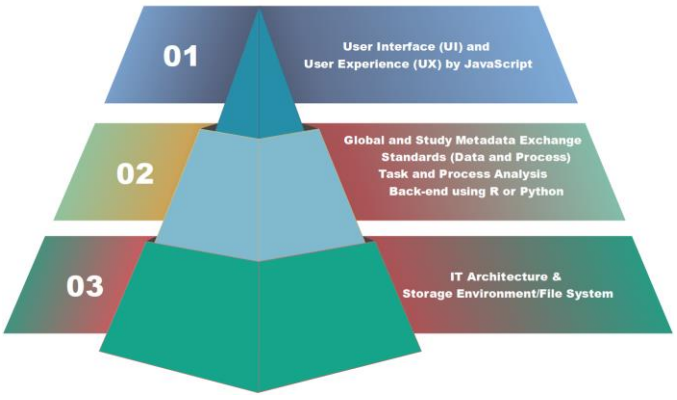

Figure 1. Structure of MedDRAH

Medical Data Review and Analysis Hub (MedDRAH) is essentially an ecosystem which consists of three levels of contents. The lowest or third level is the foundation of whole platform and includes the IT architecture like web server, storage environment and file system. The second level contains metadata, data, Back-end writing with Python or R, output, software, and various types of documents. The highest or first level is the view level through which users can perform data review, analysis, and other tasks by interacting with the elements in user interface.

ACCESS CONTROL

Data in clinical trials is extremely sensitive information and Role-Based Access Control (RBAC) must be implemented to manage data access. Based on stakeholders' role and responsibilities, only authorized individuals or groups can access data.

To achieve this RBAC, we designed following folder structures. All metadata files are placed under ***ar-mdr*** folder while all clinical trial data and output files are placed under ***ar-dev*** folder. Compound level folder (e.g. ***training***) and study level folder (e.g. ***standard***) are extensible. Task level folders including ***mdr***, ***narrative*** and ***csr*** are not extensible. The extensibility of compound level and study level folders as well as inextensibility of task level folders can be applied to implement interface level access control. Name of compound level, study level and task level folders (e.g. in blue from Figure 2) for which users have been granted access will be displayed in web (bottom-left part of Figure 2).

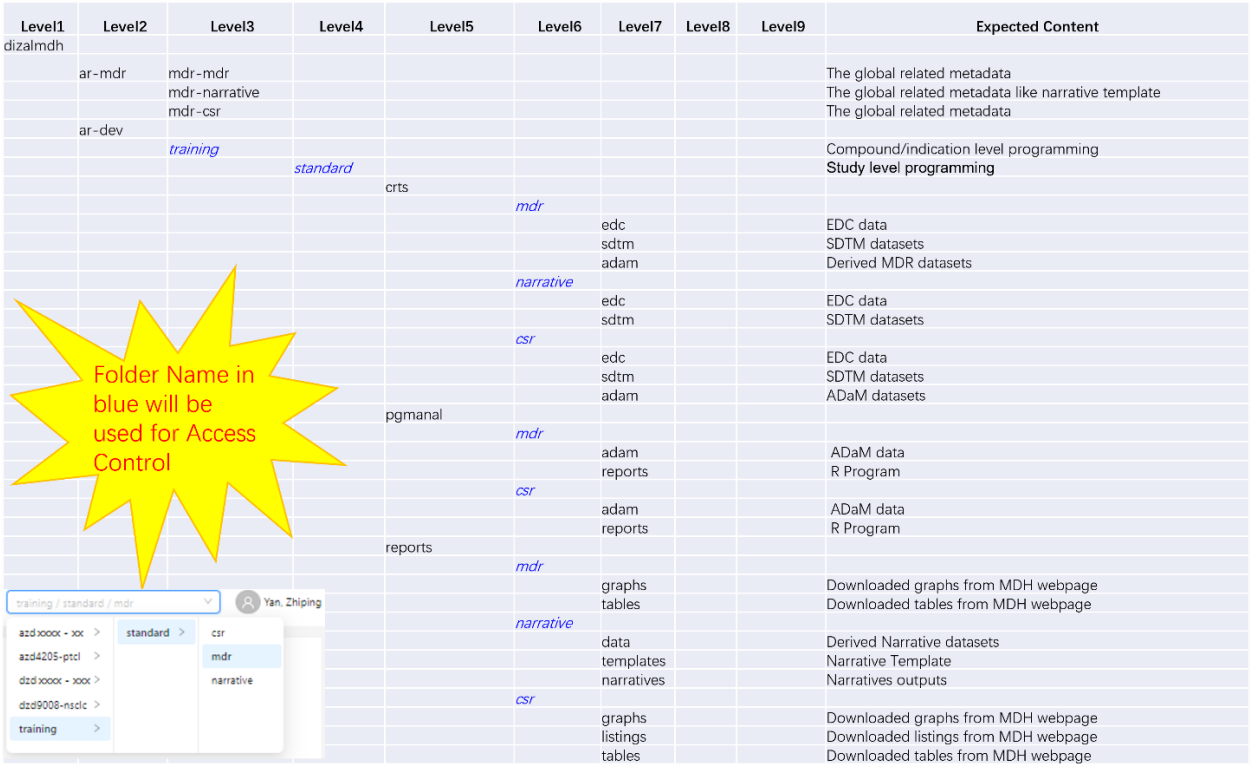

Figure 2. Folder Structure of MedDRAH

Each study level folder has a unique AD (Active Directory) security group. For example, AD security group for ***standard*** folder showed in Figure 2 is ***DZ-MDH-STANDARD***. With this one-to-one mapping rule between AD security group name and study level folder, folders for which users have access can be easily retrieved by knowing AD security group names for which users have been assigned.

With API from IT colleagues, a dictionary contains ***groups*** key will be received as part of response. This returned value is unique for each user. Value of groups key includes AD security group name (e.g. DZ-MDH-**STANDARD**, DZ-MDH-DZXXXXXXX1, DZ-MDH-DZXXXXXXX2, DZ-MDH-DZXXXXXXX3) of all folders for which users have access.

These AD security group names can be served as inputs for a MedDRAH API (e.g. http://xxxxxx.dizalxxx.xxx/xxx /xxxx/getdirs/?groups= DZ-MDH-**STANDARD**, DZ-MDH-DZXXXXXXX1, DZ-MDH-DZXXXXXXX2, DZ-MDH-DZXXXXXXX3) to get a list of dictionary which will be used as value of access dropdown.

By filling returned list (4th box in Figure 3), folder names for which users have access will be displayed in dropdown in top-right corner of web page. By selecting **training -> standard -> mdr** of dropdown, users can only get access to **/training/standard/mdr** folder and its sub folders.

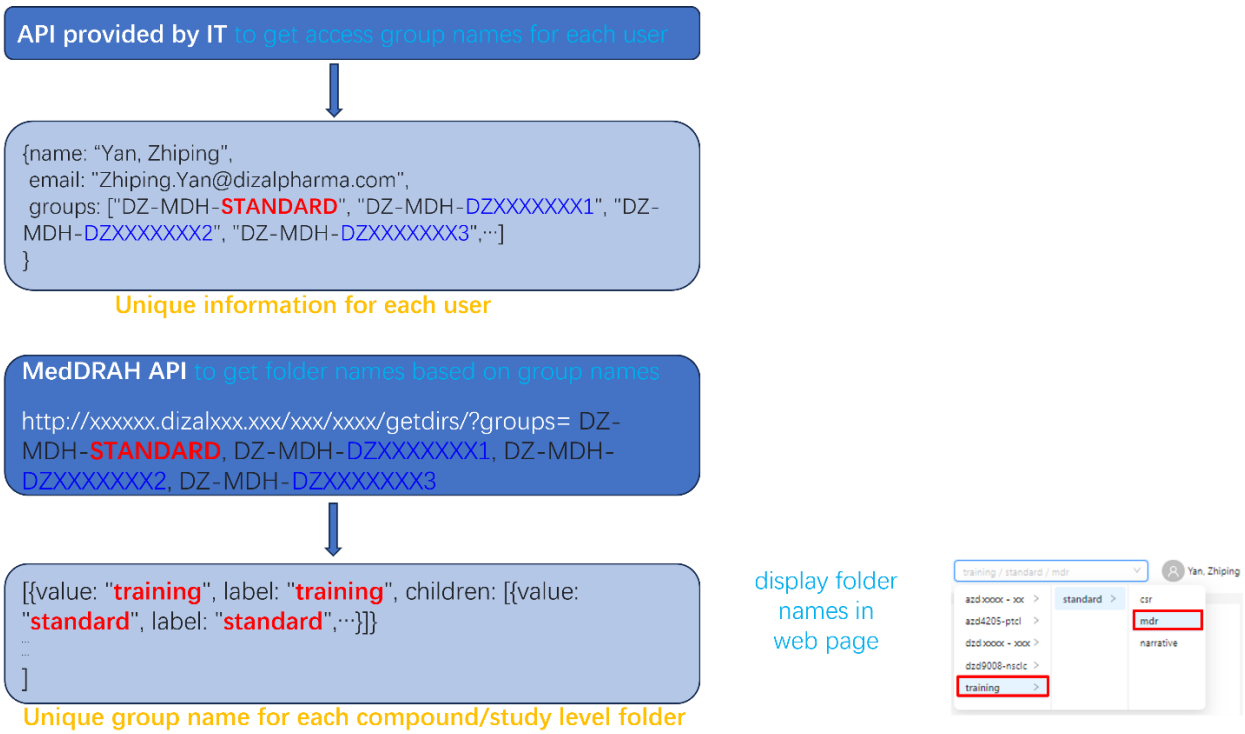

Figure 3. API for Access Control

All input datasets such as EDC datasets, SDTM datasets, and ADaM datasets are placed under /crts folders. With previous described approach, all users including Medical Scientist, PK Scientist, Statistician and PV Physician can also get access to these datasets. Obviously, this is not expected and should be avoided.

By assigning all statistical programmers to ***DZ-MDH-PROGRAMMING*** group and only this AD security group is allowed to have access to /crts folder and its sub folders. With inheritance property of Window AD access control system, the goal that only study level statistical programmer has access to study level /crts folder can be achieved. In our example, only statistical programmer assigned to ***DZ-MDH-STANDARD*** security group have access to **/training/standard/crts** folders.

OVERVIEW OF FRONT-END & BACK-END TECHNOLOGIES

Our web application is supported by two back-ends and one front-end. The R back-end is applied to process SAS data for medical data review while Python back-end is used for generation of narrative or pre-clinical pharmacology report.

For R back-end, ***Plumber*** package is used for API generation while other packages such as ***haven***, **dplyr**, ***tidyr***, ***stringr***, ***rjson***, ***labelled***, ***ggplot2***, ***survival*** and ***survminer*** are used for data processing.

Regarding Python back-end, ***Flask*** is used to generate API. Packages including ***pandas***, ***docx***, ***pyreadstat***, ***re***, ***os***, ***math***, ***matplotlib*** and ***numpy*** are used for SAS dataset, Excel file and DOCX file processing.

Front-end was developed using React. Four React packages were used to improve web application building efficiency. ***antd*** provides plenty of UI components to enrich web applications and improve components experience consistently. ***react-table*** offers components to display data in table format. For figures, ***plotly*** and ***echarts*** are used.

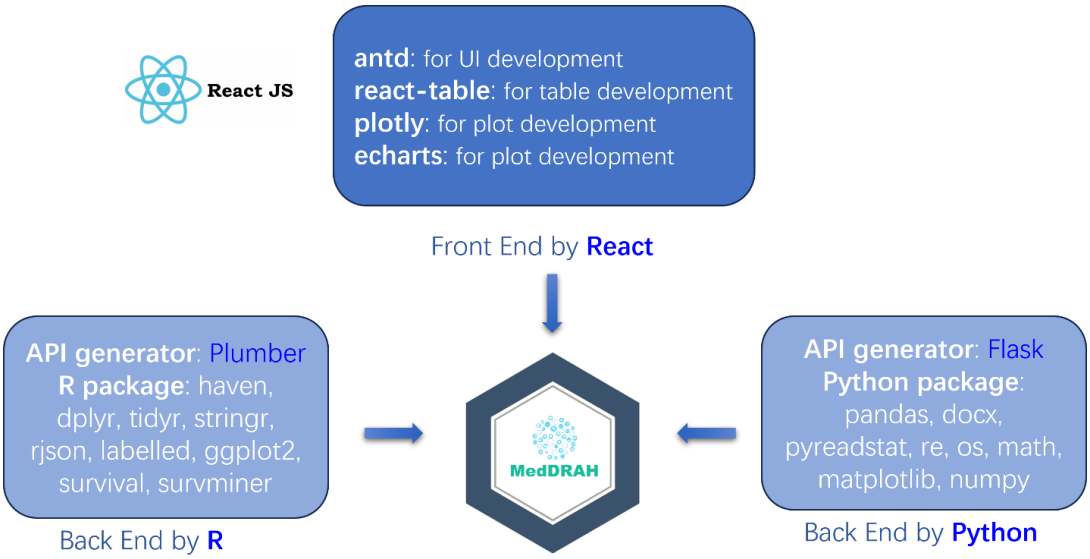

Figure 4. Overview of Front-End and Back-End Technologies

BACK-END BY R VIA PLUMBER

Plumber allows you to create a web API by merely decorating your existing R source code with roxygen2-like comments (e.g. ***#* @get mdh/mdr/adam*** in Figure 5). These comments allow plumber to make your R functions (e.g. ***function(study_task_path, param, adam)*** in Figure 5) available as API endpoints. All R functions for API should be written in plumber.R file. This plumber.R file should be sourced in runPlumber.R to start back-end and provide corresponding APIs.

```
#####################################################################
# Program:              plumber.R
# Keywords:
# Function:             This program is used to create back end function
# Author:               Zhiping Yan
# Date Completed:
# R Version:            4.2.3
# Input Data:
# Function Called:
# Program Output:
#
# Limitations/Cautions:
#
# Revisions:

#####################################################################

#*@apiTitle Example Plumber API

library(haven)
library(dplyr)
library(tidyr)
library(stringr)
library(rjson)
library(labelled)
source("./adam/read_adam.R")

#####################################################################
# Generate ADaM
#####################################################################
#* @get mdh/mdr/adam
function(study_task_path, param, adam){

  source('./rootdir.R')
  root_dir <- rootdir()

  path_lst <- as.list(strsplit(study_task_path, ","))
  data_path <- paste(unlist(path_lst),collapse = '/')

  if (file.exists(paste0(root_dir, data_path,"/pgmanal/adam/", adam, ".R")) =
    source(paste0(root_dir, data_path,"/pgmanal/adam/", adam, ".R"))
    print('import study level source adam function')
  }else{
    source(paste0("./adam/", adam, ".R"))
    print('import global level source adam function')
  }

  if (adam == "adsl") {
```

```
#############################################################
# Program:              runPlumber.R
# Keywords:
# Function:             This program is used to start back end
# Author:               Zhiping Yan
# Date Completed:
# R Version:            4.2.3
# Input Data:
# Function Called:
# Program Output:
#
# Limitations/Cautions:
#
# Revisions:

#############################################################

library(plumber)

pr("./plumber.R") %>%
 pr_run(host = '0.0.0.0', port = [redacted])
```

Figure 5. Back-End by R via Plumber

Figure 6 shows how to pass parameter between React and R. Parameters must be provided as dictionary data type in React. Parameter names (e.g. ***study_task_path***, ***trta***, ***siteid***, ***trtemfl***, ***display_total*** *and* ***byvar***) must be the same in both React and R.

As described above, all API are written in plumber.R file. To avoid complexity of plumber.R file, user-defined function (***t-aesum***) should be written in another R program file (***t-aesum.***R) and sourced by plumber.R file (see right part of Figure 6).

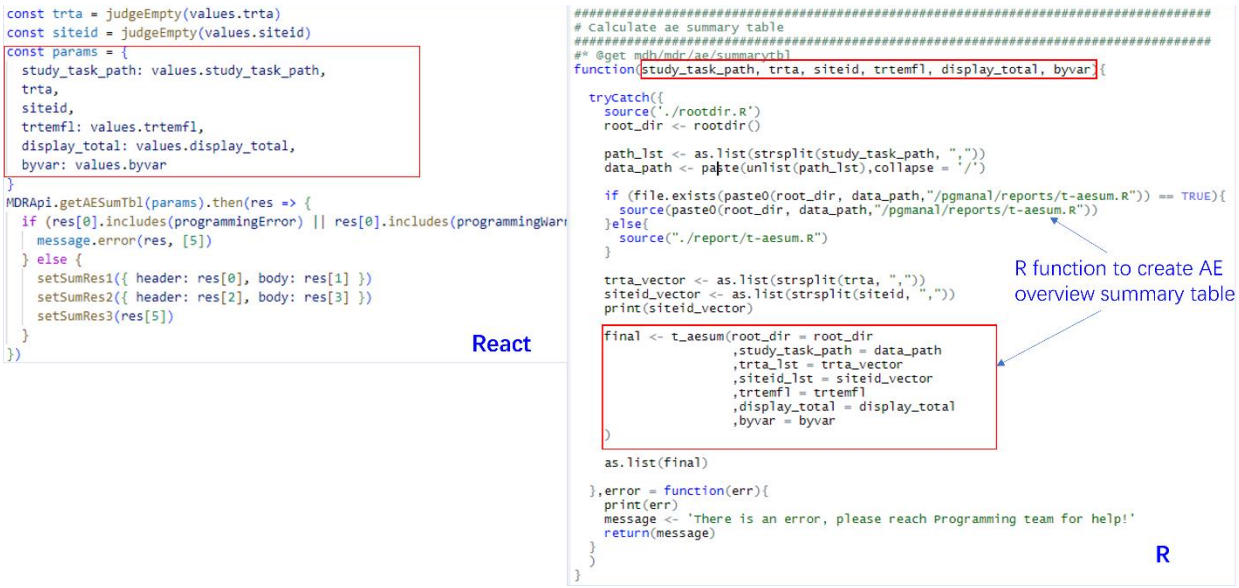

Figure 6. Pass Parameter between React and R

Figure 7 presents R code for t-aesum function which is sourced in plumber.R file. with sym function and {{ }}, R function can perform analysis by different variables like TRTA, Sex, Race, etc.

Figure 7. Example t-aesum Function sourced by plumber.R

BACK-END BY PYTHON VIA FLASK

Flask (a light WSGI web application framework) is designed to make web application getting started quick and easy. Similarly to R, API functions are placed in one python file (e.g. home.py) and this file is imported in main.py to start back-end. Blueprint can help you structure your Flask application by grouping its functionality into reusable components.

```
################################################################################
# Program:            home.py
# Keywords:
# Function:           This program is used to start generate API
# Author:             Zhiping Yan
# Date Completed:
# Revisions:

################################################################################
import json
import os
import pyreadstat
import pandas as pd
from flask import Blueprint
from flask import request
from glob2 import glob
import ast
from datetime import datetime
from root_dir import *

home = Blueprint('home', __name__)

@home.route('/mdh/home/getdirs/all', methods=["POST", "GET"])
def extract_path_all():
    lvl1_dict = {}
    for root, dirs, files in sorted(os.walk(root_dir, topdown=True)):
        if root[len(root_dir):].count(os.sep) == 0:
            lvl0_dir_lst = []
            for dir in dirs:
                dict = {}
                dict['value'] = dir
                dict['label'] = dir
                lvl0_dir_lst.append(dict)
```

```
################################################################
# Program:            main.py
# Keywords:
# Function:           This program is used to start back end
# Author:             Zhiping Yan
# Date Completed:
# Revisions:

################################################################
from flask import Flask
from flask_cors import CORS

from home import home
from narrative import narrative
from pharmacokinetics import pharmacokinetics

app = Flask(__name__)
cors = CORS(app)

app.register_blueprint(home)
app.register_blueprint(narrative)
app.register_blueprint(pharmacokinetics)

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=[redacted], threaded = True, debug=True)
```

Figure 8. Back-End by Python via Flask

Figure 9 presents how to pass parameter between React and Python. Parameter names (e.g. ***folder***) should be provided directly in React. While in python, parameter name are placed in ***request.arg.get()***.

```
if (checkedValues.target.checked === true) {
  const params = {
    folder: judgeEmpty(narrativeFormValues['folder'])
  }
  NarrativeApi.getAESIGT(params).then(res => {
    if ((res.length === 0) & (res.length === undefined)) {
      setModalOpenYN(true)
      handleRes('Please reach Study Level Programmer Lead to query why va
      setOpenYN(false)
    } else {
      setAesiYN(true)
      setAesiLst(res)
      setAesiSelectLst(res)
      console.log(aesiLst)
    }
  })
} else {
  setAesiYN(false)
}
```

```
################################################################################
# CoverPage - Get AESI GT
################################################################################
@narrative.route('/mdh/narrative/getAESIGT', methods=["POST", "GET"])
def process_criteria_getAESIGT():
    folder = request.args.get('folder')
    adamFolder = root_dir +'/' + '/'.join(list(folder.split(','))) + '/crts/adam'
    outputFolder = root_dir +'/' + '/'.join(list(folder.split(','))) + '/reports/data'
    try:
        options = data_process_criteria_getAESI_GT(adamFolder, outputFolder)
        msg = 'Successful'
    except Exception as e:
        options = []
        msg = (repr(e))

    return options
```

Figure 9. Pass Parameter between React and Python

FRONT-END BY REACT COMPONENT

As showed in Figure 10, UI components such as layout, dropdown, button, icon, divider, space, menu, input, form, select, checkbox and tab from MedDRAH are developed using ***antd***. Detailed document of **antd** can be found at https://ant.design/components/overview/.

All summary tables and listings within this platform are presented with react-table. ***react-table*** component can supercharge your tables from scratch while retaining 100% control over markup and style. https://tanstack.com/table/latest details how to use it.

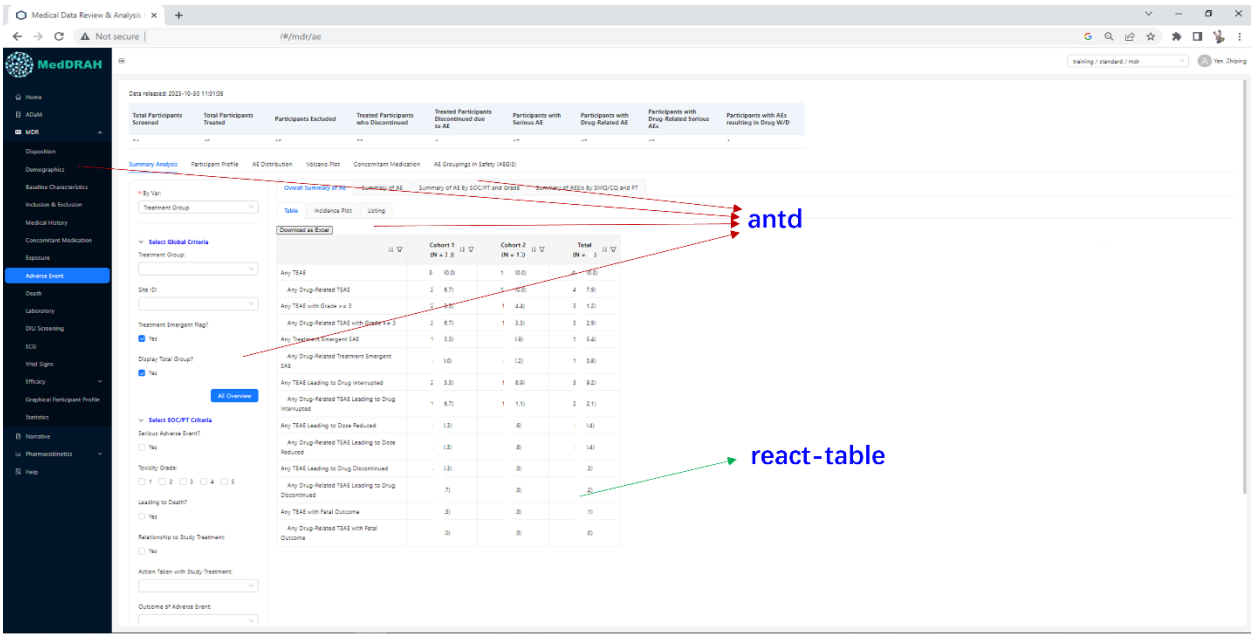

Figure 10. Front-End Component – antd and react-table

Two components - ***plotly*** (Figure 11, https://plotly.com/javascript/) and ***echarts*** (Figure 12, https://echarts.apache.org/en/index.html) – are used by our platform to present figures. plotly is a high-level, declarative charting library. It ships with over 40 chart types including 3D charts, statistical graphs and SVG maps. echarts is another open-source JavaScript visualization library.

Both libraries are easy to use. However, almost all of figures from MedDRAH are generated using plotly since it is more flexible than echarts. Providing model functions is nearly the only thing which echarts is significantly better.

Back-end API are used to process data. The post-processed data will be sent to front-end for rendering tables or figures.

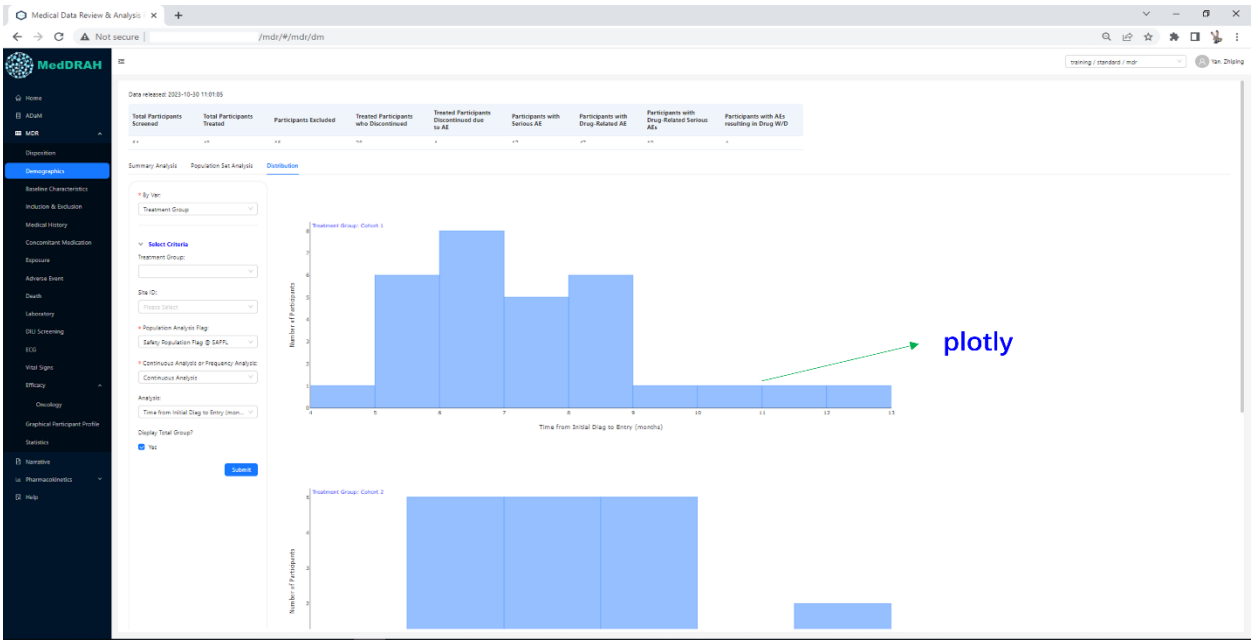

Figure 11. Front-End Component – plotly

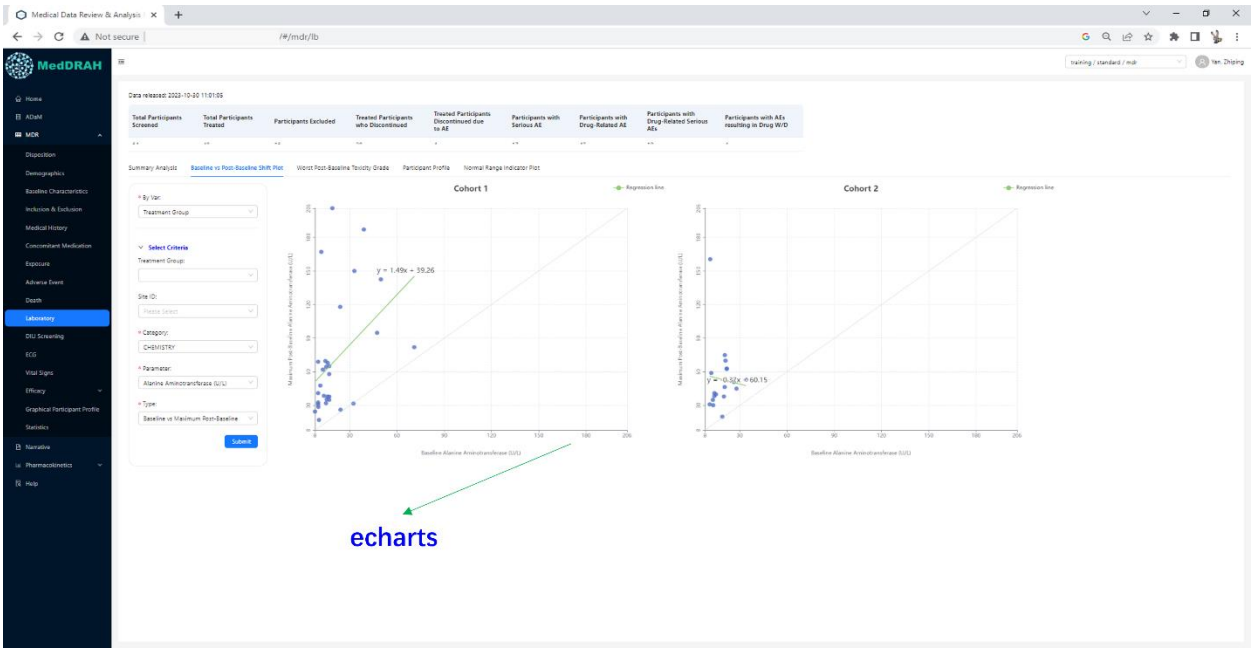

Figure 12. Front-End Component – echarts

FRONT-END BY REACT - EXAMPLE

Figure 13 is a recommended approach to perform continuous or frequency analysis using subject-level analysis dataset. Both variable label and variable name are included in dropdown values. Variable label placed before @ is intended for users to select items they are interested in. Variable name after @ can let back-end program know which variable should be analyzed.

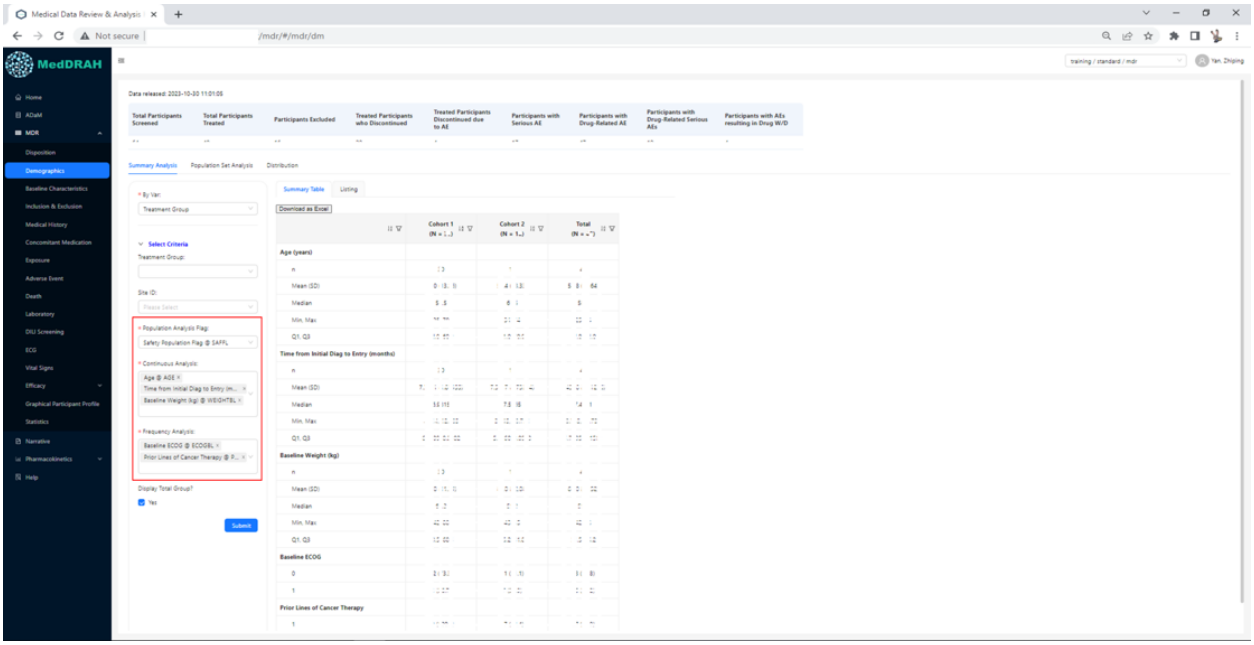

Figure 13. Front-End Example – Recommended Approach for Continuous and Frequency Analysis

With plotly and echars, almost all popular plots used in clinical trial analysis such as volcano plot, distribution plot, categorical plot, scatter plot, series plot, eDISH plot, spider plot, swimmer plot, waterfall plot can be generated.

These plots can help visualize data distribution and anomalies. Visual inspection of data reveals pattern that may not be apparent through numerical analysis alone. Just glance at Figure 14, users will know how many subjects have ALT value out of normal range.

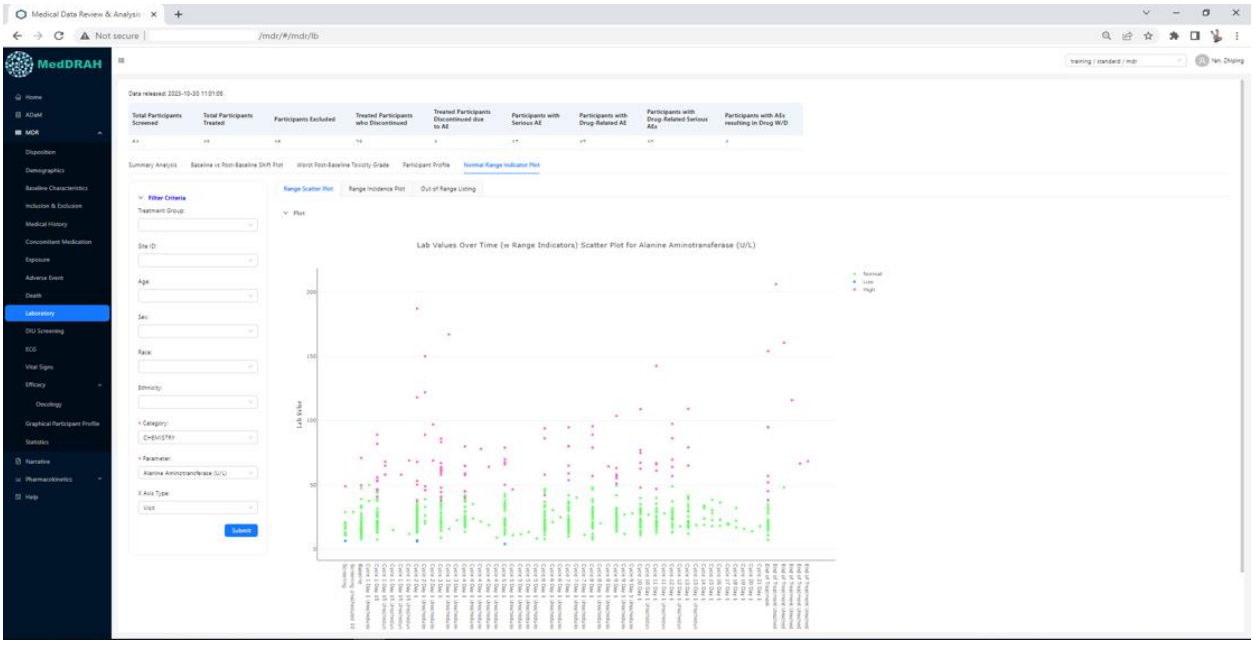

Figure 14. Scatter Plot to View Normal Range of Laboratory Data

Moreover, hover event, click event and zoom event allows users to interact with plot for further detailed information. Look at the example in Figure 15, by moving mouse over data point in eDISH plot, detailed information including subject ID, maximum BILI, maximum ALT/ASAT will be displayed.

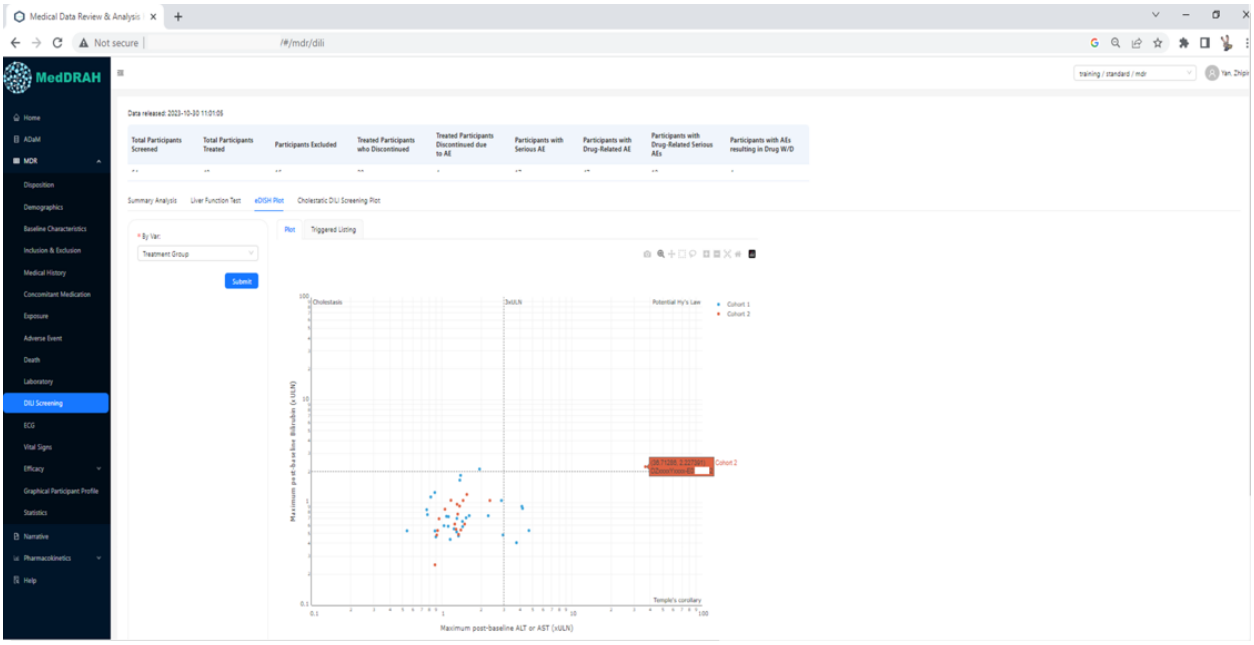

Figure 15. Hover Information of Plot

If users click on the point in Figure 15, a table showed in Figure 16 will be presented. This eliminates communication that should have been needed between medical scientist and statistical programmer. Time is saved and working efficiency is improved.

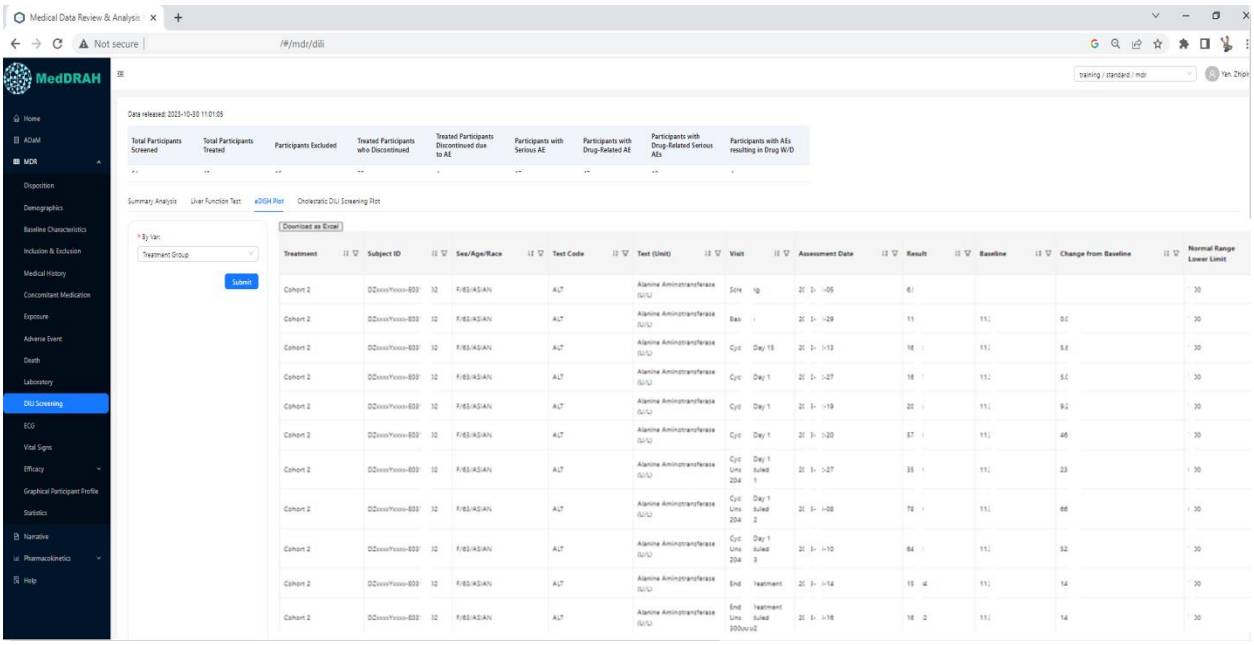

Figure 16. Click Event of Plot to Trigger Table

With extreme flexibility of plotly, special customized plot can be designed and produced. For example, series plot in Figure 17 shows ALT data of a particular subject while vertical lines present all adverse event information. By hovering over vertical lines, AE information including PT, start date and toxicity grade will be showed. Reviewers can easily identify the effect between AST and particular adverse event.

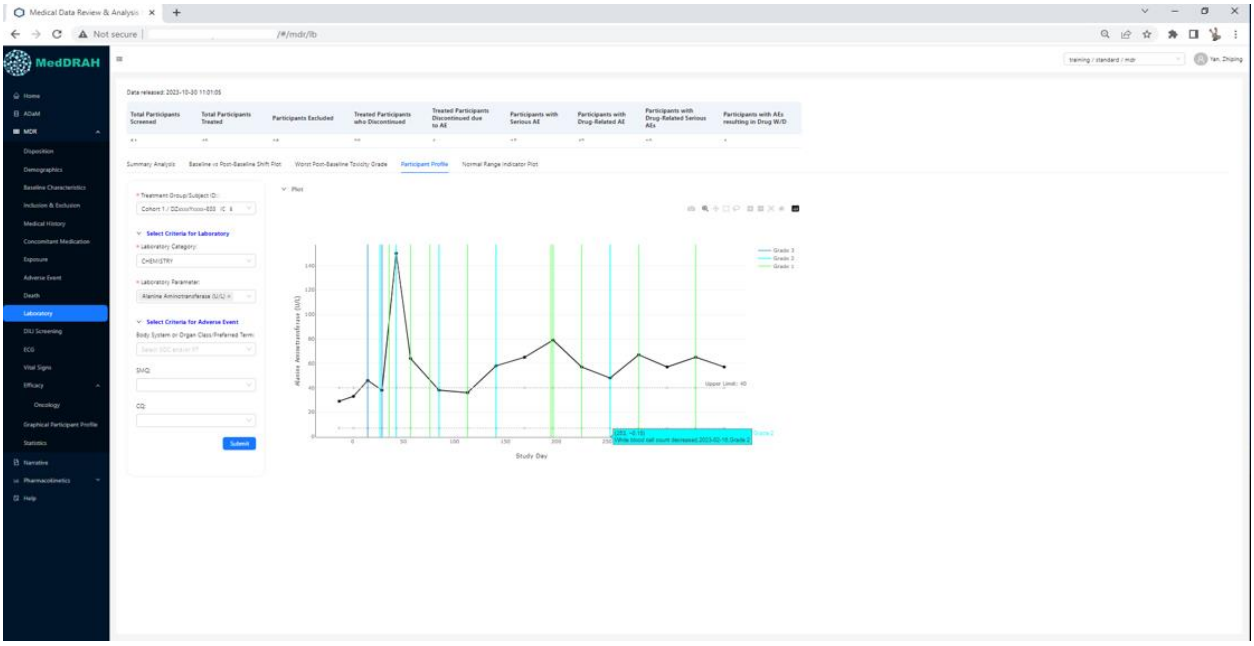

Figure 17. View Longitudinal Data and Adverse Event

Very complicated plot can also be produced. Look at Figure 18, data of laboratory, vital sign, ECG, drug administration, concomitant medication and adverse event are plotted together for users to find potential relationship. These sub plots have same X axis.

Figure 18. Check Data Across Different Domains

OTHER FEATURE

The goal of the MedDRAH is to spend as little time as possible on TLG coding. Emphasis and resources will be placed on data standardization. With same ADaM and SDTM dataset variables, this platform can help perform near real time analysis across studies.

Plus, reviewers from different functions can visit the MedDRAH and take what they need. Figure 19 shows tab for AE groupings in safety analysis which is customized for PV physician. Users from other function may not utilize this tab. This platform also provides narrative menu for medical writer only. E-R analysis, popPK and CQT analysis, pre-clinical pharmacology report generation are also provided for corresponding stakeholders, respectively.

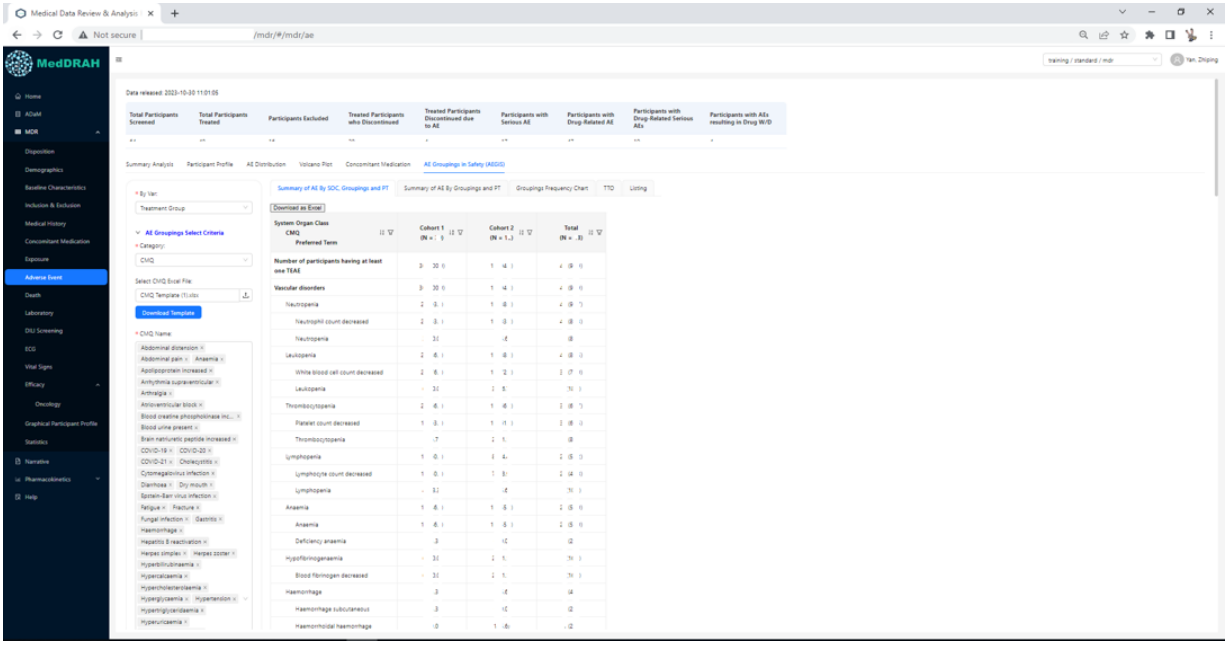

Figure 19. AEGiS Analysis for PV Physician

CONCLUSION

With common data source, the MedDRAH provides end-to-end processes which can enhance data quality, improve efficiency, and promote seamless collaboration. It effectively offers real-time monitoring and visualization of clinical data. Barriers in sharing clinical data across different functions are removed. Besides data monitoring, stakeholders such as Medical and PV Physicians, Statisticians, and PK Scientists all can also perform other kinds of tasks. Plus, the platform is extensible and other tasks can also be integrated in future.

REFERENCES

Paul Fioole. Janssen DATA4YOU platform approach for clinical data review. AD15, PharmaSUG 2024

ACKNOWLEDGMENTS

Many thanks to Huadan Li, Zongming Guo, Wei Zhang, Echo Zhuo and all Statistical Programming team for the great support!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Zhiping Yan
Dizal Pharma
Zhiping.Yan@dizalpharma.com

1. is displayed in the **File name** box, click **Insert**.