FIT3164 Data Science Software Project

User Guides and Software Test Report

Prepared by: MDS

He

Ng

Mu

Topic:

Predicting Drug Resistance in Cancer Cell Lines Using Machine Learning Approaches

Table of Contents

Please click on the table of contents to navigate to any of the sections listed below.

Section 1: End User Guide

Section 1.1: Overview of Software Deliverables and Purposes

In recent years, cancer incidence rates have remained high, resulting in significant mortality rates attributed to ineffective treatments. Understanding drug resistance in cancer cell lines is crucial for improving treatment efficacy and reducing mortality rates. The continuous investigation of drug resistance ensures that medical professionals stay updated on the latest developments and treatment options, facilitating the development of new treatment plans.

To address these challenges, Our team has developed a web application called "DRUGRES" which features an advanced predictive model built using machine learning techniques to determine drug resistance (LN_IC50) in various cancer cell lines, primarily focusing on breast cancer. Users can upload their own drug-cell pair datasets in the specified format to manually make predictions. The results are then presented in tables and visualisations for easy interpretation. Additionally, the application offers interactive features like sorting, filtering, and searching to help users navigate and analyse the data more effectively. We have also included the combined GDSC-CCLE dataset with actual LN_IC50 values as a default dataset, enabling users to quickly access drug resistance information for their research purposes.

For future potential improvements, DRUGRES can incorporate additional default datasets such as the pure GDSC and pure CCLE datasets for model training. This will not only increase the diversity of the application but also provide users with more options for drug resistance predictions. From an industrial standpoint, our application has the potential to hold promise as a foundational tool capable of predicting drug resistance in various cancer cell lines. While currently focused on breast cancer types, our model has demonstrated some improvements in prediction accuracy. This indicates the potential for future extensions to include predictions for different types of cancer cell lines. Ultimately, the reach of the application can be expanded, serving a broader audience with diverse needs and enhancing its overall value.

Section 1.2: DRUGRES - Functionalities Guide

Section 1.2.1: Options Available - Default Information Retrieval/ Manual Prediction

When the web application is launched, users will encounter two primary buttons: "**User Manual Prediction**" and "**Default Dataset**". Here is how these buttons are displayed in the application interface.

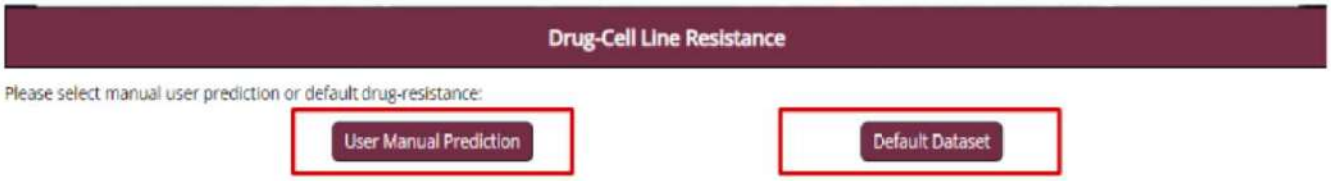


Figure 1: Display of buttons in the application interface

Initial Assumption:

Before selecting any of the two buttons, the web application will automatically assume that the default dataset option is selected. As a result, it will display the GDSC-CCLE drug-cancer dataset and the default output visualisation. Users can then proceed to choose between the two options based on your preference and requirements.

Clicking either the "User Manual Prediction" or "Default Dataset" button will lead to different user interfaces, allowing users to retrieve the drug response value(LN_IC50). Below, we will delve into the specifics of what user interface appears when each button is clicked.

User Manual Prediction

This option is **ideal for users who wish to upload their own dataset to predict drug sensitivity manually.** Upon clicking on this "User Manual Prediction" button, the interface displayed is shown as below:

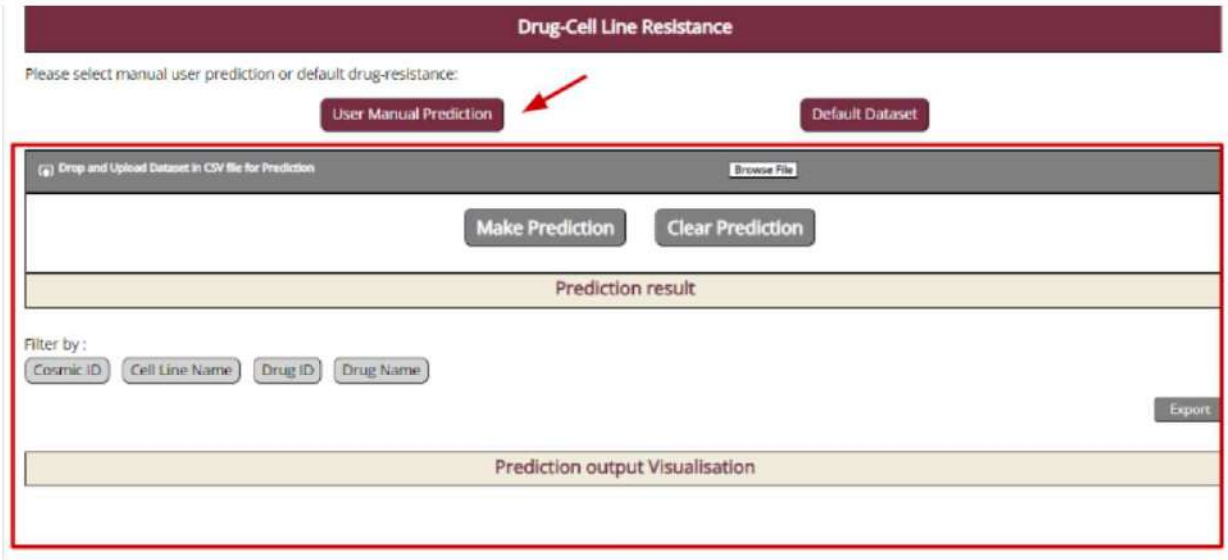


Figure 2: Interface displayed by "User Manual Prediction" button

The User Manual Prediction Guide, detailed in Section 1.2.2, will offer comprehensive instructions on how users can manually make predictions. Here, we will concentrate on outlining the distinctions in outcomes upon selecting either option, delineating their respective purposes, and highlighting the disparities in layout and interface presentation.
By default, the "**Prediction result**" section and "**Prediction output Visualisation**" section will be empty because users have not uploaded any data files or made predictions yet. As a result, there will not be any predicted results or visualisations displayed in the interface.

Default Dataset

This option is ideal for the users who prefer to **directly access drug sensitivity data (LN_IC50) from the default GDSC-CCLE drug-cancer dataset provided without making any prediction**. Upon clicking on this "Default Dataset" button, the interface displayed is shown as below:

Figure 3: Interface displayed by "Default Dataset" button

By default, the sections labelled "**Default GDSC-CCLE Drug-Cell line resistance dataset**" and "**Default GDSC-CCLE Drug-Cell Resistance Visualisation**"" will showcase the default GDSC-CCLE Drug-Cell line resistance dataset in both tabular and visual formats respectively. The table presented will include the actual drug response, denoted as LN_IC50, across various cancer types. Similarly, the visualisation will be generated based on the data presented in the default dataset. This button mainly displays the interface for users to perform information retrieval instead of making predictions.

Section 1.2.2: User Manual Prediction Guide

The following steps will guide the users to manually make a prediction:

Step 1: Click the "User Manual Prediction" button

Upon clicking the "User Manual Prediction" button, a dropbox will promptly appear, facilitating users to upload a single dataset. This dataset is expected to contain drug and cell line information in **CSV** format exclusively. As the prediction process is initiated only upon the user's upload, both the prediction result and output visualisation sections will initially remain blank. It is imperative to note that the uploaded dataset must meticulously adhere to the specific format outlined in Section 1.3: User Upload Data Format Requirements. Within the dropbox, users will find a "**Browse File**" button, enabling them to select the desired dataset from their device. Beneath the dropbox, two additional buttons, namely "**Make Prediction**" and "**Clear Prediction**", are available for executing the prediction process and clearing any previous inputs, respectively.

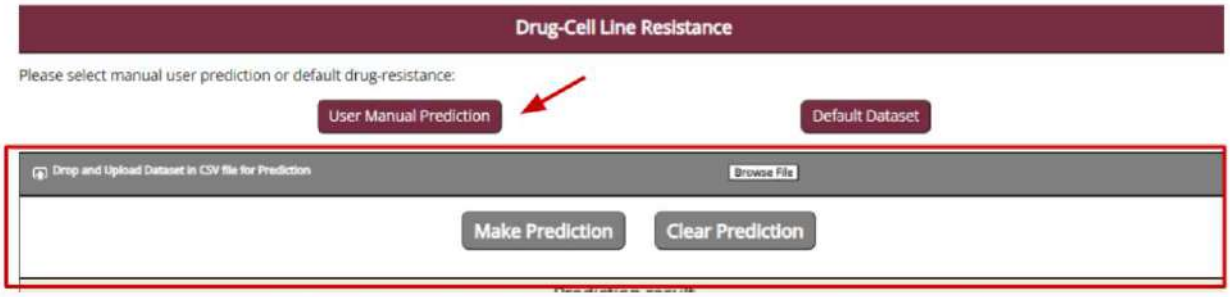


Figure 4: User Manual Prediction Uploading, Predicting and Clearing Section

Step 2: Upload the User Dataset

Upon clicking on the "Browse File" button, a local directory explorer will promptly pop up, allowing users to navigate through their device's file system. From there, users can effortlessly select the desired CSV file containing the drug and cell line information intended for prediction. It is important to note that the appearance and functionality of the directory explorer may vary slightly depending on the operating system and platform being used. As illustrated in the example figure below (for the Windows platform), clicking the "Browse File" button triggers the appearance of a file explorer.

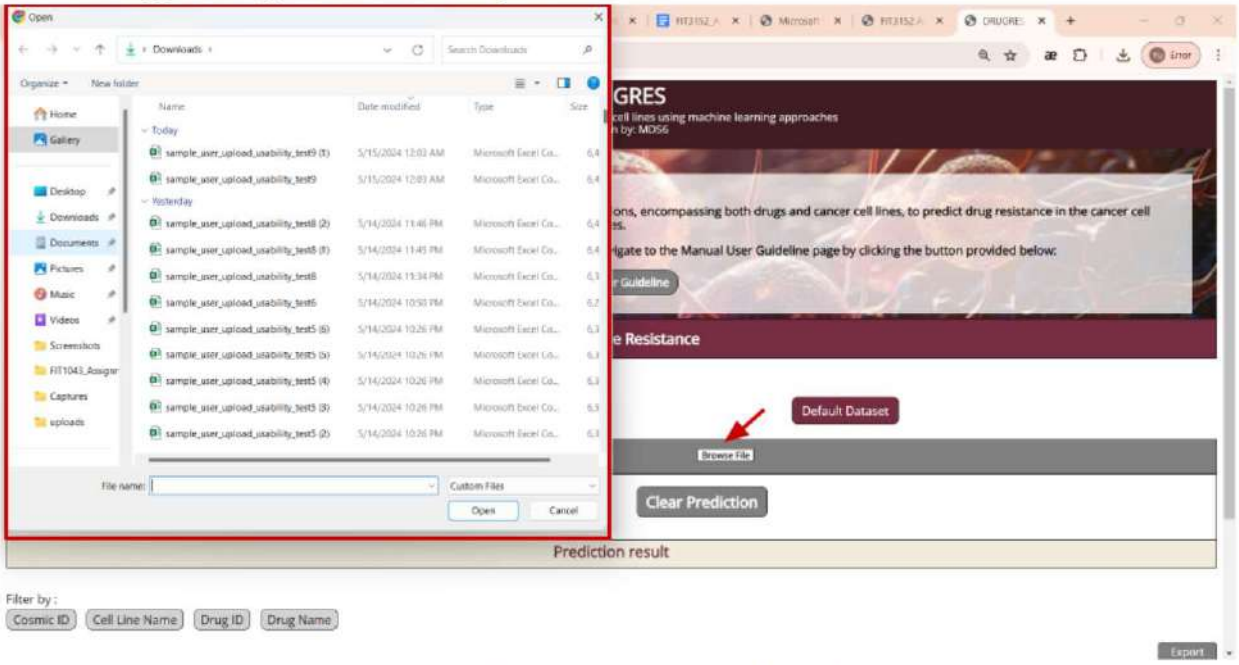


Figure 5: "Browse File" File Explorer Prompting

Additionally, for added convenience, users also have the option to directly upload their CSV file by simply dragging and dropping it into the designated area of the dropbox, highlighted in grey.

Figure 6: Files Drag and Drop Area

Upon successful upload of the CSV file, the filename will be displayed on the right side of the dropbox.

Figure 7: Display of "Uploaded File" name

**Note:
Please ensure that the CSV filename is displayed on the right side of the dropbox before proceeding to the next step by clicking the "Make Prediction" button. If the filename is not displayed, it indicates that the file has not been uploaded yet.

Step 3: Click on the **"Make Prediction"** button

Upon clicking the "Make Prediction" button, the uploaded CSV file will begin processing and being passed into the predictive model for prediction. Users are advised to wait patiently while the file is being processed. The text "Loading.." will be displayed below the "Make Prediction" button to indicate that the CSV file is still being processed.

Figure 8: Loading Message upon Making Prediction

Once the prediction process is completed, the "Loading.." message will be replaced by another message depending on different cases.

Prediction Case 1: The format of uploaded csv file without missing values aligns to the required format

The "Loading..." message will be replaced with **"Prediction completed successfully"** once the process is finished. The predicted results will then be displayed in tabular and visual formats in the respective sections.

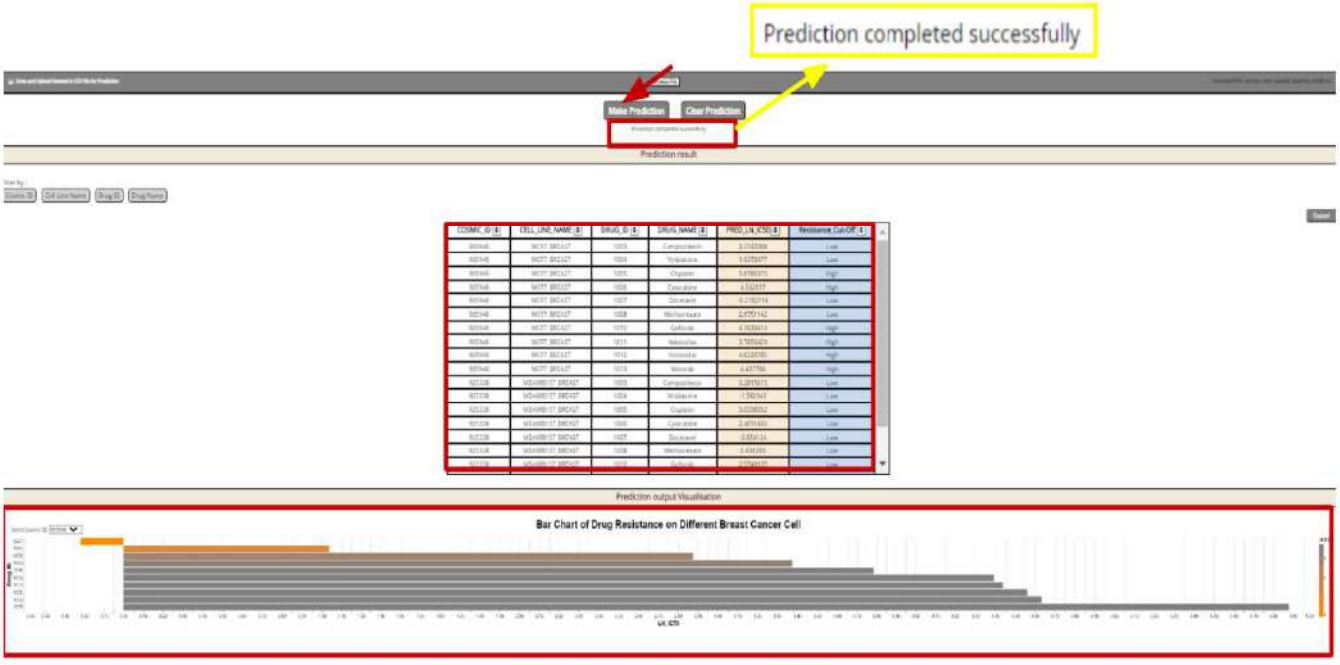


Figure 9: Shows Prediction upon Completion

Prediction Case 2: The uploaded csv file contains missing value

Once the prediction process is complete, the "Loading..." message will be replaced with "**Prediction completed, Rows with NA values have been dropped.**" This signifies that our server has removed any missing values from the uploaded CSV file and proceeded with the prediction as usual. The predicted results will still be presented in both tabular and visual formats in their respective sections.

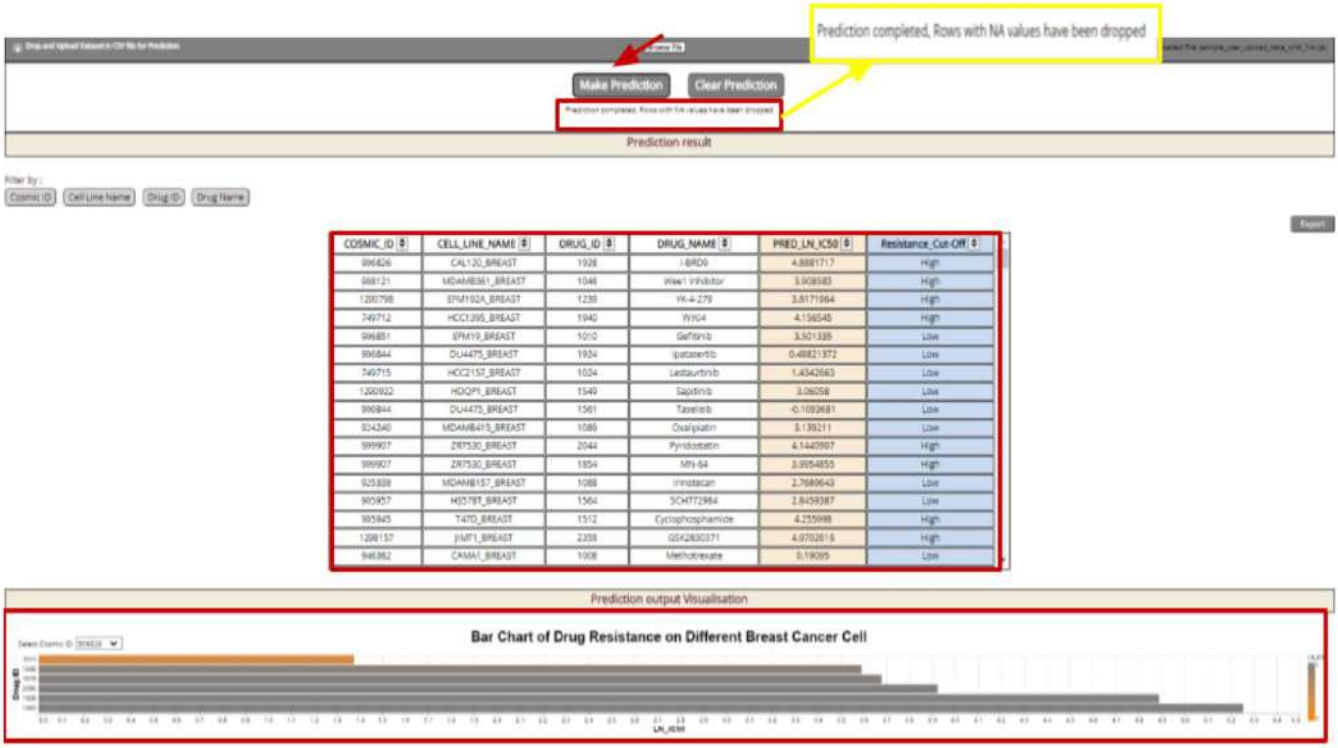


Figure 10: Printed Message and Outputs for Datasets containing NA(missing) values

Prediction Case 3: The format of uploaded csv file does not follow the required format (wrong dimension or contains non-numeric value within the 5th column - 19225th column)

Error occurs during the prediction process due to incorrect dimensions. The message "**Error during prediction: Wrong Dimension Data or Non-numeric Values Detected, Please Input Correct Data**" will replace the "Loading..." message. This indicates that our server cannot process the uploaded CSV file properly. Consequently, no predicted results will be generated, and both the "Prediction Output" and "Prediction Output Visualisation" sections will remain empty. Users will be suggested to refer to Section 1.3: User Upload Data Format Requirements and preprocess the dataset accordingly, ensuring that it adheres to the specified format outlined in the guidelines provided.

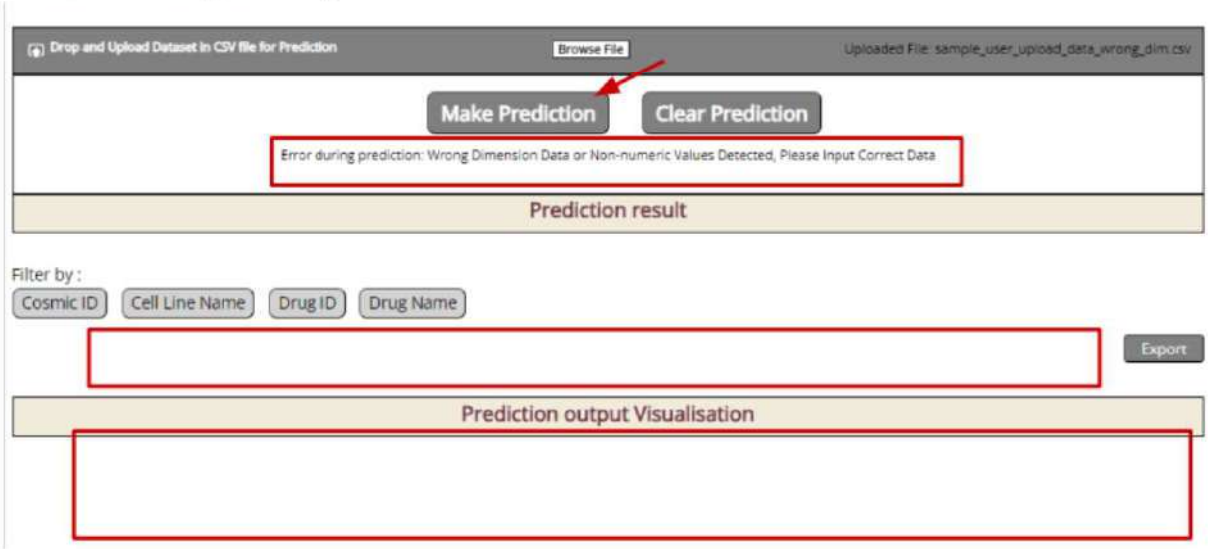


Figure 11: Printed Message and Outputs for Wrong Dimension Dataset

Step 4 (Optional): Click on the **"Clear Prediction"** button

Once users have obtained the desired information after the prediction process, they have the option to click the "Clear Prediction" button to discard the previously generated predictions. Upon clicking this button, both the "Prediction Output" and "Prediction Output Visualisation" sections will be cleared, removing all prediction results, tables, and visualisations from the interface. This action allows users to reset the application and initiate new predictions or perform other tasks as needed.

Section 1.2.3: Interactive Features of Cell-Drug Resistance Table

The web application "table section" offers four interactive features designed to enhance user interaction with the Cell-Drug Resistance Table. These features include the **filter** function, **sorting** function, **search** function, and **export** function. Each feature serves a distinct purpose and empowers users to manipulate and analyse the data within the table according to their preferences and requirements.

Filter function

The filter function offers users a convenient way to extract specific data from the table by applying filter values. Upon clicking the filter button, a dropdown menu appears, presenting users with a list of checkboxes corresponding to different filter options. With four filter buttons available, users can selectively filter data based on various columns, facilitating efficient retrieval of desired information.

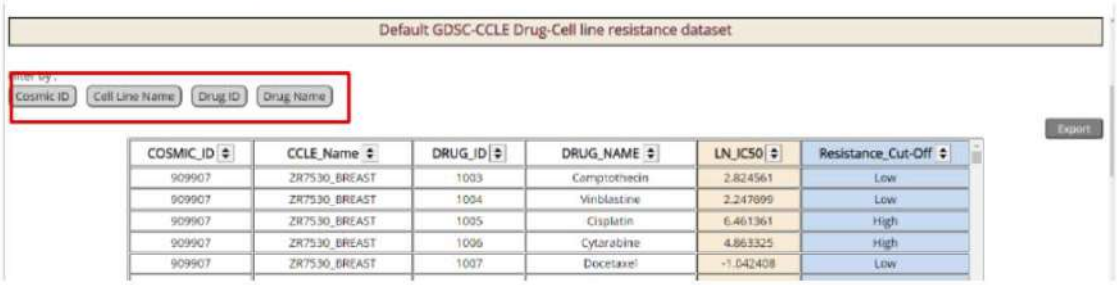


Figure 12: Shows the Filter Buttons

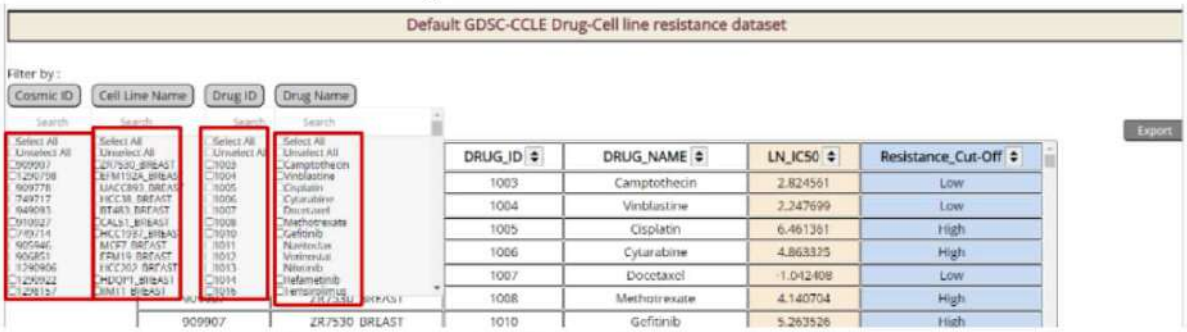


Figure 13: Shows the Filter Options for each Filter Buttons

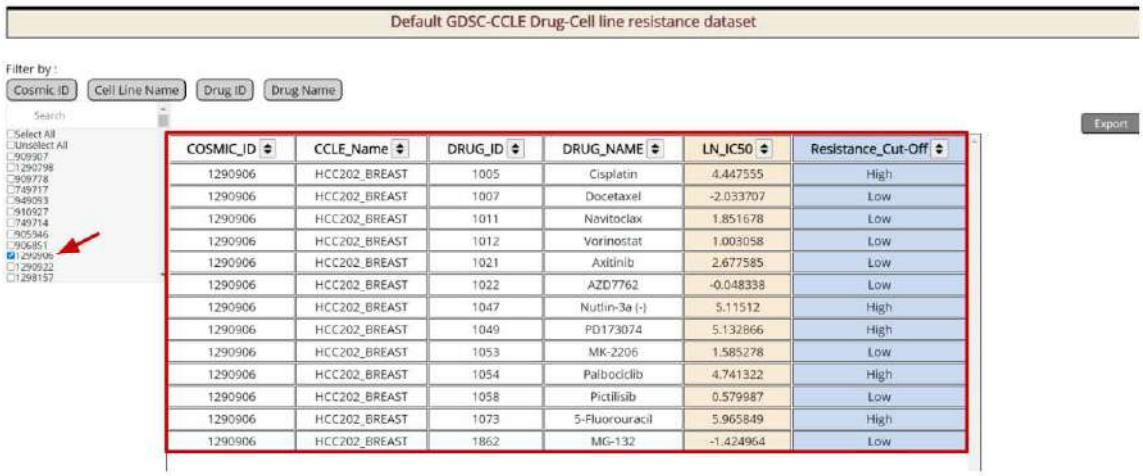
For example:

By default, the table displays the full data shown below.

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low
909907	ZR7530_BREAST	1008	Methotrexate	4.140704	High
909907	ZR7530_BREAST	1010	Gefitinib	5.263526	High
909907	ZR7530_BREAST	1011	Navitoclax	5.557596	High
909907	ZR7530_BREAST	1012	Vorinostat	3.636889	Low
909907	ZR7530_BREAST	1013	Nilotinib	2.086747	Low
909907	ZR7530_BREAST	1014	Refametinib	5.193305	High
909907	ZR7530_BREAST	1016	Temsirolimus	1.930275	Low
909907	ZR7530_BREAST	1017	Olaparib	5.944279	High
909907	ZR7530_BREAST	1018	Veliparib	6.189043	High
909907	ZR7530_BREAST	1019	Bosutinib	5.400951	High
909907	ZR7530_BREAST	1020	Lenalidomide	6.843033	High

Figure 14: Shows the default dataset table display

When users select the checkbox corresponding to a specific COSMIC_ID (e.g., COSMIC_ID = 1290906), the table will automatically filter and display only the data entries associated with that particular COSMIC_ID (in this case, COSMIC_ID = 1290906). This functionality ensures that users can quickly access and view data specific to their selected COSMIC_ID without the need for manual sorting or searching.



COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
1290906	HCC202_BREAST	1005	Cisplatin	4.447555	High
1290906	HCC202_BREAST	1007	Docetaxel	-2.033707	Low
1290906	HCC202_BREAST	1011	Navitoclax	1.851678	Low
1290906	HCC202_BREAST	1012	Vorinostat	1.003058	Low
1290906	HCC202_BREAST	1021	Axitinib	2.677585	Low
1290906	HCC202_BREAST	1022	AZD7762	-0.048338	Low
1290906	HCC202_BREAST	1047	Nutlin-3a (-)	5.11512	High
1290906	HCC202_BREAST	1049	PD173074	5.132866	High
1290906	HCC202_BREAST	1053	MK-2206	1.585278	Low
1290906	HCC202_BREAST	1054	Palbociclib	4.741322	High
1290906	HCC202_BREAST	1058	Pictilisib	0.579987	Low
1290906	HCC202_BREAST	1073	5-Fluorouracil	5.965849	High
1290906	HCC202_BREAST	1862	MG-132	-1.424964	Low

Figure 15: Shows Filtering Process for COSMIC ID

Users have the flexibility to apply the functions of two or more filter buttons simultaneously. For instance, if users select the checkbox corresponding to a specific COSMIC_ID and a specific Drug_ID,

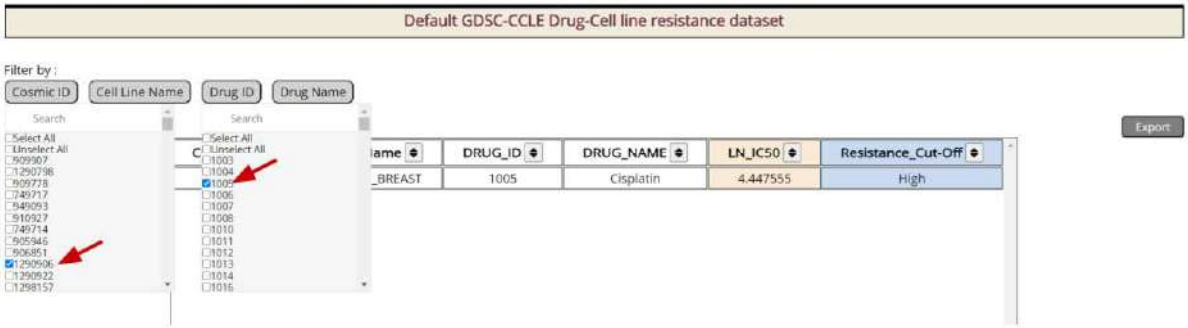


Figure 16: Shows Filtering Process for >1 Filters Applied

The table will directly display only the data with the selected COSMIC_ID and DRUG_ID.

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
1290906	HCC202_BREAST	1005	Cisplatin	4.447555	High

Figure 17: Shows Filtering Process Completion

Within the dropdown menu, there is a convenient "Select All" checkbox. Upon ticking this checkbox, all checkboxes within the same dropdown will be automatically selected. This feature also streamlines the process for users who wish to exclude only a few specific data points. Similarly, within the dropdown menu, there exists a checkbox labelled "Unselect All". By selecting this checkbox, users can efficiently clear all selected checkboxes within the dropdown. This feature is particularly useful for users who wish to reset the filter function and start anew.

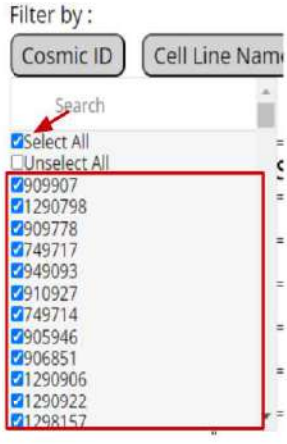


Figure 18: Shows Select All Option

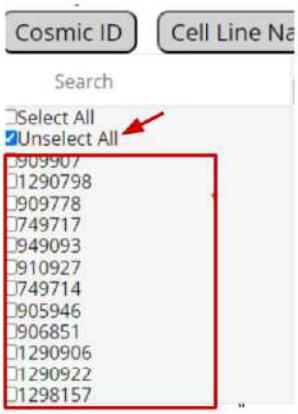


Figure 19: Shows Unselect All Option

Sort function

The sort function allows users to arrange the data either in ascending or descending order, facilitating quick and efficient access to desired information based on the initial character of the data value. Positioned alongside each header title, six sort buttons are available, enabling users to organise specific data according to various column criteria.

For example:

By default,all data will be arranged based on the original dataset/uploaded csv file.

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low
909907	ZR7530_BREAST	1008	Methotrexate	4.140704	High
909907	ZR7530_BREAST	1010	Gefitinib	5.263526	High
909907	ZR7530_BREAST	1011	Navitoclax	5.557596	High
909907	ZR7530_BREAST	1012	Vorinostat	3.636889	Low
909907	ZR7530_BREAST	1013	Nilotinib	2.086747	Low
909907	ZR7530_BREAST	1014	Refametinib	5.193305	High
909907	ZR7530_BREAST	1016	Temsirolimus	1.930275	Low
909907	ZR7530_BREAST	1017	Olaparib	5.944279	High

Figure 20: Default Dataset

If the users click the sort button that is located beside the specific header, the data will be arranged based on the value of the selected column in **ascending** order (for example, DRUG_ID).

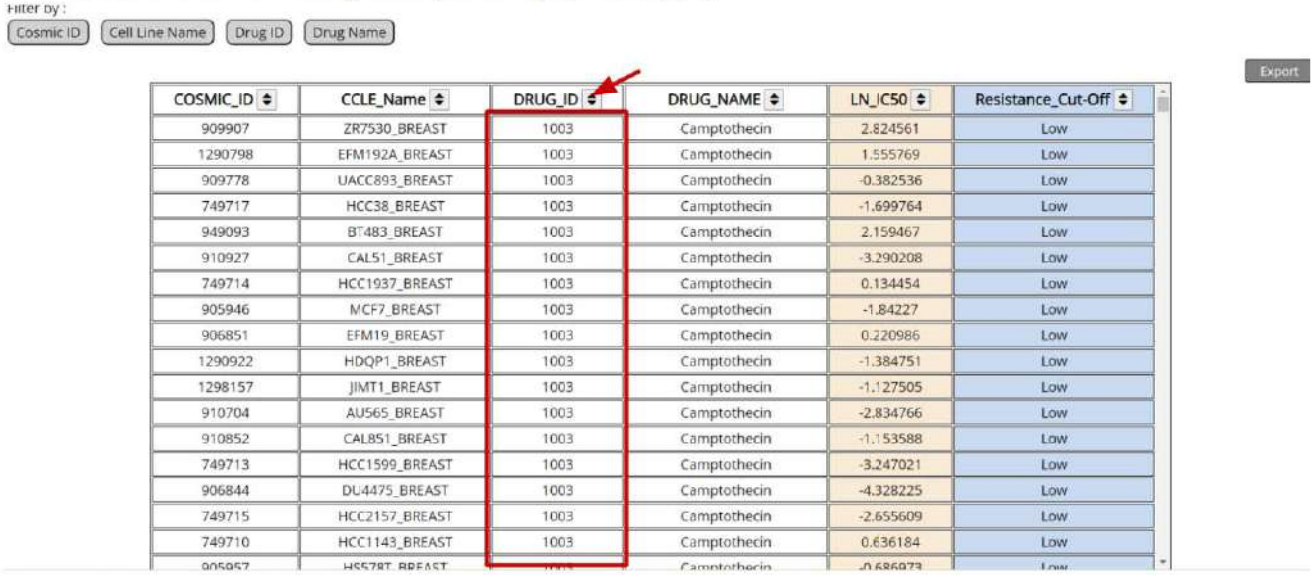
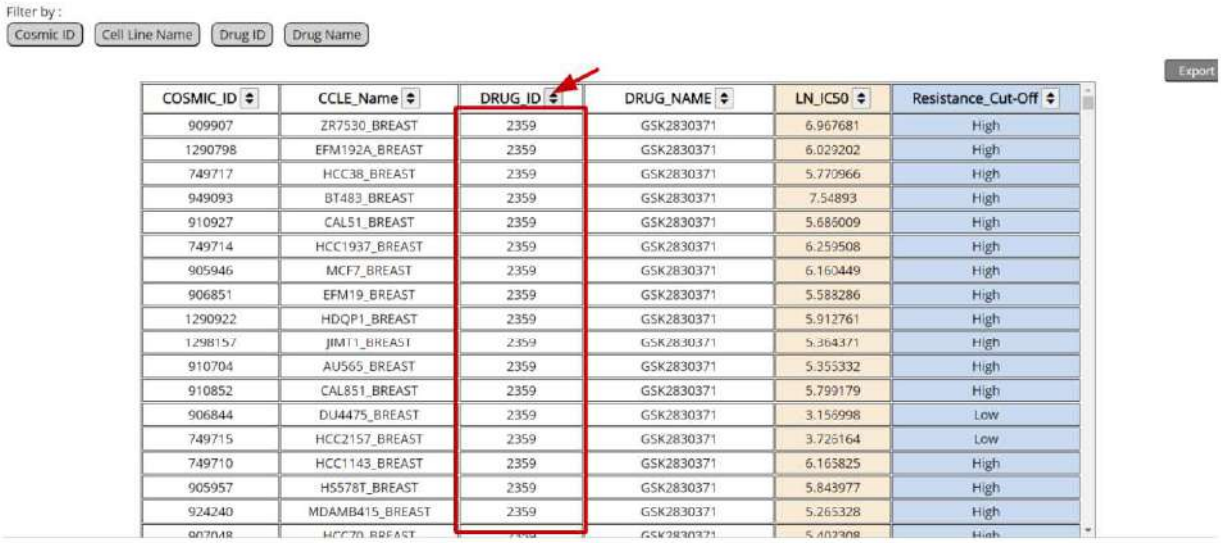


Figure 21: Shows Sort Button for DRUG_ID in Ascending Order

If the users click the **same** sort button again, the data will be arranged based on the value of Drug_ID in **descending** order.



COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	2359	GSK2830371	6.967681	High
1290798	EFM192A_BREAST	2359	GSK2830371	6.029202	High
749717	HCC38_BREAST	2359	GSK2830371	5.770966	High
949093	BT483_BREAST	2359	GSK2830371	7.54893	High
910927	CAL51_BREAST	2359	GSK2830371	5.686009	High
749714	HCC1937_BREAST	2359	GSK2830371	6.259508	High
905946	MCF7_BREAST	2359	GSK2830371	6.160449	High
906851	EFM19_BREAST	2359	GSK2830371	5.588286	High
1290922	HDQP1_BREAST	2359	GSK2830371	5.912761	High
1298157	JIMT1_BREAST	2359	GSK2830371	5.364371	High
910704	AU565_BREAST	2359	GSK2830371	5.355332	High
910852	CAL851_BREAST	2359	GSK2830371	5.799179	High
906844	DU4475_BREAST	2359	GSK2830371	3.156998	Low
749715	HCC2157_BREAST	2359	GSK2830371	3.726164	Low
749710	HCC1143_BREAST	2359	GSK2830371	6.165825	High
905957	HS578T_BREAST	2359	GSK2830371	5.843977	High
924240	MDAMB415_BREAST	2359	GSK2830371	5.265328	High
907048	HCC70_BREAST	2359	GSK2830371	5.402308	High

Figure 22: Shows Sort Button for DRUG_ID in Descending Order

Search function

This functionality empowers users to search for specific checkbox values to filter the data efficiently. Users can easily find specific column values by entering a few characters of the desired value. Positioned above all checkboxes, four search fields are provided, enabling users to search for specific checkbox values and filter data based on different columns with ease. The dropdown menu initially presents all the checkboxes corresponding to the specific column, ensuring each checkbox holds a unique value. If users enter the initial characters of the values of interest, for example COSMIC_ID starting with "129", only the checkbox values beginning with "129" will appear in the dropdown menu.

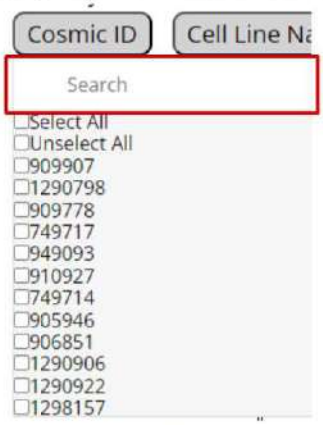


Figure 23: Shows Search Tab

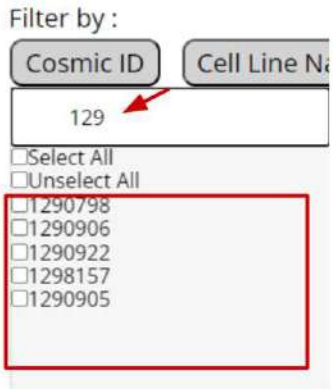
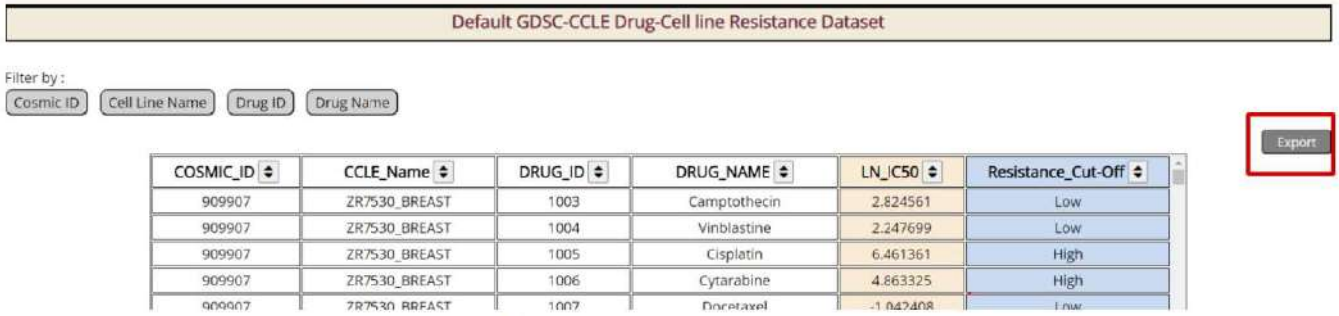


Figure 24: Shows Options given in Search

Export function

The export function allows users to download the dataset currently displayed in the table section as a CSV file. This feature simplifies the process of obtaining dataset information for users' further analysis.



COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low

Figure 25: Shows Export Button

When the export button is clicked, a csv file with filename 'LN_IC50' will be exported to your local explorer.

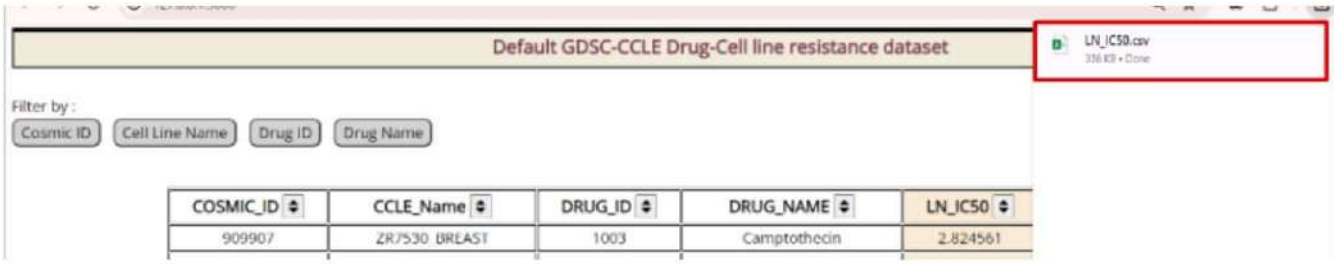


Figure 26: Shows the output file has been exported and downloaded in the device

Section 1.2.4: Interactive Visualisation Analysis Guide

Below is a sample of the bar chart visualisation provided in "DRUGRES":

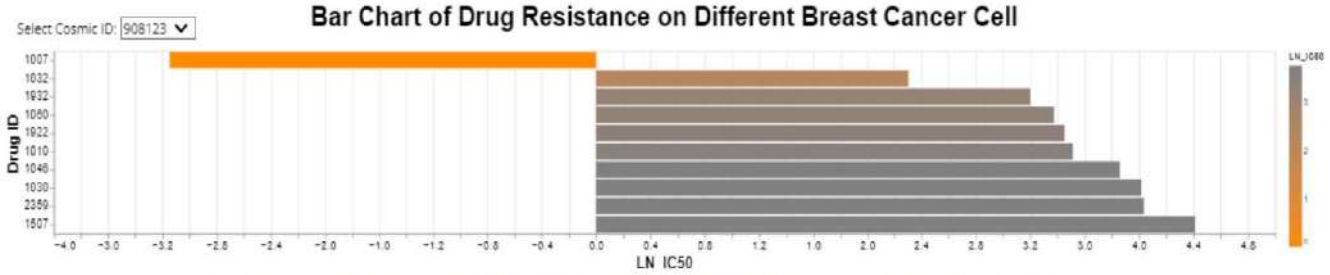


Figure 27: shows a sample of the bar chart visualisation provided in "DRUGRES":

Both the "User Manual Prediction" and "Default Dataset" options will feature a bar chart, offering visual representation to showcase Drug Resistance across various Cancer Cell types. The **x-axis** will represent the drug resistance "**LN_IC50**," while the **y-axis** will denote the **drug_id**.

Discussions and Analysis of Insights Gained from the Visualisation

(1) The visualisation presents bars sorted in **ascending** order. This means that drugs with the **lowest LN_IC50** values will be positioned at the **top** of the bar chart, while those with the **highest LN_IC50** values for a particular cancer type will appear at the **bottom**.

(2) The **legend** indicates the colour scheme used in the bars: shades of **orange** represent **lower resistance** to the specific cancer cell, while shades of **grey** indicate **higher resistance**. The colours follow a **gradient** pattern, with **darker shades** of **orange** indicating **greater efficacy** of the drug for treating the particular cancer cell, whereas **darker shades** of **grey** suggest **poorer performance** in treating the specific cancer cell.

(3) By examining the position and colour of the bars representing each drug in the chart, users can therefore readily identify which drugs are more appropriate for treating the particular cancer cell line. Drugs positioned at the **top**, appearing **more orange**, are **better** suited for treatment, while those at the **bottom**, appearing **more grey**, are less effective and **not recommended for treatment**.

Interactive Components

(a) There is a **filter** function which enables users to switch between bar charts for various types of breast cancer cells. This functionality allows users to view unique bar charts tailored to different cancer cells, visualising the LN_IC50 values of different drugs for each specific cell line.

For example:

By default it is displaying the visualisation of the prediction result of COSMIC_ID = 208123

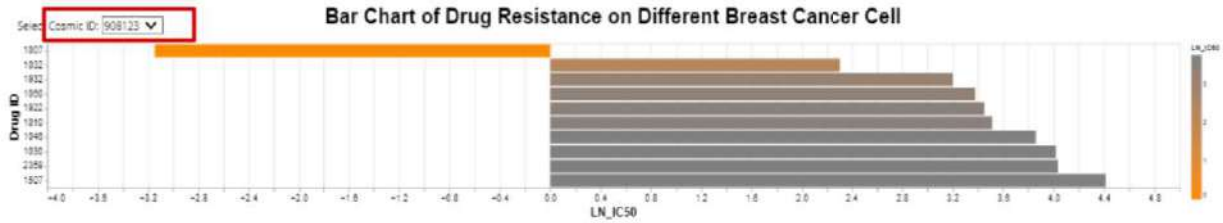


Figure 28: shows the filter component of the Bar Chart

If the user selects another cancer type:

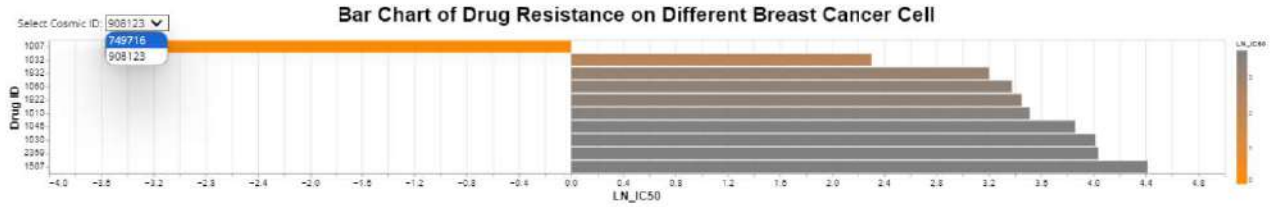


Figure 29: shows the filter component of the Bar Chart

It will now display the bar chart of the prediction result for COSMIC_ID = 749716

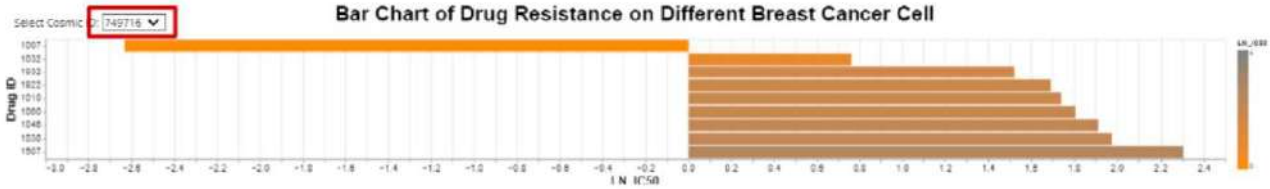


Figure 30: shows the filter component of the Bar Chart

(b) **Tooltip** displays will accompany the bars as users **hover** over them. Each tooltip will be unique to its corresponding bar, presenting relevant information specific to that bar.

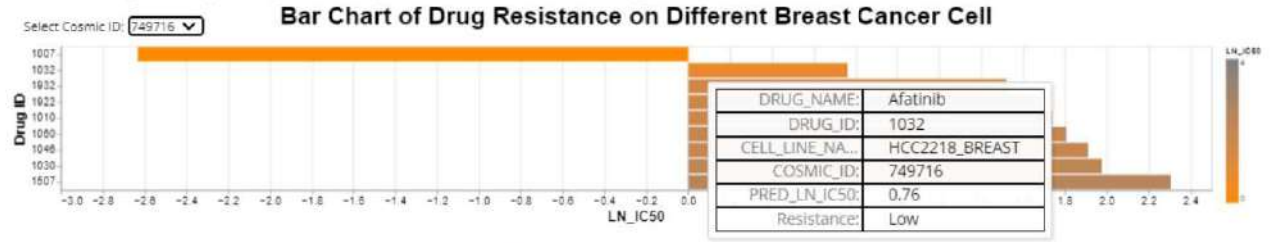


Figure 31: shows the tooltip display of the Bar Chart

Section 1.3: User Upload Data Format Requirements

The uploaded dataset should strictly adhere to the provided format. For reference, the link below provides a sample file illustrating the format of the uploaded dataset by user used for manual prediction.
https://drive.google.com/file/d/1xx2n8QtVBoP5IdJVbuwrq-jYnWWvXjIj/view?usp=sharing

```
In [13]: 1 user_df
```

Out[13]:

	DRUG_NAME	DRUG_ID	COSMIC_ID	CELL_LINE_NAME	TSPAN6 (7105)	TNMD (64102)	DPM1 (8813)	SCYL3 (57147)	C1orf112 (55732)	FGR (2268)	...
0	Linsitinib	1510	905946	MCF7_BREAST	2.403268	0.000000	7.490249	2.606442	3.177918	0.014355	...
1	Afuresertib	1912	906851	EFM19_BREAST	3.485427	0.028569	6.984134	2.560715	3.806324	0.485427	...
2	SCH772984	1564	910910	UACC812_BREAST	2.482848	0.000000	7.835545	2.266037	2.707083	0.097611	...
3	UMI-77	1939	905946	MCF7_BREAST	2.403268	0.000000	7.490249	2.606442	3.177918	0.014355	...
4	MN-64	1854	909907	ZR7530_BREAST	3.472488	0.000000	5.959306	3.878725	3.646163	0.000000	...
...	...	...	...	...	...	...	...	...	...	...	...
1022	CCT007093	1067	905951	BT549_BREAST	3.390943	0.000000	7.142924	2.169925	4.195348	0.014355	...
1023	Alisertib	1051	909907	ZR7530_BREAST	3.472488	0.000000	5.959306	3.878725	3.646163	0.000000	...
1024	P22077	1933	749715	HCC2157_BREAST	3.523562	0.000000	6.872459	2.776104	3.733354	0.014355	...
1025	Refametinib	1014	749716	HCC2218_BREAST	3.587365	0.000000	6.563463	2.636915	3.640390	0.014355	...
1026	AZD6482	2169	910927	CAL51_BREAST	5.705425	0.000000	6.413289	3.298658	4.052242	0.124328	...

1027 rows × 19226 columns

Figure 32: shows the dimensions of uploaded dataset

Requirements:

Req 1: The uploaded dataset MUST contains **19226 columns** of data, which are made up of the columns:

(a) **One** column of "**DRUG_NAME**"
(b) **One** column of "**DRUG_ID**"
(c) **One** column of "**COSMIC_ID**"
(d) **One** column of "**CELL_LINE_NAME**"
(e) **19221** columns of CCLE Gene Expressions (refer to DepMap Public 22Q2 CCLE_expression.csv)
(f) **One** column of "**isosmiles**"

Req 2: The **column names** for **(a), (b), (c), (d),** and **(f)** must exactly match the specified column names in **Req1**.

Req 3: All values of "**DRUG_ID**", "**COSMIC_ID**" and **Gene Expressions (5th column to 19225th column)**need to be in **numeric** format.

Req 4: For the columns representing gene expressions level for different genes, **NA** values are acceptable.

Req 5: The upload file should be in **csv** format.

The technical guide for preparation of user upload data is provided in Section 2.2: User Upload Data Preprocessing Guide below.

Notes:

(1) If the dataset format does not meet the specified requirements, an error message will notify the user upon clicking the "Make Prediction" button. The message will indicate that the uploaded dataset does not align with the specified format. Please review the dataset to ensure no columns are missing or extra columns are added.
(2) Any rows in the user-uploaded dataset containing NA values will be removed before making predictions. This is done to prevent bias in the predictions.
(3) It is recommended that the cell lines in the user-uploaded dataset are "Breast Cancer" type, as our software tool currently focuses exclusively on predicting drug resistance in breast cancer cell lines.

Reasonings:

➢ The user-uploaded data must have the same columns as the provided sample file to match the dimensions of the drug resistance predictive model. This model is trained using 19,221 gene expressions and drug chemical compound structures (isosmiles). The user-uploaded data will be inputted into the predictive model for making predictions, therefore, it requires the same number of features as the training dataset used to train the predictive model. Also, the training features must be in numeric format, therefore, there is a restriction of data types, particularly among the 5th column to 19225th column of the user upload data, which are the gene expressions.

Section 1.4: Abbreviations and Definitions

Terminology 1: Isosmiles

IsoSMILES, short for "Isomeric Simplified Molecular Input Line Entry System," is a variant of the Simplified Molecular Input Line Entry System (SMILES) notation used to represent chemical structures in a textual format. Like SMILES, IsoSMILES represents molecules as a linear string of characters, where each character or combination of characters represents specific atoms, bonds, and structural features. However, IsoSMILES includes additional information about the isomeric forms of molecules, particularly stereochemistry, which is critical for representing three-dimensional arrangements of atoms in the molecule accurately. This extension allows IsoSMILES to capture the structural diversity of isomeric compounds more comprehensively, making it particularly useful in fields such as cheminformatics, drug discovery, and computational chemistry where accurate representation of molecular structures is essential. In our case, isosmiles represents the **chemical molecular compound structure for different drug compounds**.

Terminology 2: LN_IC50

LN_IC50 refers to the natural logarithm of the half-maximal inhibitory concentration (IC50) value. In pharmacology and drug discovery, IC50 represents the concentration of a drug needed to inhibit a biological process, typically by 50%. LN_IC50 is often used as a measure of drug potency against a specific target or cell line, such as cancer cells. It quantifies the logarithmic relationship between the concentration of a drug and its inhibitory effect on a biological system. A **lower LN_IC50** value indicates a more potent drug, reflecting **higher sensitivity** of the cancer cell line to the drug and **lower resistance**. (Li et al., 2021) Conversely, a higher LN_IC50 value suggests lower drug sensitivity and higher resistance of the cancer cells to the drug. Therefore, LN_IC50 serves as a crucial parameter in evaluating the efficacy of drugs and their potential therapeutic applications in cancer treatment.

Terminology 3: Resistance Cut-Off

The resistance cut-off, also known as the IC50 cutoff value or LN_IC50 cut off point, is a predetermined threshold used to classify drugs based on their sensitivity or resistance to a specific biological process or cell line, such as cancer cells. In the context of the provided document, the resistance cut-off refers to the IC50 concentration value that distinguishes between drug sensitivity and resistance in patient-derived tumour samples. In this case, the optimal IC50 cutoff value for drug sensitivity in patient-derived tumour samples is determined to be 43.26 mmol/L (Tang et al., 2023), corresponding to a natural logarithm (LN_IC50) value of approximately 3.77. **LN_IC50 values below 3.77 indicate drugs with high sensitivity** to the cancer cell line, suggesting low resistance. Conversely, **LN_IC50 values equal or above 3.77 suggest low sensitivity** to the cancer cell line, indicating high resistance to the drug. Therefore, the resistance cut-off serves as a critical threshold for evaluating the efficacy of drugs in cancer treatment, helping to guide clinical decision-making regarding drug selection and treatment strategies.

Section 1.5: Default GDSC-CCLE Cell-Drug Resistance Table/ Prediction Output Discussion and Analysis

Below is a sample of first five rows of the **Default GDSC-CCLE Cell-Drug Resistance Table**:

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low

Figure 33: shows a sample of first five rows of the Default GDSC-CCLE Cell-Drug Resistance Table

Below is the first five rows of the sample **prediction output** generated from random predictions:

COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_IC50	Resistance_Cut-Off
1290922	HDQP1_BREAST	1004	Vinblastine	-3.2854412	Low
1290922	HDQP1_BREAST	1005	Cisplatin	2.9929354	Low
1290922	HDQP1_BREAST	1006	Cytarabine	1.3996433	Low
1290922	HDQP1_BREAST	1007	Docetaxel	-3.996065	Low
1290922	HDQP1_BREAST	1008	Methotrexate	-1.7595619	Low

Figure 34: shows first five rows of the sample prediction output

The tables provide information on drug-cell resistance relationships, with the headers:

(a) **COSMIC_ID**: Unique identifier of the cancer cell line
(b) **CCLE_Name**/ CELL_LINE_NAME: The name of the cancer cell line
(c) **DRUG_ID**: Unique identifier of the drug, which is also known as GDSC Drug ID
(d) **DRUG_NAME**: The name of the drug
(e) **LN_IC50/ PRED_LN_IC50**: The natural logarithm of the "Half-maximal Inhibitory Concentration (IC50)" value, indicating the resistance/ sensitivity of the drug to the cancer cell line
(f) **Resistance_Cut-Off:** The cut-off point of LN_IC50, distinguishing between high resistance and low resistance effects of the drug on the cell line.

Discussions and Analysis of Insights Gained from the Tables

With reference to definition of LN_IC50 value and Resistance_Cut-Off provided in Section 1.3: Abbreviations and Definitions,

(1) After the user submits their prediction dataset, initiating the "Make Prediction" action yields **predictions for the "PRED_LN_IC50"** value and provides the "Resistance_Cut-Off" to help determine whether the drug is sensitive to the cell or not.
(2) Users can obtain insights into the drug resistance of specific drugs on a particular cell line by examining the LN_IC50 value. A **lower LN_IC50** value signifies **greater drug sensitivity** and **lower resistance** of the drug on the cancer cell, suggesting that the drug may be **effective** in treating the respective cancer cell.
(3) On the contrary, if a drug exhibits a **high LN_IC50** value on a specific cell line, it indicates **low drug sensitivity** and **high resistance**, suggesting potential **ineffectiveness** in treating the cancer cell using that drug.
(4) Previously, it was understood that LN_IC50 values at or below 3.77 indicate drugs with high sensitivity to the cancer cell line (low resistance), whereas LN_IC50 values above 3.77 suggest low sensitivity to the cancer cell line (high resistance). By considering the "Resistance_Cut-Off", drugs with a **low "Resistance_Cut-Off"** are **more effective** for treating the paired cell line, whereas drugs with a high "Resistance_Cut-Off" may not be suitable for treating the respective cancer cell using that drug.
LN_IC50 < 3.77 : Low Resistance
LN_IC50 >= 3.77: High Resistance

Section 1.6: Data Source

Data Source 1: Genomics of Drug Sensitivity in Cancer (GDSC)

The Genomics of Drug Sensitivity in Cancer (GDSC) database, specifically the GDSC2-dataset shown in **Figure 35**, encompasses drug response data for 969 cancer cell lines and 295 drugs. This comprehensive resource integrates drug resistance information with genomic data, including somatic mutations in cancer genes, gene amplification and deletion, tissue type, and transcriptional data (Yang et al., 2012). The GDSC is instrumental for model training and development aimed at predicting drug resistance in cancer cell lines such that it aids in identifying molecular markers and patterns that predict responses to anti-cancer drugs, streamlining the search for new therapeutic biomarkers in cancer treatment. The GDSC was selected as the basis for developing the predictive model due to its distinct drug screening assays and relatively smaller sample size for most drugs (Sharifi-Noghabi et al., 2021). The dataset includes crucial data for researching drug sensitivity, such as LN_IC50, which serves as the target attribute which can be employed for training and pattern recognition, crucial for developing an accurate predictive model for drug resistance in cancer cell lines.

DATASET	NLME_RES	NLME_CU	COSMIC_I	CELL_LINE	SANGER_I	TCGA_DES	DRUG_ID	DRUG_NA	PUTATIVE	PATHWAY	COMPANY	WEBRELE/	MIN_CON	MAX_CON	LN_IC50	AUC	RMSE	Z_SCORE
GDSC2	343	15946310	683667	PFSK-1	SIDM0113	MB	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-1.46389	0.93022	0.089052	0.433123
GDSC2	343	15946548	684052	A673	SIDM0084	UNCLASSII	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-4.86946	0.61497	0.111351	-1.4211
GDSC2	343	15946830	684057	ES5	SIDM0026	UNCLASSII	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-3.36059	0.791072	0.142855	-0.59957
GDSC2	343	15947087	684059	ES7	SIDM0026	UNCLASSII	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-5.04494	0.59266	0.135539	-1.51665
GDSC2	343	15947369	684062	EW-11	SIDM0020	UNCLASSII	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-3.74199	0.734047	0.128059	-0.80723
GDSC2	343	15947651	684072	SK-ES-1	SIDM0111	UNCLASSII	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-5.14296	0.582439	0.137581	-1.57002
GDSC2	343	15947932	687448	COLO-829	SIDM0090	SKCM	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-1.23503	0.867348	0.09347	0.557727
GDSC2	343	15948212	687452	5637	SIDM0080	BLCA	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-2.63263	0.834067	0.076169	-0.20322
GDSC2	343	15948491	687455	RT4	SIDM0108	BLCA	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-2.96319	0.821438	0.094466	-0.3832
GDSC2	343	15948772	687457	SW780	SIDM0116	BLCA	1003	Camptothe	TOP1	DNA replic	1046	Y	0.0001	0.1	-1.44914	0.90505	0.074109	0.441154

Figure 35: Genomics of Drug Sensitivity in Cancer Dataset

Data Source 2: Cancer Cell Line Encyclopedia (CCLE)

The CCLE-dataset, shown in **Figure 36**, contains gene expression data between 19221 genes and 1406 cancer cell lines for 33 primary diseases. It contains a collection of gene expression, chromosomal copy number, and high-throughput sequencing data from 1406 human cancer cell lines(Barretina et al, 2012). CCLE was selected to merge with GDSC via Cosmic ID, creating a combined GDSC-CCLE dataset. This approach reduces heterogeneity issues and allows for the extraction of gene expression data specific to breast cancer, aligning with our project scope and mapping these expressions to their corresponding LN_IC50 values. By providing unique gene expressions that represent different cancer cell lines, the CCLE facilitates the distinct characterization of cells, essential for model training and development.

	TSPAN6 (7	TNMD (64	DPM1 (881	SCYL3 (571	C1orf112 (	FGR (2268	CFH (3075	FUCA2 (25	GCLC (272	NFYA (480	STPG1 (905	NIPAL3 (5	LAS1L (81	ENPP4 (22	SEMA3F (6	CFTR (108	ANKIB1 (5	CYP51A1 (	KRIT1 (889	RAD52 (58	BAD (572)	LAP3 (5105
ACH-0011	4.331092	0	7.364397	2.792855	4.470537	0.028569	1.226509	3.042644	6.499686	4.739848	3.451541	4.181103	5.035184	2.280956	3.218781	0	5.084064	6.646883	5.561326	3.805292	5.159871	4.783457
ACH-0012	4.566815	0.584963	7.106537	2.543496	3.50462	0	0.189034	3.813525	4.221104	3.481557	1.903038	3.257011	4.95977	3.621759	1.257011	0.028569	5.122633	7.13904	4.991408	2.969012	5.543805	6.333603
ACH-0013	3.15056	0	7.379032	2.333424	4.227279	0.056584	1.35034	6.687061	3.682573	3.273516	2.881665	3.494416	4.331992	0.790772	0.275007	0.042644	4.078097	5.653347	4.754888	2.632268	4.549053	6.16088
ACH-0015	5.08534	0	7.154109	2.545968	3.084064	0	5.868143	6.165309	4.489928	3.956986	2.467279	4.41819	5.557042	3.485427	2.983678	0.028569	3.83996	6.187253	3.65764	2.689299	5.537607	5.244507
ACH-0002	6.729145	0	6.537607	2.456806	3.867896	0.799087	7.208381	5.569856	7.127014	4.568032	1.933573	2.620586	5.135863	0.879706	5.829596	0.070389	4.309613	6.794156	4.840463	3.107688	6.119979	5.71177
ACH-0007	4.272023	0.189034	7.022923	2.555816	3.841973	0	0.097611	4.888013	4.926474	3.526069	2.505891	3.760221	5.137504	2.671293	1.970854	6.641691	3.764474	6.270155	4.621173	2.867896	4.85997	5.320485
ACH-0003	3.337711	0	5.927185	1.944858	2.678072	0.014355	3.089159	6.011227	3.642702	2.879706	1.521051	3.77926	4.105175	3.280956	0.378512	0.815575	3.357552	6.295356	4.191405	2.752749	4.599913	5.781097
ACH-0002	0.056584	0	6.093602	3.970854	3.731183	0.028569	6.092969	3.033863	3.422233	4.725741	3.690417	3.82069	5.247548	3.207893	0.454176	0	4.64039	5.327687	4.898208	4.076388	5.267536	5.839708
ACH-0004	4.01614	0	6.533875	2.226509	3.02148	0.028569	0.084064	5.588265	6.380418	3.864929	2.521051	3.053111	5.80271	2.389567	2.893362	0	3.982765	5.000451	4.146046	2.99458	5.30163	5.811214
ACH-0007	4.411426	0	6.412443	2.364572	4.275087	0.042644	0.201634	2.563158	6.104127	4.585563	2.713696	3.667892	5.1815	2.82985	3.244887	0.014355	4.275752	5.273516	4.671859	3.617063	5.638074	4.622345
ACH-0017	3.871844	0.056584	6.763943	1.937344	3.152183	0.62293	6.944624	7.088735	4.242603	3.339137	3.929791	6.301039	4.905928	1.664483	1.744161	0.014355	3.689299	4.296457	3.687061	2.568032	6.687341	7.188737

Figure 36: Cancer Cell Line Encyclopedia Dataset

Data Source 3: PubCHEM

The PubChem database, shown in **Figure 37**, contains 118 million drug compounds and 318 million drug substances. It includes data from the Substance, Compound, and BioAssay databases (Kim et al., 2015). PubChem features the Isomiles attribute, which distinguishes different drugs by acting as a compound molecular identifier. This aids in pattern recognition during model training by applying Morgan Fingerprints that convert drug features to 256-bit vectors. Additionally, PubChem's drug ID facilitates mapping to the corresponding Isomiles in the combined GDSC-CCLE dataset. Chosen for its comprehensive drug information, PubChem supports training and drug recognition.

cid	cmpdnam	cmpdsyno	mw	mf	polararea	complexity	xlogp	heavycnt	hbonddon	hbondacc	rotbonds	inchi	isomiles	canonicals	inchikey	iupacname	exactmass	monoisotc	charge	covalentur	isotopeatc	totalatom	de
1	Acetylcam	Acetyl-DL-	203.24	C9H17NO4	66.4	214	0.4	14	0	4	5	InChI=1S/(	CC(=O)OC(	CC(=O)OC(	RDHQFKQI	3-acetylox	203.116	203.116	0	1	0	1	
6	1-Chloro-2	1-chloro-2	202.55	C6H3ClN2	91.6	224	2.3	13	0	4	0	InChI=1S/(	C1=CC(=C(	C1=CC(=C(	VYZAHLCB	1-chloro-2	201.978	201.978	0	1	0	0	
11	1,2-Dichlo	1,2-dichlor	98.96	C2H4Cl2	0	6	1.5	4	0	0	1	InChI=1S/(	C(CCl)Cl	C(CCl)Cl	WSLDOOZ	1,2-dichlor	97.969	97.969	0	1	0	0	
34	2-Chloroet	2-chloroet	80.51	C2H5ClO	20.2	10	-0.1	4	1	1	1	InChI=1S/(	C(CCl)O	C(CCl)O	SZIFAVKTN	2-chloroet	80.0029	80.0029	0	1	0	0	
38	2-Dehydro	2-dehydro	146.14	C6H10O4	74.6	159	-0.1	10	2	4	3	InChI=1S/(	CC(C)(CO)(	CC(C)(CO)(	PKVVTUW	4-hydroxy	146.058	146.058	0	1	0	0	
49	3-Methyl-2	3-Methyl-2	116.11	C5H8O3	54.4	115	0.7	8	1	3	2	InChI=1S/(	CC(C)C(=O	CC(C)C(=O	QHKABHO	3-methyl-2	116.047	116.047	0	1	0	0	
51	2-Oxoglut	2-ketoglut	146.1	C5H6O5	91.7	171	-0.9	10	2	5	4	InChI=1S/(	C(CC(=O)C	C(CC(=O)C	KPGXRSRH	2-oxopent	146.022	146.022	0	1	0	0	
66	2,5-Dichlo	2,5-dichlor	163	C6H4Cl2O	20.2	97.1	3.1	9	1	1	0	InChI=1S/(	C1=CC(=C(	C1=CC(=C(	RANCECPK	2,5-dichlor	161.964	161.964	0	1	0	0	
70	4-Methyl-2	4-methyl-2	130.139	C6H10O3	54.4	126	0.9	9	1	3	3	InChI=1S/(	CC(C)CC(=	CC(C)CC(=	BKAINAXT	4-methyl-2	130.063	130.063	0	1	0	0	
71	2-Oxoadip	2-Oxoadip	160.12	C6H8O5	91.7	183	-0.5	11	2	5	5	InChI=1S/(	C(CC(=O)C	C(CC(=O)C	FGSBNEBH	2-oxohexa	160.037	160.037	0	1	0	0	

Figure 37: PubCHEM Dataset

In our project, data is preprocessed from three key sources: GDSC, CCLE, and PubChem. Detailed information about these data sources can be found in Appendix Sections 9.3, 9.1, and 9.5, respectively. The resulting preprocessed dataset is provided in Appendix Section 9.6.

Section 1.7: Software Limitations

1. Absence of Feature Extraction in CCLE Gene Expressions

The software faces a limitation due to the absence of feature extraction on CCLE gene expressions. This is because despite employing various methods such as variance thresholding, information gain, randomised search CV, and PCA for training data, surprisingly, the baseline model without feature extraction on gene expression data exhibited superior performance, as evidenced by the root mean square error (RMSE) in **Table 1**. This discrepancy may arise from the preprocessed nature of the gene expression data obtained from the CCLE website, indicating that the included gene expressions adequately represent cancer cells. However, implementing feature extraction could potentially complicate the model's ability to identify corresponding cancer cells if some features are extracted. Consequently, the absence of feature extraction results in the inclusion of all 19,221 gene expression columns as training features, leading to a very high dimensionality of the training dataset. This limitation impacts the software's ability to refine the training data and may lead to issues related to high dimensionality, model complexity, and interpretability. To mitigate this limitation, future iterations of the software could incorporate further feature extraction techniques tailored specifically to gene expression data. Additionally, conducting thorough analyses to determine the necessity and effectiveness of feature extraction methods on CCLE gene expressions could provide valuable insights into enhancing model performance while managing dimensionality.

Ranking	Features Extraction Method	RMSE
1	**Baseline Approach - Without Feature Extraction**	**0.952**
2	Variance Thresholding	1.248
3	Information Gain	1.962
4	Randomized Search CV	1.984
5	PCA	2.019

Table 1: Table of Summary of Results Upon Employing Different Features Extraction Method

2. Exclusive Use of Combined GDSC-CCLE Dataset for Model Training

The software is limited by its exclusive training using the combined GDSC and CCLE dataset, neglecting the potential to train three distinct models using different datasets: pure GDSC, pure CCLE, and the combined GDSC-CCLE dataset. While the GDSC dataset focuses on drug response and the CCLE dataset concentrates on gene expression, combining these datasets offers a more comprehensive dataset, maximising the value of each individual dataset. However, the decision to solely train the model with the combined GDSC-CCLE dataset was made during the software development process. This decision overlooked the opportunity to develop two additional models trained with the pure CCLE dataset and pure GDSC dataset, which could have provided users with more options for prediction models. Moreover, training models with different data sources would enable the validation of each model's performance. This limitation arises from the neglect of other combinations of data sources for model training, potentially limiting the software's versatility and applicability to different datasets and scenarios. Future iterations of the software could address this limitation by exploring the development of multiple models trained with different datasets to offer users a broader range of predictive capabilities and enhance the software's adaptability to diverse data sources.

3. Lack of Precision in Predicted LN_IC50 Values

While our model's predicted results demonstrate validity, as indicated by RMSE values typically below 1, they lack the precision required for precise LN_IC50 predictions. Despite falling within the range of 0.8 to 1, the RMSE metric does not guarantee the accuracy necessary for precise LN_IC50 predictions, highlighting a limitation in the model's performance. This lack of precision may impact the model's utility in scenarios where accurate LN_IC50 predictions are essential for informed decision-making or analysis, potentially leading to suboptimal outcomes. Addressing this limitation requires further refinement of the model through parameter tuning, algorithm exploration, or incorporation of additional features or data sources to enhance prediction accuracy and precision. Rigorous validation and testing procedures, including cross-validation techniques and comparison with benchmark datasets, are essential for assessing the effectiveness of these improvements and guiding future iterations of the model, to further reduce the RMSE.

4. Constrained User Input Format and Dimension

Our software imposes limitations on user input formats, restricting uploads to CSV files only. Additionally, the dimensions of user input data are significantly inflated due to the inclusion of all CCLE gene expressions without

extraction, aligning with the dimensions of the model's training data. This constraint may cause inconvenience to users, particularly those seeking to upload data in alternative formats like xlsx or txt. Furthermore, the current handling of non-numeric data within user uploads presents a limitation; instances of non-numeric data lead to direct rejection of the entire dataset for prediction. Given the substantial dimensionality of the input dataset, users may overlook the presence of non-numeric entries. To mitigate this limitation, enhancements such as modifying the model to drop rows containing non-numeric data could be explored, which is mentioned in Section 8.2. Also, additional efforts are required from users to prepare datasets for prediction, reflecting the inherent challenges posed by the large dimensionality of input data. These limitations underscore the importance of refining user input processes and considering alternative data handling approaches to enhance usability and mitigate user inconvenience.

5. Lack of Specific Error Messaging for Non-Numeric and Dimensional Errors

Our software presents a limitation in its error messaging system, where non-numeric and incorrect dimensional issues are both communicated under the same error message, as observed in the blackbox testing section 4.1. This approach fails to provide users with specific details regarding the nature of the error, potentially leading to confusion. Users may struggle to discern whether the error stems from incorrect dimensions or the presence of non-numeric values in the dataset. Additionally, the absence of specific error messages further complicates user troubleshooting. For instance, when encountering dimensional errors, users are not informed about the missing or extra dimensions, while errors related to non-numeric values do not specify which columns or rows contain these values. Moreover, the handling of NA values lacks specificity, as users are not informed about the rows where NA values are dropped. Given the large dimensionality of user input data, the lack of specific error messaging impedes users' ability to identify the root cause of errors. This limitation highlights the need for further improvements in error handling mechanisms to enhance user experience and facilitate troubleshooting. Although currently categorised as a minor issue since error cases are still addressed, addressing this limitation would contribute to a more user-friendly and intuitive software interface. Future iterations of the software are expected to address this limitation by implementing enhanced error messaging features for improved user guidance and error resolution.

6. Absence of Gene Expression Visualization Hinders Relationship Identification

Our software faces a limitation in visualising gene expression data, comprising over 19,000 genes per cell, which impedes the identification of relationships between gene expression and LN_IC50. The inability to display gene expression data stems from potential loading issues, inefficiency, and lag associated with visualising such vast datasets within the software interface. As a result, users are deprived of crucial insights into the relationship between gene expression patterns and drug response represented by LN_IC50 values. Without visualising gene expression data, users are unable to discern patterns or correlations that may exist, limiting their ability to derive meaningful insights and make informed decisions regarding drug response predictions. Addressing this limitation would involve optimising the software's visualisation capabilities to accommodate large gene expression datasets efficiently, enabling users to explore and interpret the relationship between gene expression profiles and drug sensitivity more effectively. Enhancing gene expression visualisation features would greatly enhance the software's utility and facilitate comprehensive analysis for users. However, the most promising approach is to detect the top influencing gene expressions and focus on the visualisations among these gene expressions.

Section 2: Technical Guide

Section 2.1: Software Configuration Guide

Section 2.1.1: Prerequisites - Tool and Platform Set-Up

Prerequisites:

1. Make sure to download **Visual Studio Code** as the primary platform for software installation and launching. If you do not have it installed yet, you can download from this **link** here.
2. Make sure you have **Python version 3.9.7** or later installed. You can verify your Python version by entering "**python --version**" in your command prompt or terminal in Visual Studio Code. If you do not have Python in your Visual Studio Code, follow the instructions in this **link** to set up the Python extension.

Section 2.1.2: DRUGRES Installation

Step 1: Download the folder **"Predictive Model Web Application"** from the GitHub repository using this **link**, save it as a zip file, and extract it to your local repository. This is the source code of our software application.

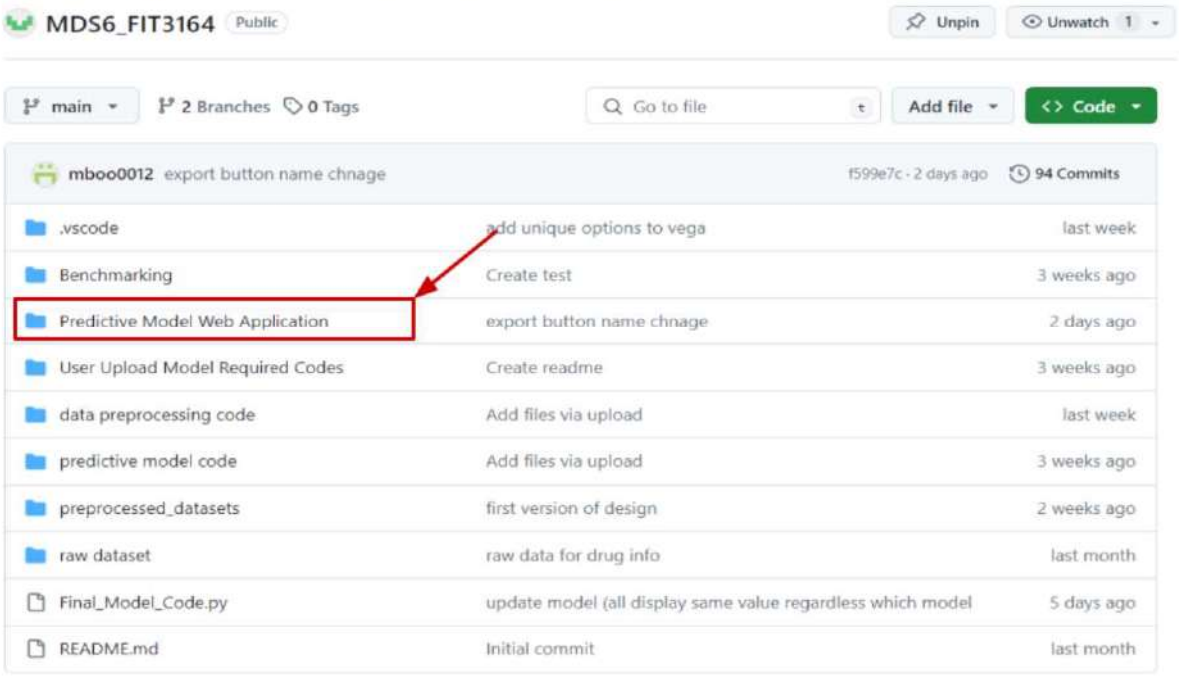


Figure 38: shows the Folder "Predictive Model Web Application" from the GitHub repository

Step 2: Open the "Predictive Model Web Application" folder in **Visual Studio Code**. Change the directory to "**src**" by running the command **cd .\src**. Ensure the **directory** is correct, as shown below.

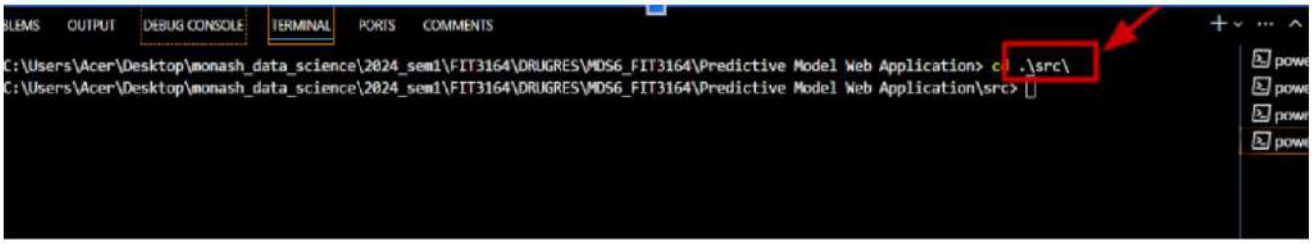


Figure 39: shows the directory "cd.\src\." to follow

Step 3: Execute the following commands to **install** the necessary **packages** and **libraries**.

pip install flask
pip install flask_session
pip install rdkit
pip install tensorflow
pip install scikit-learn

pip install pandas

** Note: If you encounter pip version issues, run the following command to ensure the latest version of pip is installed.

pip install –upgrade pip

** Note: If you encounter tensorflow "ModuleNotFoundError", try running the following command

pip install tensorflow –user

Step 4: Run the command below to **initiate** the software application **launching**.

python app.py

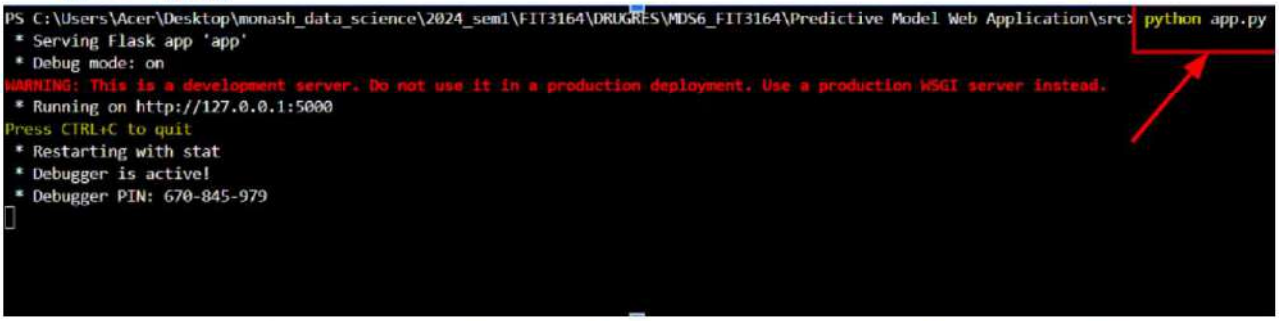


Figure 40: shows how to run the application by executing the command "python app.py"

Step 5: After running the command, a link will be displayed, indicating "**Running on http://127.0.0.1:5000**".
Click on the link shown in the terminal to **launch** the software, "DRUGRES" in your web browser. The interface shown below is what you should expect to see once the software is launched.

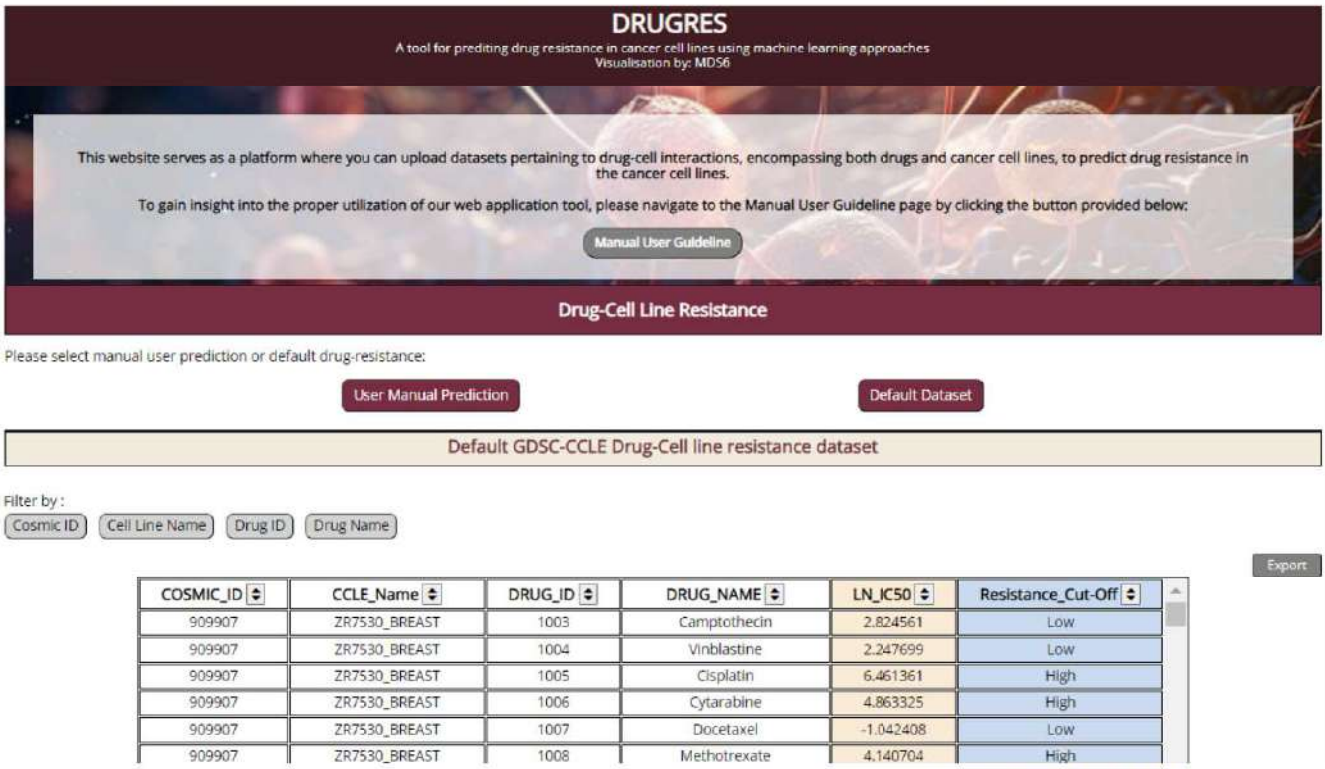


Figure 41: shows the deployment of website interface upon clicking on link http://127.0.0.1:5000

Section 2.2: User Upload Data Preprocessing Guide

Section 2.2.1: Recommended Tool/ Software

It is advisable to preprocess the user-uploaded dataset using the "**Anaconda**" tool, especially considering the dataset's large dimensions resulting from a high number of columns. Using conventional tools like "Excel" or "Google Sheets" might result in loss of information or inefficient performance. If you have not installed "Anaconda" yet, you can do so

by following this link. The recommended software for data preprocessing is "**Jupyter Notebook**", which can be launched via "Anaconda". Another alternative option is to use "**GoogleCollab**".

Section 2.2.2: User Upload Data Preprocessing Guides

Case 1: Prediction on Currently Exists Cancer Cells

Step 1: Download and **Read** the **CCLE Gene Expression Dataset:**

Download the "CCLE_expression.csv" from this link.

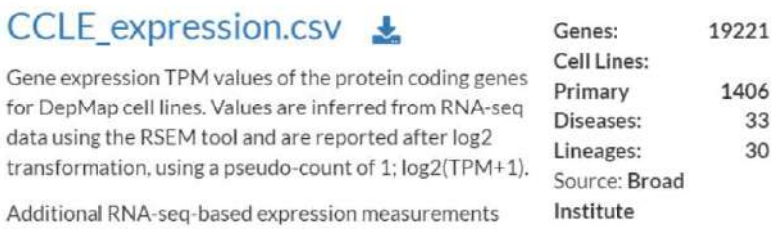


Figure 42: shows downloaded CCLE_expression.csv

In your Jupyter notebook, create a new **Python 3** notebook. Place the downloaded "CCLE_expression.csv" under the same local repository of the Python 3 notebook. Run the following commands:

```
import pandas as pd
import numpy as np
ccle_gene_exp_raw = pd.read_csv("CCLE_expression.csv")
```

	Unnamed: 0	TSPAN6 (7105)	TNMD (64102)	DPM1 (8813)	SCYL3 (57147)	C1orf112 (55732)	FGR (2268)	CFH (3075)	FUCA2 (2519)	GCLC (2729)	...	H3C2 (8358)	H3 (83
0	ACH-001113	4.331992	0.000000	7.364397	2.792855	4.470537	0.028569	1.226509	3.042644	6.499686	...	2.689299	0.189
1	ACH-001289	4.566815	0.584963	7.106537	2.543496	3.504620	0.000000	0.189034	3.813525	4.221104	...	1.286881	1.049
2	ACH-001339	3.150560	0.000000	7.379032	2.333424	4.227279	0.056584	1.310340	6.687061	3.682573	...	0.594549	1.097
3	ACH-001538	5.085340	0.000000	7.154109	2.545968	3.084064	0.000000	5.868143	6.165309	4.489928	...	0.214125	0.632
4	ACH-000242	6.729145	0.000000	6.537607	2.456806	3.867896	0.799087	7.208381	5.569656	7.127014	...	1.117695	2.358
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1401	ACH-000285	0.056584	0.000000	6.604071	3.264536	4.972693	0.411426	0.097611	0.704872	4.829850	...	2.229588	0.084
1402	ACH-002669	3.109361	0.000000	7.031219	1.541019	3.664483	0.014355	3.624101	6.805292	4.472488	...	0.189034	0.400
1403	ACH-001858	4.390943	0.000000	7.013127	1.887525	3.252476	0.028569	3.286881	6.902074	5.410748	...	1.097611	0.400
1404	ACH-001997	5.057017	0.000000	7.814935	2.538538	3.893362	0.028569	4.078951	6.971429	4.469886	...	0.831877	0.847
1405	ACH-000052	4.247928	0.000000	6.174127	2.316146	3.823749	0.189034	1.321928	3.536053	3.943921	...	1.244887	1.201

1406 rows × 19222 columns

Figure 43: The Output of CCLE_expression.csv

Step 2: Download and **Read** the **Sample Info Dataset:**

Download the "sample_info.csv" from this link.

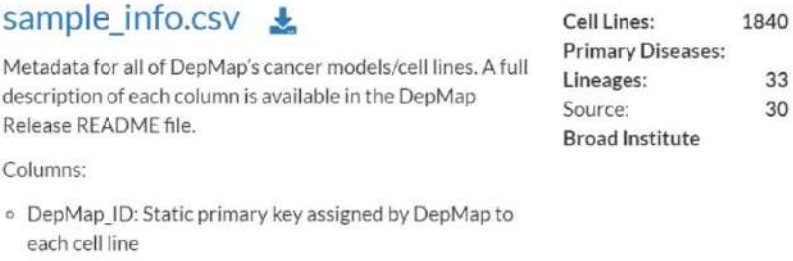


Figure 44: shows downloaded sample_info.csv

Place the downloaded "sample_info.csv" under the same local repository of the Python 3 notebook. Run the following commands:

```
sample_info_raw = pd.read_csv("sample_info.csv")
```

	DepMap_ID	cell_line_name	stripped_cell_line_name	CCLE_Name	alias	COSMI
0	ACH-000016	SLR 21	SLR21	SLR21_KIDNEY	NaN	
1	ACH-000032	MHH-CALL-3	MHHCALL3	MHHCALL3_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	NaN	
2	ACH-000033	NCI-H1819	NCIH1819	NCIH1819_LUNG	NaN	
3	ACH-000043	Hs 895.T	HS895T	HS895T_FIBROBLAST	NaN	
4	ACH-000049	HEK TE	HEKTE	HEKTE_KIDNEY	NaN	
...	...	...	...	...	...	
1835	ACH-002393	CRO-AP3	CROAP3	CROAP3_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	NaN	
1836	ACH-002394	GEO	GEO	GEO_LARGE_INTESTINE	NaN	
1837	ACH-002395	HuH-6 Clone 5	HUH6CLONE5	HUH6CLONE5_LIVER	NaN	
1838	ACH-002396	Sarc9371	SARC9371	SARC9371_BONE	NaN	
1839	ACH-002397	KMH-2	KMHDASH2	KMH2_THYROID	NaN	205409

1840 rows × 29 columns

Figure 45: shows the read sample_info.csv as dataframe

Step 3: Run the following commands to **merge** the "**ccle_gene_exp_raw**" with "**sample_info_raw**" to retrieve the COSMIC_ID for the cell lines in the "ccle_gene_exp_raw"

```
sample_info= sample_info_raw[['COSMICID','DepMap_ID', 'CCLE_Name', 'primary_disease']]

# Rename the 'Unnamed: 0' column to 'DepMap_ID' in ccle_gene_exp_raw DataFrame
ccle_gene_exp_raw = ccle_gene_exp_raw.rename(columns = {'Unnamed: 0':'DepMap_ID'}, inplace =
False)

# Merge ccle_gene_exp_raw DataFrame with sample_info DataFrame on 'DepMap_ID' column
ccle_gene_exp = pd.merge(ccle_gene_exp_raw, sample_info, on=['DepMap_ID'], how='left')
to_drop = ['DepMap_ID']

# Drop the 'DepMap_ID' column from ccle_gene_exp DataFrame
ccle_gene_exp.drop(to_drop, inplace=True, axis=1)

# Extract 'CCLE_Name' column from ccle_gene_exp DataFrame and create a new DataFrame
cell_line_name = ccle_gene_exp[['CCLE_Name']]

# Rename the 'CCLE_Name' column to 'Cell_Line_Name' in cell_line_name DataFrame
cell_line_name = cell_line_name.rename(columns = {'CCLE_Name':'Cell_Line_Name'}, inplace = False)
# Convert 'Cell_Line_Name' column values to list
cell_line_name_lst = cell_line_name['Cell_Line_Name'].values.tolist()

# Insert 'Cell_Line_Name' column at the beginning of ccle_gene_exp DataFrame
ccle_gene_exp.insert(loc = 0,
                     column = 'Cell_Line_Name',
                     value = cell_line_name_lst)
to_drop = ['CCLE_Name']

# Drop the 'CCLE_Name' column from ccle_gene_exp DataFrame
ccle_gene_exp.drop(to_drop, inplace=True, axis=1)

# Extract 'COSMICID' column from ccle_gene_exp DataFrame and create a new DataFrame
cosmic_id = ccle_gene_exp[['COSMICID']]

# Rename the 'COSMICID' column to 'COSMIC_ID' in cosmic_id DataFrame
cosmic_id = cosmic_id.rename(columns = {'COSMICID':'COSMIC_ID'}, inplace = False)

# Extract 'COSMIC_ID' column values to list
cosmic_id_lst = cosmic_id['COSMIC_ID'].values.tolist()

# Insert 'COSMIC_ID' column at the beginning of ccle_gene_exp DataFrame
ccle_gene_exp.insert(loc = 0,
```

```
                column = 'COSMIC_ID',
                value = cosmic_id_lst)
to_drop = ['COSMICID']
# Drop the 'COSMICID' column from ccle_gene_exp DataFrame
ccle_gene_exp.drop(to_drop, inplace=True, axis=1)
# Drop rows with missing values (NaN) from ccle_gene_exp DataFrame
ccle_gene_exp = ccle_gene_exp.dropna(axis='index')
# Convert 'COSMIC_ID' column values to int64 data type
ccle_gene_exp['COSMIC_ID'] = ccle_gene_exp['COSMIC_ID'].apply(np.int64)
```

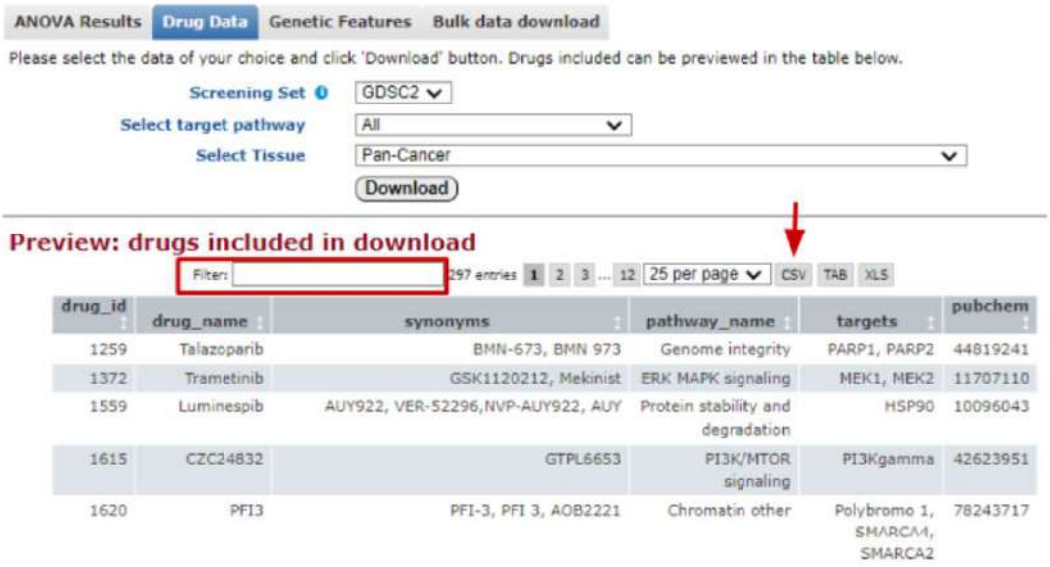
Step 4: You can now **select** the **cancer type of interest** via the command below. It is highly suggested to extract the data of "Breast Cancer" type since our model is trained primarily on Breast Cancer Cells.

```
ccle_cancer_data = ccle_gene_exp[ccle_gene_exp['Cancer_Type']== 'Cancer Type of Interest']
```

Step 5: Select the **Cancer Cells of Interests** for the prediction to be made on by running the command below.

```
ccle_cancer_data = ccle_cancer_data [ccle_cancer_data ['COSMIC_ID'].isin([C1, C2, C3, …])]
```

Step 6: From the GDSC website, browse and **filter the drugs of interest, output as csv file.**



drug_id	drug_name	synonyms	pathway_name	targets	pubchem
1259	Talazoparib	BMN-673, BMN 973	Genome integrity	PARP1, PARP2	44819241
1372	Trametinib	GSK1120212, Mekinist	ERK MAPK signaling	MEK1, MEK2	11707110
1559	Luminespib	AUY922, VER-52296,NVP-AUY922, AUY	Protein stability and degradation	HSP90	10096043
1615	CZC24832	GTPL6653	PI3K/MTOR signaling	PI3Kgamma	42623951
1620	PFI3	PFI-3, PFI 3, AOB2221	Chromatin other	Polybromo 1, SMARCA4, SMARCA2	78243717

Figure 46: shows the filter tab from GDSC website

Step 7: **Extract** the "**pubchem**" from the downloaded csv file and output it as a csv file. Assume the downloaded csv filename is "export.csv", run the commands below for retrieval of pubchem ids of drugs of interest

```
pubchemid = pd.read_csv("export.csv")
drug = pubchemid['pubchem']
drug.to_csv('my_drug_list.csv', index=False, header=False)
```

Step 8: The **"isosmiles"** column in the sample dataset can be obtained from the PubChem website following these steps:

(1) Click on the "**UPLOAD ID LIST**" button.

Figure 47: shows upload in Explore Chemistry

(2) Select the Identifier Type "**Compound, e.g. CID like 2244**". **Enter** a list of drug pubchem ids manually in the text field, separating each **drug pubchem ID by a comma** (e.g., 1013, 1014, 1015).

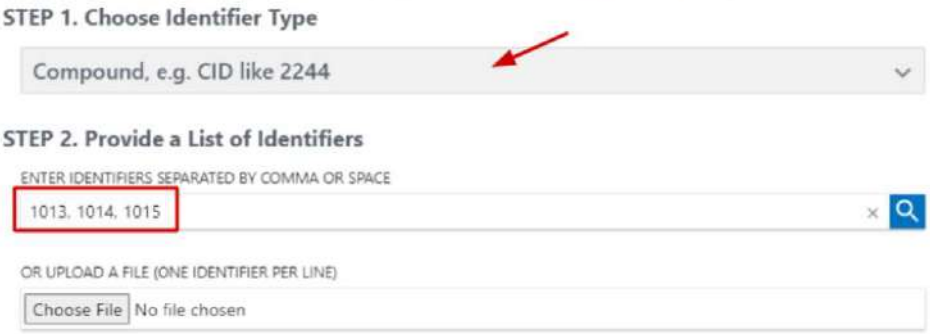


Figure 48: shows operation for Identifier Type

(3) Alternatively, upload a **CSV file** containing a drug list (pubchem ids), with one drug per line. (The file **from Step 7**)

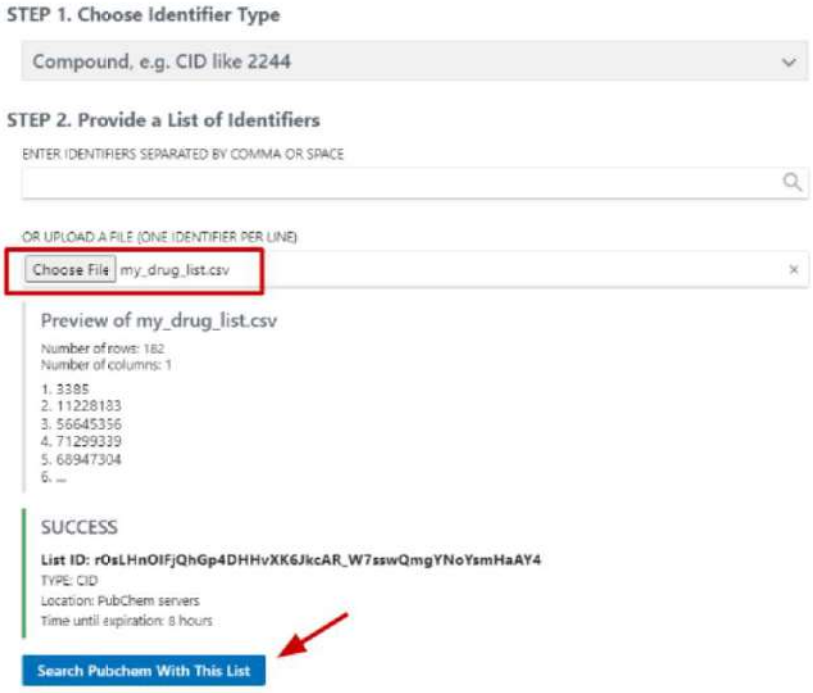


Figure 49: shows the upload csv file

(4) After entering or uploading the drug list, **download** the "isosmiles" in **CSV** format to retrieve the "isosmiles" of the drugs to be predicted.

Figure 50:shows how to download the isosmiles

Step 9: Assigns drugs from the previously exported GDSC drugs of interest file "exported.csv" **to different cell lines** based on user preferences. For instance, if a user is interested in evaluating the effectiveness of drug A, drug B, drug C, drug D, and drug E on COSMIC_ID A, they need to assign these five drugs to COSMIC_ID A in the "**ccle_gene_data**" dataframe. This means that instead of one row for COSMIC_ID A, there will be five rows, each paired with one of the five drugs respectively. The drug information required for **merging with the "ccle_gene_data" dataframe** includes "**DRUG_ID**", "**DRUG_NAME**", and "**PubCHEM**". No sample commands are provided for this step as each user may have different preferences for the drug-cell pairs of interest, requiring specific preprocessing work. Assume the output dataframe of this step has the name "**gene_drug_df**".

Step 10: Read the **drug isosmiles file downloaded** in **Step 8.**

Place the downloaded "PubChem_compound.csv" under the same local repository of the Python 3 notebook. Run the following commands:

```
compound_data_raw = pd.read_csv("PubChem_compound.csv")

# Extract relevant columns from compound_data_raw DataFrame
compound_data = compound_data_raw[['cid', 'isosmiles']]

# Rename column ' cid' to 'PubCHEM' in compound_data DataFrame
compound_data = compound_data.rename(columns = {'cid':'PubCHEM'}, inplace = False)

# Convert 'PubCHEM' column values to string type
compound_data['PubCHEM']=compound_data['PubCHEM'].astype(str)
```

	PubCHEM	isosmiles
0	3385	C1=C(C(=O)NC(=O)N1)F
1	11228183	CN(C)CC[C@H](CSC1=CC=CC=C1)NC2=C(C=C(C=C2)S(=O...
2	56645356	CC1=CC=CC=C1C(C(=O)NC2CCCCC2)N(C3=CC(=CC=C3)F)...
3	71299339	C1CC1NS(=O)(=O)C2=CC(=C(C=C2)C3=CSC=C3)NC(=O)N...
4	68947304	C[C@@H](C1=C(N=C2C=C(C=CC2=C1)F)C3=CC=CC=N3)NC...
...	...	...

Figure 51: shows the read isosmiles file downloaded

Step 11: Merge the "gene_drug_df" with the **"compound_data"** to pair the unique isosmiles for each drug based on "**PubCHEM**". Run the following commands. This will produce the **final user upload data**. Now, the user can proceed to upload the csv file to "DRUGRES' for manual prediction.

```
gene_drug_isosmiles = pd.merge(gene_drug_df, compound_data, on=['PubCHEM'], how='left')
to_drop = ['PubCHEM']
gene_drug_isosmiles .drop(to_drop, inplace=True, axis=1)
gene_drug_isosmiles = gene_drug_response_isosmiles.reset_index(drop= True)
gene_drug_isosmiles.to_csv("User_Sample_Upload_Data.csv")
```

Step 12: Additional Preprocessing/ Checking on the output "User_Sample_Upload_Data.csv"

Please note that the examples provided above serve as a demonstration and guide on how to prepare the user upload dataset. The specific commands may vary depending on individual user requirements and preferences. The crucial aspect is to ensure that the data format aligns with the specifications outlined in Section 1.3: User Upload Data Format Requirements.

Case 2: Prediction on New Cancer Cells (Not available in Cancer Cell Line Encyclopedia)

A significant consideration is the emergence of new cancer cell types over time, such as novel variants of breast cancer cell lines not documented in the Cancer Cell Line Encyclopedia (CCLE). Consequently, researchers often seek to determine the most effective drugs for treating these new cancer cells. If a user aims to assess the efficacy of various drugs on these New Cancer Cells, Steps 1 to 5 in Case 1 would be replaced with acquiring the new gene expressions specific to the new cancer cell. It is important to note that data for all **19221** types of **gene expressions** are necessary for this new cell line, necessitating extensive research and experimentation. This scenario is primarily relevant in real-world medical contexts. The remaining steps of the process remain unchanged.

Case 3: Prediction on New Drugs (Not available in PubChem)

Similar to case 2, for this case, Steps 6 to 8 of Case 1 will be omitted. Instead, experiments will be conducted to obtain the chemical compound structure, "**isosmiles**", of the new drug. This information will then be merged with the CCLE gene expression dataframe.

Section 3: Software Test/ QA Introduction

Section 3.1: Summary of Software Development Process and Introduction to Test Planning

Testing constitutes a pivotal phase in software development, ensuring that applications perform accurately, meet user needs, and deliver reliable results. In the context of our drug resistance prediction web application, DRUGRES, comprehensive testing is indispensable to validate functionality, prediction accuracy, and user experience. This testing plan delineates the methodologies and procedures to ascertain the dependability and efficacy of our software.

To effectively strategize our testing plan, it is crucial to grasp the software development life-cycle of our project. Our development process encompasses several stages, including data preprocessing, predictive model development, testing, and web development, as illustrated in **Figure 52**.

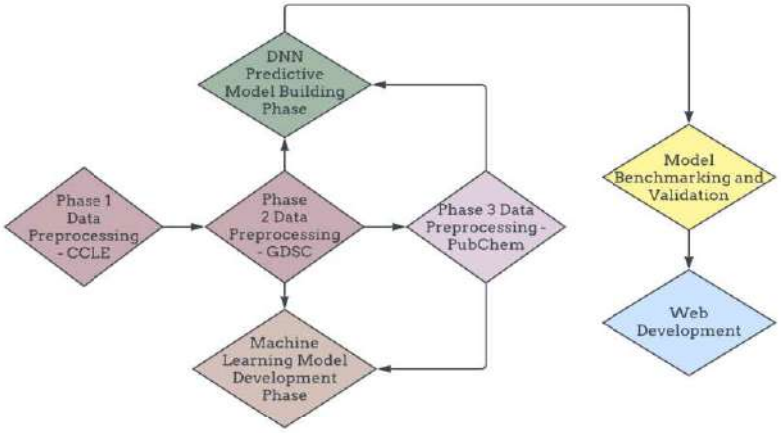


Figure 52: Overall Software Design Methodology

Our project commences with the aggregation of gene expression datasets for cancer cells, drug resistance (LN_IC50) from drug response datasets, and isomeric SMILES compound structures of drugs from PubCHEM, aimed at producing a training dataset for drug resistance prediction with the design shown in **Figure 53** below.

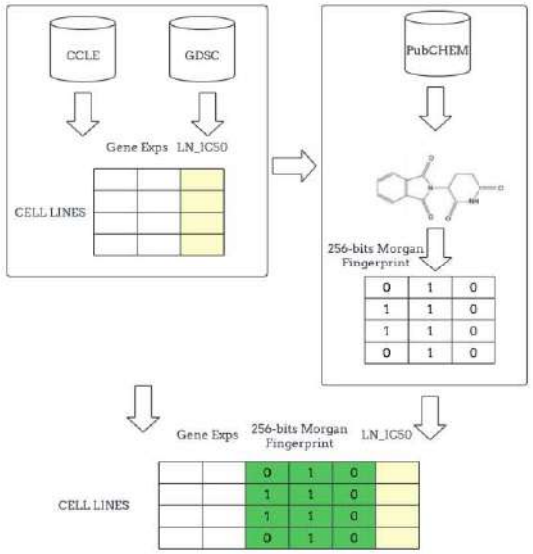


Figure 53: Training Dataset Design

Consequently, our data preprocessing unfolds in three primary phases. Phase 1 entails cleaning and feature selection from CCLE gene expression data, with a focus on breast cancer cells. Phase 2 involves extracting drug-cell pairs and their LN_IC50 values, which are then merged with the Phase 1 dataset to forge a gene-drug dataset. Finally, in Phase 3, drug IDs are extracted from the gene-drug dataset, their isomeric SMILES compound structures are retrieved from PubChem via PubChem ID, and merged with the previously preprocessed dataset, and subjected to Morgan fingerprints

to convert drug structures into 256-bit binary representations. Throughout the process, the data preprocessing actions we have taken encompass feature extraction, data cleansing, removal of duplicates and NA values, and chemical structure conversion. The data preprocessing processes are shown in **Figure 54** below.

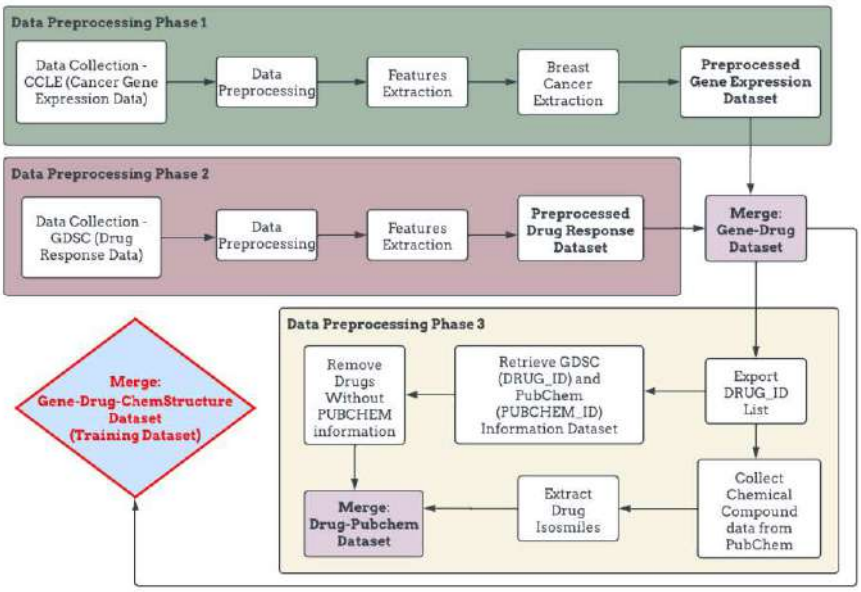


Figure 54: Design Methodology for Data Preprocessing

We have explored various machine learning approaches and settled on the Deep Neural Network approach due to its better performance in the initial phases. Leveraging the preprocessed "Gene-Drug-ChemStructure" Dataset, we partition it into training, validation, and testing datasets. For our predictive model design, we adopt a sequential Deep Neural Network approach. Features are normalised, and hyperparameter tuning is conducted to ascertain the optimal parameter combination for model training. Once trained and validated, the model predicts LN_IC50 values on the test set. Model performance evaluation on the test set involves metric assessment and benchmarking. Iterative model refinement and tuning culminate in the finalised DNN model depicted in **Figure 55**.

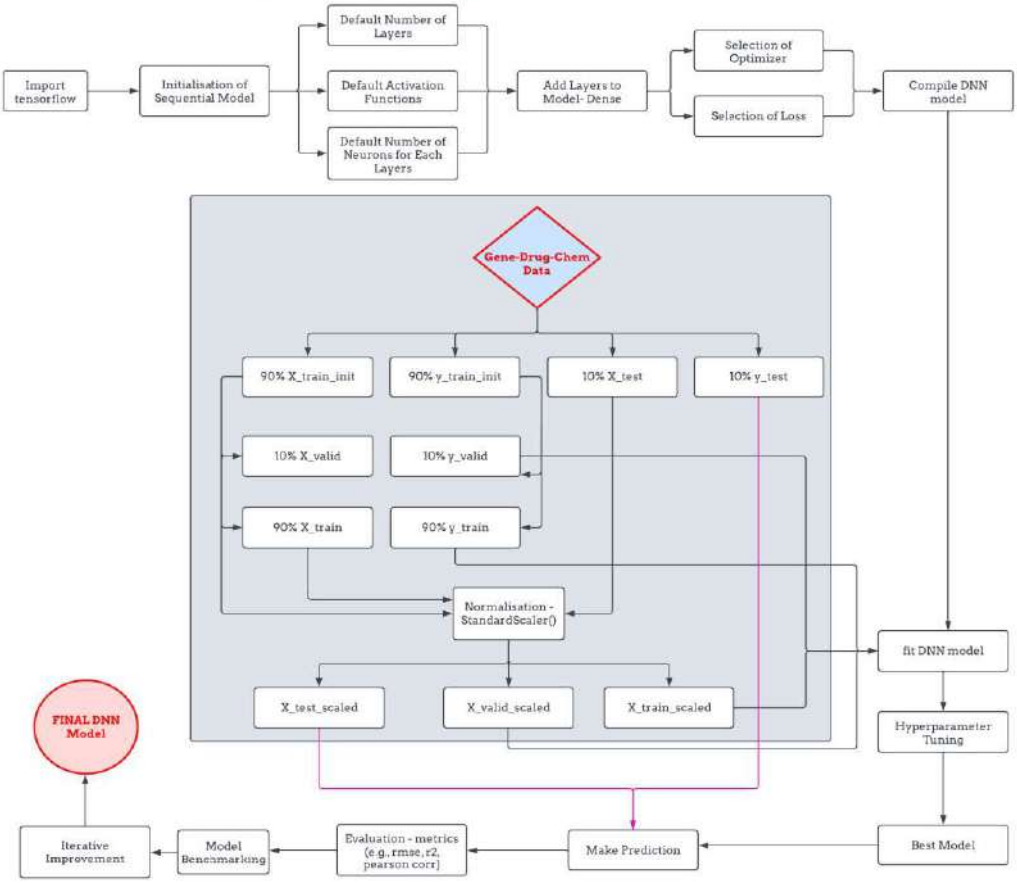


Figure 55: Design Methodology for Predictive Model Development

In our web design concept, we integrate the trained DNN model into a user-friendly web application for drug resistance prediction. Users upload datasets as X_test inputs, and the model forecasts LN_IC50 values accordingly. The model classifies drugs based on high or low resistance to specific cells using a predefined resistance threshold. Prediction outcomes are presented via tabular format and visualisations within the web application. Supplementary usability features and buttons are incorporated to ensure seamless user interaction. As the web application nears completion, it transitions to the software **testing phase**, where we will now delve into our testing strategies, methodologies, and practical execution to validate the application's functionality. The development flow of our web application is outlined in **Figure 56** below.

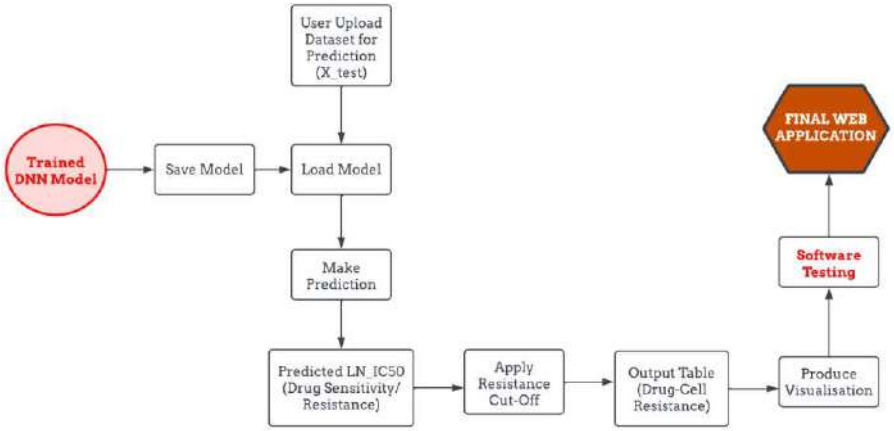


Figure 56: Design Methodology for Web Application

Section 3.2: Software Testing Plan

Based on the software development design and processes outlined in Section 3.1, our software application serves three primary domain functionalities: prediction generation based on input gene-drug information, validation of prediction performance and results, and visualisation of drug resistance relationships with different cancer cells. To support these functionalities, various interactive features must be integrated to handle tasks such as receiving input files, executing predictive models, passing gene-drug data as input parameters, displaying prediction results, generating visualisations, and providing additional features for result exporting and analysis. Furthermore, ensuring user-friendly interaction with the web application is also a key goal, as usability plays a critical role in facilitating seamless usage by diverse users. Therefore, our test planning encompasses three main components: **blackbox testing**, **integration testing**, and **usability testing**. Each component is tailored to address specific aspects of our software's functionality and performance, ensuring comprehensive evaluation and validation. Below is an overview of our testing plan

For **black box testing**, our focus lies on the input data for predictions, specifically the user-upload dataset comprising gene expressions, drug information, and chemical compound structures. It is imperative that this dataset matches the dimensions of our training dataset to ensure compatibility with our predictive model. Our testing plan encompasses ***user data format testing*** to assess the software's ability to handle various data formats. This includes scenarios such as datasets with incorrect dimensions or missing values, where we aim to verify the software's capability to handle such issues effectively. ***Upload testing*** is another crucial aspect, ensuring the reliability and functionality of the data upload feature. This involves simulated delay testing to assess the responsiveness of the upload process and prevent accidental predictions on previous datasets. We also conduct upload logical testing to validate the software's behaviour when handling different datasets. For instance, we test whether uploading files with the same filename but different content yields different results and whether the latest uploaded dataset is prioritised for predictions. Testing the successful reading of uploaded files into the model and their passage as X-test values is essential. We need to guarantee that the user-upload dataset is properly integrated into the model before predictions are made for the prediction action to be done. Another critical aspect of black box testing involves ***prediction testing*** and ***prediction outputting testing***. This entails testing the user manual prediction functionality to ensure that the "make prediction" action can be executed seamlessly. We need to examine how this action is initiated and whether it effectively loads the predictive model to perform the prediction process. This includes verifying that the predictive model is correctly accessed and utilised during the prediction phase. In addition, we must conduct thorough testing of the prediction output data which examines how the prediction action outputs the results and how these results are stored within the software. We need to confirm that the

output is valid and directly related to the input dataset, ensuring that irrelevant or erroneous data is not generated. It is crucial to verify that the number of rows in the output matches the number of rows in the input dataset and that the drug-cell pairs in the output align with those in the input dataset. Furthermore, a key focus of black box testing is ***validating the accuracy and reliability of the prediction results*** produced by the software. We need to ensure that the output results are valid and acceptable, reflecting the true predictive capabilities of the model. Therefore, comprehensive testing of the predictive model's performance is essential. This involves evaluating its ability to accurately predict drug resistance in cancer cell lines based on the provided input data, ensuring that the predictions align with expected outcomes.

For **integration testing**, we have outlined a comprehensive plan to ensure the seamless integration of various components within our drug resistance prediction web application. Firstly, we will focus on the ***integration of the predictive model into the web application*** architecture. This testing phase aims to verify that the predictive model can be effectively called and utilised within the web application environment, ensuring that it functions correctly and produces accurate predictions. Moving on, our testing will focus on the ***integration of user upload data preprocessing steps***. This involves validating the preprocessing steps applied to user-uploaded data before it is passed as input to the predictive model. We will specifically test the handling of missing values (NA), removal of rows, standard scaling, and the application of Morgan fingerprints to ensure that the input data is properly prepared for model input. Next, we will focus on the ***integration of prediction output into the web application***. This testing phase will examine how the output generated by the predictive model is displayed within the web interface. We will verify that the displayed output aligns with the prediction results produced by the model, ensuring consistency and accuracy in the presentation of prediction data to the users. Lastly, our testing will concentrate on the ***integration of visualisation output display into the web application***. This involves verifying the integration of visualisation outputs, such as bar charts and tooltips, into the web interface. We will ensure that these visual elements are displayed correctly and provide users with meaningful insights into the prediction results. Overall, our integration testing plan aims to validate the seamless integration of the predictive model, data preprocessing steps, prediction output, and visualisation outputs within our web application. By rigorously testing these integration points, we aim to ensure the reliability, accuracy, and usability of our software application.

Lastly, for **usability testing**, we have developed a comprehensive plan to evaluate the user interface (UI) and user experience (UX) of our drug resistance prediction web application. First of all, we will focus on ***testing the UI and UX aspects***, which involves User Interface Navigation Testing. This testing phase aims to evaluate the ease of navigation within the web application, ensuring that users can intuitively navigate through different sections and features. Additionally, we will conduct Upload and Export Dataset Testing to assess the usability of the dataset upload and export functionalities. We will evaluate the ease of uploading datasets, as well as the clarity of instructions provided to users. Furthermore, we will perform Table Sorting and Filtering Testing to assess the usability of sorting and filtering features within the application. This testing phase aims to ensure that users can easily sort and filter dataset tables to find relevant information efficiently. Moving on, we will conduct ***Third Party Testing*** to gather feedback from external users. In this part, we will set up third-party testing by inviting peers from other teams, personal friends, and family members to try installing our software application. We will gather feedback on the software installation and configuration process, as well as identify any potential usability issues. Finally, we will review and respond to the feedback received from third-party testers. This feedback will be analysed to identify areas for improvement and implement necessary enhancements to enhance the usability of our web application. Overall, our usability testing plan aims to ensure that our software application provides an intuitive and user-friendly experience for all users, ultimately enhancing user satisfaction and engagement.

Section 3.3: Description of Test Approach Used

Figure 57 illustrates the breakdown and structure of the **Blackbox Testing** process. Blackbox testing for our software involves evaluating the system's functionalities and focuses on the input and output of the software, ensuring that the application behaves as expected under various scenarios. We have chosen **manual testing** as our primary approach for blackbox testing. Below, we describe the test approaches for different aspects of blackbox testing and explain why manual testing is preferred for our needs.

Data format testing is essential for ensuring that the application handles various input scenarios correctly. We manually upload datasets with different characteristics to the web application and make predictions to observe how the system processes them. Scenarios include datasets with incorrect dimensions, missing values, and non-numeric values to ensure the application properly rejects or handles them. Additionally, we test datasets adhering to the specified format to ensure accurate predictions. Manual testing here is effective as it simulates real user interactions, allowing us to observe the system's behaviour in real-time and ensure that error messages are user-friendly and informative. This hands-on approach helps us understand the user experience and identify usability issues that might not be apparent through automated testing. ***Upload testing*** is another critical component, involving simulated delay testing where we manually upload datasets and simulate delays to observe the application's responsiveness. We ensure predictions can only be made once uploads are complete, preventing accidental predictions on incomplete datasets. Upload logical testing verifies the application's ability to prioritize the most recently uploaded dataset for predictions, ensuring it always uses the latest dataset. Manual testing in this area helps assess the real-time performance and reliability of the upload process, simulating various network conditions and user behaviours that might affect it. This thorough evaluation helps us identify and address potential issues that could disrupt the user experience. In the ***prediction and prediction output testing*** phase, we manually trigger predictions using the web application and observe if the model is correctly called and loaded. We verify the output and storage of prediction results, ensuring they are correctly stored and retrievable by the user. Manual testing allows direct interaction with the prediction functionality, ensuring the process is intuitive and error-free. It helps identify any issues in the prediction workflow and ensures the results are accurately stored and displayed. To test the ***accuracy and reliability of the prediction results,*** we manually use unseen datasets to make predictions with the trained model and compare the outputs with known results. Manual testing enables controlled evaluation of the model's performance. By manually selecting unseen datasets, we can closely monitor the prediction results and assess their accuracy. This approach allows us to fine-tune the model, ensuring it provides reliable and valid predictions for new data, capturing nuanced issues that automated tests might overlook, especially with complex biological data.

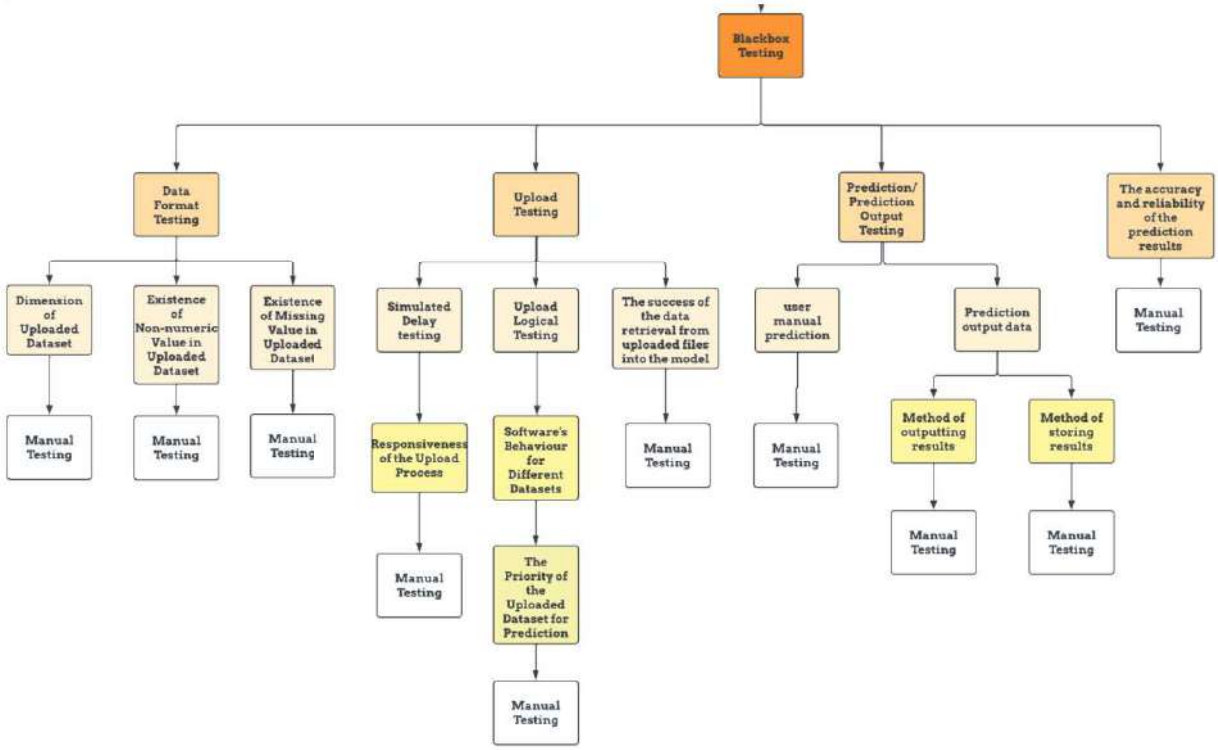


Figure 57: Breakdown of Test Approach of Blackbox Testing

Figure 58 shows the breakdown and structure of our **Integration Testing** process. Integration testing ensures that various components of a web application work together seamlessly. Our approach relies heavily on **manual testing**, which offers several unique advantages despite the availability of automated tools. Below, we outline our manual testing methodology for key integration scenarios and explain its effectiveness in these contexts.

The first step in our integration testing involves manually ***incorporating the predictive model into the web application***. We repeatedly trigger the model loading process to ensure it loads within acceptable time limits and operates reliably, making sure that the model can be called in well manner. This hands-on approach allows us to closely monitor performance, identify any delays or failures, and ensure smooth integration. Manual testing is ideal here because it captures minor issues, such as occasional load failures, that automated tests might overlook but can significantly impact the user experience. ***Preprocessing user-uploaded data*** is crucial to ensure it aligns with the model's requirements. We manually upload datasets with known characteristics and monitor the preprocessing steps applied by the web application. We verify data transformations, such as normalisation, scaling, and format adjustments, by inspecting logs and intermediary data outputs. Manual testing allows for a thorough examination of preprocessing steps, including handling edge cases, ensuring consistent and correct data preparation before model ingestion. The ***integration of prediction results into the web application*** is another critical focus. We manually execute predictions and observe how the results are processed and displayed. This involves verifying that prediction results are correctly stored and that the correct result paths are read and displayed in the web application. Manual testing enables direct interaction with the application, ensuring accurate reflection of prediction results in the user interface and identifying potential issues in the storage and retrieval processes. Finally, we ensure that the ***visualisation of prediction results*** functions correctly within the web application. We manually trigger the visualisation process after a prediction is made, verifying that the results are correctly passed to the visualisation module and displayed accurately. Manual testing is crucial here because visualisations often involve dynamic and interactive components that require human judgement to assess. By manually testing these visualisations, we ensure the visual representation of data is accurate and intuitive for the end-user.

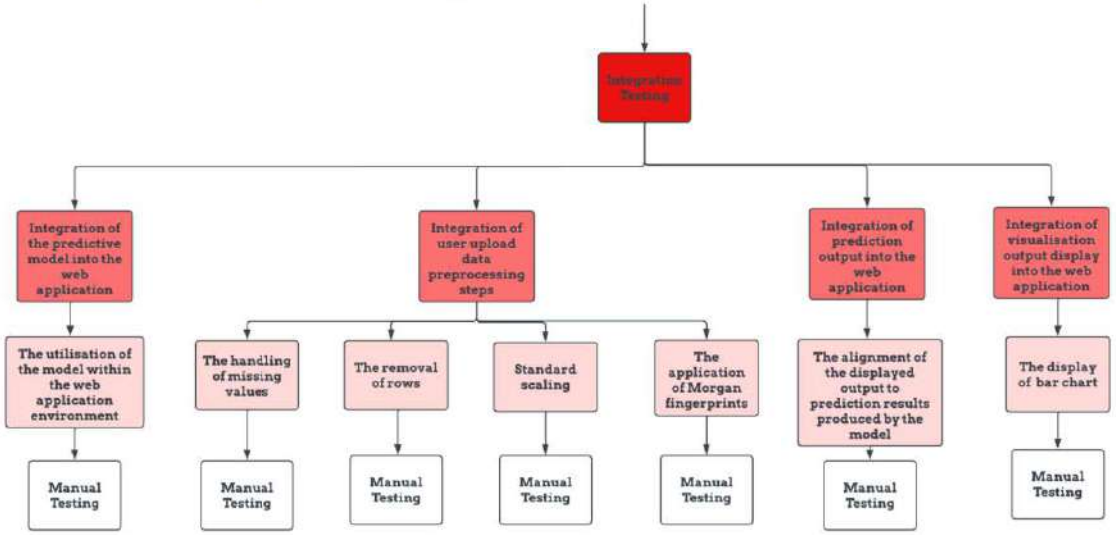


Figure 58: Breakdown of Test Approach of Integration Testing

Lastly, **Figure 59** shows the breakdown and structure of our usability testing. Usability testing is crucial for evaluating the user experience (UX) and user interface (UI) of a web application, ensuring it is intuitive, efficient, and user-friendly. Our approach relies heavily on **manual testing**, leveraging human judgement and interaction to assess various aspects of usability. Below, we outline our manual testing methodology for key usability scenarios and explain why manual testing is essential.

User interface navigation testing assesses the ease of navigation within the web application. Testers interact with the application's interface, navigating through different sections and features as an end user would. They evaluate the intuitiveness of the navigation flow, determining how easily they can transition between pages, access user guidelines, and return to the default interface. This hands-on approach helps identify navigation issues, such as confusing menu structures or hidden features, that could hinder the user experience. ***Testing upload and export functionality*** involves manually assessing these features' usability. Testers upload datasets to the application, evaluating the clarity of instructions provided and the ease of the upload process. They also test the export functionality to ensure users can

easily export data in various formats. Manual testing simulates real user interactions, providing insights into potential usability issues like unclear instructions or technical glitches. ***Table sorting and filtering testing*** is crucial for evaluating data manipulation features' usability. Testers manually interact with tables, sorting and filtering data to assess the responsiveness and efficiency of these functionalities. They ensure users can easily manipulate data to find relevant information, identifying performance issues or usability challenges. Manual testing allows for exploring different scenarios and edge cases, ensuring thorough evaluation of table sorting and filtering features. ***Testing visualisation interactive features***, such as filtering and tooltips, involves manual interaction with visual elements. Testers assess these features' responsiveness and functionality, evaluating how effectively users can interact with visualisations to gain insights from the data. Manual testing helps identify issues with interactivity or usability that could impact the overall user experience. ***Third-party testing*** involves soliciting feedback from external users to gather diverse perspectives on the application's usability. We prepare test materials and guides, such as user scenarios and testing instructions, and generate feedback forms using platforms like Google Forms. We then distribute these forms to external users, such as peers, friends, or family members, and collect feedback on various aspects of the application, including software configuration and usability issues. Manual testing facilitates direct engagement with external users, collecting qualitative feedback and insights that can inform usability improvements.

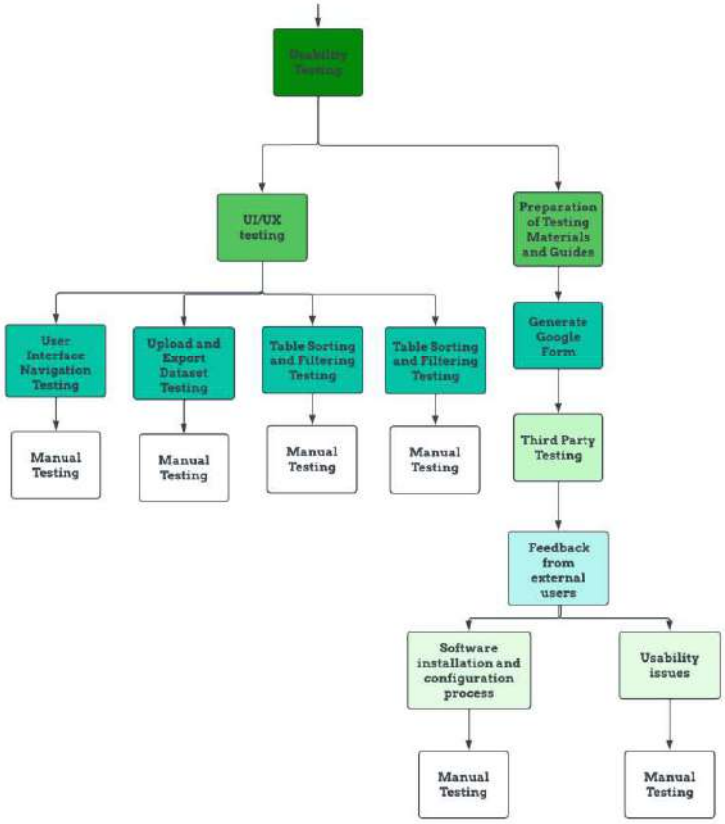


Figure 59: Breakdown of Test Approach of Usability Testing

In conclusion, manual testing is the preferred approach for our testing due to its ability to provide comprehensive insights into the application's behaviour, user experience, and overall functionality. Through manual testing, we can ensure that our drug resistance prediction web application delivers accurate, reliable, and user-friendly performance, ultimately enhancing its value to users in the medical and scientific community.

Section 4: Blackbox Testing

Section 4.1: User Data Format Testing

(a) Test Description (What is being tested)

The User Data Format Testing is designed to comprehensively evaluate the **handling of various data formats** within the software application. The primary objective of this test is to **ensure that the user input data format aligns with the requirements for successful integration into the predictive model.** Firstly, the test evaluates the software's ability to handle erroneous data formats, such as datasets with **incorrect dimensions**. If the uploaded data does not match the expected format, indicated by **missing columns or a mismatched number of columns** compared to the model's requirements (19226 columns), it should be promptly rejected. This ensures that only data conforming to the specified format is utilised for prediction, preventing potential errors in model training or inference. Additionally, the test assesses the software's capability to **manage datasets containing NA (missing) values**. In such scenarios, the software should effectively handle missing values by removing corresponding rows during prediction generation, ensuring that the prediction output remains accurate and reliable. Also, **data with non-numeric values** in the gene expression columns (starting from column 5 to column 19225 of user data) will be **rejected** as those are the training features, and it needs to be numeric in order to be able to execute the prediction action. Conversely, when the input dataset **adheres to the expected format** without missing values or dimension mismatches, the software should seamlessly **proceed with normal prediction** processes. By conducting thorough testing on various data formats and scenarios, this testing phase aims to validate the software's robustness in handling diverse user input data formats while maintaining the integrity and accuracy of prediction results.

(b) Test Set-Up/ Methodology (How it is being tested)

There are several potential data format issues to consider, but our focus is primarily on four specific scenarios for testing the software.

Scenario 1: The User Data has an incorrect dimension, failing to meet the specified requirement of 19226 columns outlined in the User Guide's data format specifications

To emulate this scenario, we will upload a dataset with an incorrect dimension into the web application, deviating from the specified requirement of 19226 columns as outlined in the User Guide. Once the dataset is uploaded, we will proceed to trigger the "Make Prediction" action within the web application interface. Subsequently, we will closely observe the software's response, particularly focusing on the terminal output. Our objective is to ascertain whether the software is able to detect and appropriately handle the dataset with an incorrect dimension. We will pay particular attention to any error messages generated during the prediction process, which may indicate issues related to the dataset's dimensionality. Additionally, we will inspect the web application interface to determine if any prediction results are displayed. The absence of prediction results or the presence of error messages would suggest that the software is effectively identifying and rejecting datasets that do not conform to the specified dimension requirement.

Scenario 2: User Data contains missing values (NA)

To simulate this scenario, we will upload a dataset containing missing values in some of the cells. Upon uploading the dataset, we will proceed to initiate the prediction process by clicking on the "Make Prediction" button within the web application. Similarly, we will carefully observe the terminal output to determine if any error messages are generated during the prediction process due to the presence of missing values in the dataset. The focus will be on ensuring that the software effectively handles missing values by excluding corresponding rows during the prediction generation process. Subsequently, we will also inspect the web application interface to determine if any prediction results are displayed.

Scenario 3: User Data contains any non-numeric value in the 5th column to the 19225th column

To replicate this scenario, we will upload a dataset containing non-numeric values in the specified range from the 5th column to the 19225th column. Following the upload, we will execute the prediction process by initiating the "Make Prediction" action. Next, we will closely monitor the terminal output for any error messages indicating issues related to non-numeric data processing. Additionally, we will inspect the web application interface to identify if any prediction result is displayed on the web.

Scenario 4: User Data strictly follows the format specified in the user guide, correct dimension, no missing or non-numeric values

In this scenario, we will upload a dataset that strictly adheres to the format specified in the user guide, with the correct dimension of 19226 columns, no missing values and no non-numeric values for gene-expressions columns. After uploading the dataset, we will proceed with initiating the prediction process by clicking on the "Make Prediction" button. We will observe the terminal output to ensure that prediction is successfully made, and observation of web interface will be conducted as well.

(c) Inputs to the code or part of code being tested

Below are the codes utilised to conduct data format checks. In the "model.py" file illustrated in **Figure 60**, there is a process of dropping rows containing NA values to address missing values by removing them during prediction. As for the dimension problem and non-numeric values, they are automatically managed by the model itself since X_features can only contain numeric values and the testing and training data have the same dimension. Therefore, no additional steps are required to conduct checks for these two cases.

```
18  def run_model(model, df_upload):
104
105     initial_row_count = prediction_df.shape[0]
106
107     prediction_df.dropna(inplace=True)
108
109     final_row_count = prediction_df.shape[0]
110     dropped = False
111     rows_dropped = initial_row_count - final_row_count
112     if rows_dropped > 0:
113         dropped  = True
114
115
116     print(prediction_df)
117
118     return dropped, prediction_df
```

Figure 60: Code segment of "run_model" function which perform NA rows dropping actions

Figure 61 exhibits the code responsible for printing error messages in the web application, executed within the "make_prediction" function in the "app.py" file.

```
55  def make_prediction():
76      # Run the model and handle exceptions gracefully
77      try:
78          dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)
79
80          # if prediction_df is None:
81          #     return jsonify({'error': prediction_df}), 400  # Return a bad request with the error message
82
83          # prediction_df = run_model(MODEL_FOLDER, csv_path)
84          timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  # Correctly access datetime class
85          result_filename = f'user_prediction_result_{timestamp}.csv'
86          result_path = os.path.join(app.config['RESULT_FOLDER'], result_filename)
87          prediction_df.to_csv(result_path, index=False)
88          if dropped:
89              return jsonify({'message': 'Prediction completed, Rows with NA values have been dropped', 'result_filename': result_filename})
90          else:
91              return jsonify({'message': 'Prediction completed successfully', 'result_filename': result_filename})
92      except Exception as e:
93          print(f"Error during model prediction: {e}")
94          return jsonify({'message': 'Wrong Dimension Data or Non-numeric Values Detected, Please Input Correct Data', 'details': str(e)}), 500
```

Figure 61: Code segment of "make_prediction" function which manages the printing of error messages

For the execution of the testing process, there will be four files being uploaded to the web application for testing purposes:

Scenario 1: Input user file "sample_user_upload_data_wrong_dim.csv"
Scenario 2: Input user file "sample_user_upload_data_with_NA.csv"
Scenario 3: Input user file "non-numeric_gene_data.csv"
Scenario 4: Input user file "sample_user_upload_2.csv"

These four files will be passed into the "run_model" function as the input parameter "df_upload" to trigger the model prediction on the respective data uploaded.

(d) Expected Outputs

In **scenario 1** and **scenario 3**, the expected outcome entails an **error prompt in the terminal** during prediction, indicating **rejection of the input user data** due to format errors. Consequently, **no prediction output** will be displayed in either the **web interface or the terminal**.

Regarding **scenario 2**, the expectation is that the model **can execute predictions** on the input data; however, **rows with NA values will be dropped.** Consequently, while **prediction outputs will be displayed** in both the **web interface** and the **terminal**, the output will **exclude** predictions for **rows with NA values**.

Lastly, for **scenario 4**, a smooth prediction process is anticipated, resulting in **prediction outputs being displayed in both the terminal and the web application**.

(e) Actual Outputs and Discussions

Scenario 1:

Figure 62 shows that the terminal output reveals an error message indicating a discrepancy in the received input shape by the predictive model. The expected input shape for the predictive model is 19477 columns, yet the received input shape is (32, 19475). This discrepancy arises from the user's input data, which possesses a shape of 19224 columns, augmented by 256 columns of binary Morgan fingerprints after preprocessing. Following the removal of 5 columns (drug name, drug id, cosmic id, cell line name, isosmiles), the resultant input shape amounts to 19475 columns. Consequently, an error occurs during model prediction when calling Sequential.call(). **Figure 63** illustrates that no output is displayed in the web interface, accompanied by an error message stating, "Error during prediction: Wrong Dimension Data or Non-numeric Values Detected, Please Input Correct Data". Therefore, it is proven that Scenario 1 has **PASSED** the test case, showing proper handling of data with the wrong dimension.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_data_wrong_dim.csv'}
Constructed CSV path: user_file\sample_user_upload_data_wrong_dim.csv
       DRUG_NAME  DRUG_ID  COSMIC_ID  CELL_LINE_NAME  ...  ELOA3D (100506888)  ELOA3 (162699)  CDR1 (1038)                                          isosmiles
0     Linsitinib     1510     905946     MCF7_BREAST  ...                 0.0             0.0          0.0  CC1(CC(C1)C2=NC(=C3N2C=CN=C3N)C4=CC5=C(C=C4)C=...
1    Afuresertib     1912     906851     EFM19_BREAST  ...                 0.0             0.0          0.0  CN1C(=C(C=N1)Cl)C2=C(SC(=C2)C(=O)N[C@@H](CC3=C...
2      SCH772984     1564     910910  UACC812_BREAST  ...                 0.0             0.0          0.0  C1CN(C[C@@H]1C(=O)NC2=CC3=C(C=C2)NN=C3C4=CC=NC...
3         UMI-77     1939     905946     MCF7_BREAST  ...                 0.0             0.0          0.0  C1=CC=C2C(=C1)C(=CC(=C2O)SCC(=O)O)NS(=O)(=O)C3...
4          MN-64     1854     909907   ZR7530_BREAST  ...                 0.0             0.0          0.0           CC(C)C1=CC=C(C=C1)C2=CC(=O)C3=CC=CC=C302
..           ...      ...        ...             ...  ...                 ...             ...          ...                                                ...
269       LGK974     1598     749709  HCC1954_BREAST  ...                 0.0             0.0          0.0  CC1=CC(=CN=C1C2=CC(=NC=C2)C)CC(=O)NC3=NC=C(C=C...
270    Lapatinib     1558     910704    AU565_BREAST  ...                 0.0             0.0          0.0  CS(=O)(=O)CCNCC1=CC=C(O1)C2=CC3=C(C=C2)N=CN=C3...
271      AZD8186     1918     906826   CAL120_BREAST  ...                 0.0             0.0          0.0  C[C@H](C1=CC(=CC2=C1OC(=CC2=O)N3CCOCC3)C(=O)N(...
272  Refametinib     1014     910704    AU565_BREAST  ...                 0.0             0.0          0.0  COC1=CC(=C(C(=C1NS(=O)(=O)C2(CC2)C[C@@H](CO)O)...
273     YK-4-279     1239     910852   CAL851_BREAST  ...                 0.0             0.0          0.0  COC1=CC=C(C=C1)C(=O)CC2(C3=C(C=CC(=C3NC2=O)Cl)...

[274 rows x 19224 columns]
2024-05-17 01:30:08.029152: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in per
formance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the
model.
Error during model prediction: Exception encountered when calling Sequential.call().

Input 0 of layer "dense" is incompatible with the layer: expected axis -1 of input shape to have value 19477, but received input with shape (32, 19475)

Arguments received by Sequential.call():
  • inputs=tf.Tensor(shape=(32, 19475), dtype=float32)
  • training=False
  • mask=None
```

Figure 62: Terminal output of Scenario 1

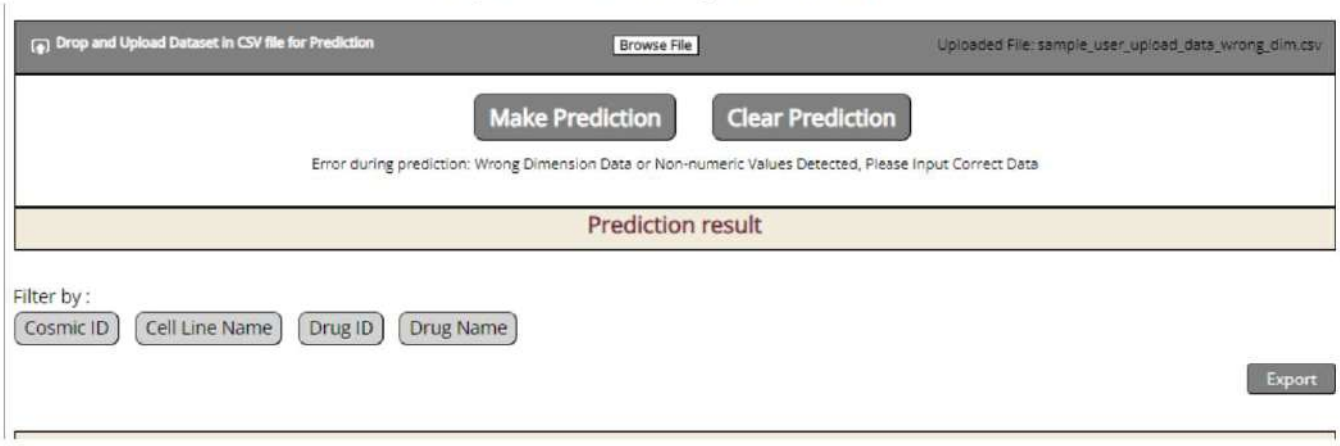


Figure 63: Web Interface Output of Scenario 1

Scenario 2:

In **Figure 64**, the input user data's first row exhibits missing values (NaN), notably at the columns "ELOA3 (162699)" and "CDR1 (1038)". Despite these missing values, the model successfully conducts predictions by dropping rows containing NaN values. For example, the terminal output illustrates that the row with DRUG_ID = 1003 is affected by NaN values. Upon observing the prediction output in **Figure 65**, it becomes evident that the row corresponding to DRUG_ID = 1003 has been eliminated due to its NaN values. Initially comprising 633 rows, the input data is reduced to 631 rows after the removal of two rows with missing values. In **Figure 66**, the web interface effectively displays the prediction results. Notably, the output table indicates the absence of prediction results pertaining to DRUG_ID = 1003. A message confirming successful prediction is generated, accompanied by a notification regarding the removal of rows with NaN values. Consequently, Scenario 2 successfully **PASSES** the test case, aligning with the expected outcome whereby predictions can still be made, albeit with rows containing missing values being dropped.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_data_with_NA.csv'}
Constructed CSV path: user_file\sample_user_upload_data_with_NA.csv
       DRUG_NAME  DRUG_ID  COSMIC_ID   CELL_LINE_NAME  ...  ELOA3D (100506888)  ELOA3 (162699)  CDR1 (1038)                                                isosmiles
0     Camptothecin     1003     1290922     HDQP1_BREAST  ...            0.042644             NaN          NaN  CC[C@@]1(C2=C(COC1=O)C(=O)N3CC4=CC5=CC=CC=C5N=...
1      Vinblastine     1004     1290922     HDQP1_BREAST  ...            0.042644        0.042644     0.000000  CC[C@@]1(CC2C[C@@](C3=C(CCN(C2)C1)C4=CC=CC=C4N...
2       Cisplatin     1005     1290922     HDQP1_BREAST  ...            0.042644        0.042644     0.000000                                  N.N.[Cl-].[Cl-].[Pt+2]
3      Cytarabine     1006     1290922     HDQP1_BREAST  ...            0.042644        0.042644     0.000000  C1=CN(C(=O)N=C1N)[C@H]2[C@H]([C@@H]([C@H](O2)C...
4       Docetaxel     1007     1290922     HDQP1_BREAST  ...            0.042644        0.042644     0.000000  CC1=C2[C@H](C(=O)[C@@]3([C@H](C[C@@H]4[C@]([C@...
..            ...      ...         ...              ...  ...                 ...             ...          ...                                                ...
628      AZD6482     2169      905960  MDAMB231_BREAST  ...            0.000000        0.000000     0.097611  CC1=CN2C(=O)C=C(N=C2C(=C1)[C@@H](C)NC3=CC=CC=C...
629          JQ1     2172      905960  MDAMB231_BREAST  ...            0.000000        0.000000     0.097611  CC1=C(SC2=C1C(=N[C@H](C3=NN=C(N32)C)CC(=O)OC(C...
630        PFI-1     2173      905960  MDAMB231_BREAST  ...            0.000000        0.000000     0.097611    CN1CC2=C(C=CC(=C2)NS(=O)(=O)C3=CC=CC=C3OC)NC1=O
631      SGC0946     2177      905960  MDAMB231_BREAST  ...            0.000000        0.000000     0.097611  CC(C)N(CCCNC(=O)NC1=CC=C(C=C1)C(C)(C)C)C[C@@H]...
632   GSK2830371     2359      905960  MDAMB231_BREAST  ...            0.000000        0.000000     0.097611  CC1=C(C=C(C=N1)Cl)NCC2=CC=C(S2)C(=O)N[C@@H](CC...

[633 rows x 19226 columns]
```

Figure 64: Terminal Output of Data Reading of Scenario 2

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
     COSMIC_ID   CELL_LINE_NAME  DRUG_ID     DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
1      1290922     HDQP1_BREAST     1004    Vinblastine     -3.285442               Low
2      1290922     HDQP1_BREAST     1005     Cisplatin      2.992935               Low
3      1290922     HDQP1_BREAST     1006     Cytarabine     1.399642               Low
4      1290922     HDQP1_BREAST     1007      Docetaxel    -3.996065               Low
5      1290922     HDQP1_BREAST     1008    Methotrexate    -1.759563               Low
..         ...              ...      ...            ...          ...               ...
628     905960  MDAMB231_BREAST     2169       AZD6482      2.149139               Low
629     905960  MDAMB231_BREAST     2172           JQ1      2.388358               Low
630     905960  MDAMB231_BREAST     2173         PFI-1      3.019472               Low
631     905960  MDAMB231_BREAST     2177       SGC0946      3.397459               Low
632     905960  MDAMB231_BREAST     2359    GSK2830371      3.562152               Low

[631 rows x 6 columns]
127.0.0.1 - - [17/May/2024 01:38:41] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:38:41] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 01:38:41] "GET /results/user_prediction_result_20240517013841.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:38:41] "GET /results/user_prediction_result_20240517013841.csv HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 01:38:41] "GET /results/user_prediction_result_20240517013841.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:38:41] "GET /results/user_prediction_result_20240517013841.csv HTTP/1.1" 304 -
```

Figure 65: Terminal Output of Prediction Result of Scenario 2

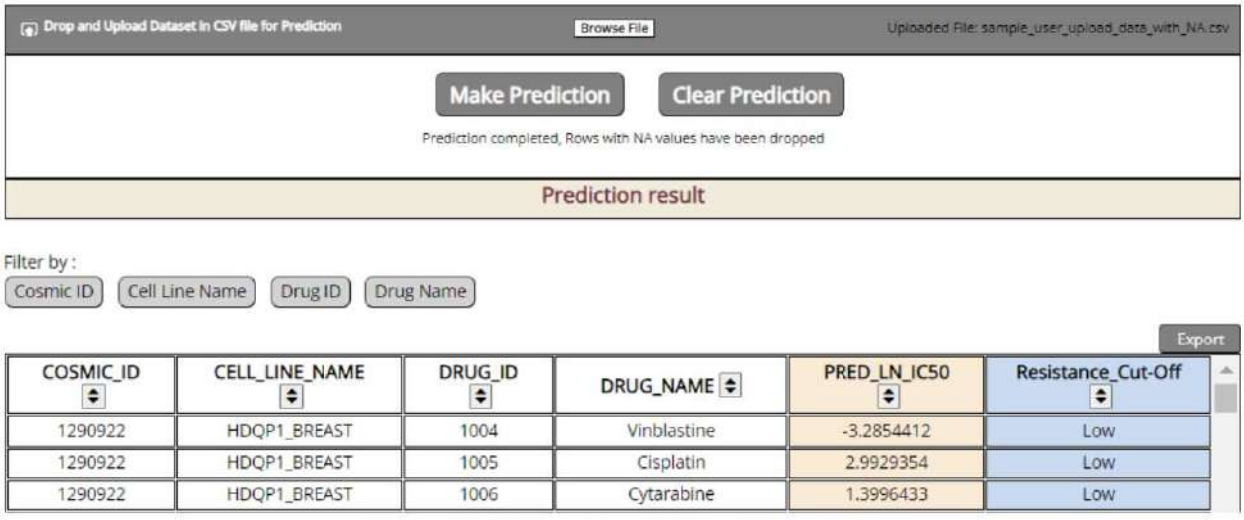


Figure 66: Web Interface Output of Scenario 2

Scenario 3:

In **Figure 67**, the user input data shows non-numeric values in the gene expression columns, specifically "ELOA3 (162699)" and "CDR1 (1038)", which contain the values “def” and “abc”. Upon attempting to make a prediction, the terminal outputs an error message: “Error during model prediction: could not convert string to float”. This error occurs because the predictive model's X_features require numeric values, and the presence of non-numeric values prevents the model from executing, resulting in no prediction being made or outputted in the terminal. In **Figure 68**, it is evident that no prediction table is displayed, and an error message prompts the user to re-upload the correct file. Consequently, Scenario 3 aligns with the expected outcome, the test cases have **PASSED**, demonstrating that the model cannot handle non-numeric input. The display of the error message in the web interface further confirms that the software appropriately handles the case of user input with non-numeric values in the gene expression columns.

```
Received JSON data: {'uploaded_filename': 'non-numeric_gene_data.csv'}
Constructed CSV path: user_file\non-numeric_gene_data.csv
      DRUG_NAME  DRUG_ID  COSMIC_ID  CELL_LINE_NAME  ... ELOA3D (100506888)  ELOA3 (162699)  CDR1 (1038)                                  isosmiles
0   Camptothecin     1003     749710  HCC1143_BREAST  ...           0.000000             def          abc  CC[C@@]1(C2=C(COC1=O)C(=O)N3CC4=CC5=CC=CC=C5N=...
1    Vinblastine     1004     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  CC[C@@]1(CC2C[C@@](C3=C(CCN(C2)C1)C4=CC=CC=C4N...
2      Cisplatin     1005     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0                                N.N.[Cl-].[Cl-].[Pt+2]
3     Cytarabine     1006     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  C1=CN(C(=O)N=C1N)[C@H]2[C@H]([C@@H]([C@H](O2)C...
4      Docetaxel     1007     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  CC1=C2[C@H](C(=O)[C@@]3([C@H](C[C@@H]4[C@]([C@...
5   Methotrexate     1008     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  CN(CC1=CN=C2C(=N1)C(=NC(=N2)N)N)C3=CC=C(C=C3)C...
6      Gefitinib     1010     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  COC1=C(C=C2C(=C1)N=CN=C2NC3=CC(=C(C=C3)F)Cl)OC...
7     Navitoclax     1011     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  CC1(CCC(=C(C1)CN2CCN(CC2)C3=CC=C(C=C3)C(=O)NS(...
8     Vorinostat     1012     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0                  C1=CC=C(C=C1)NC(=O)CCCCCCC(=O)NO
9      Nilotinib     1013     749710  HCC1143_BREAST  ...           0.000000             0.0          0.0  CC1=C(C=C(C=C1)C(=O)NC2=CC(=CC(=C2)C(F)(F)F)N3...
10  Camptothecin     1003     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CC[C@@]1(C2=C(COC1=O)C(=O)N3CC4=CC5=CC=CC=C5N=...
11   Vinblastine     1004     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CC[C@@]1(CC2C[C@@](C3=C(CCN(C2)C1)C4=CC=CC=C4N...
12     Cisplatin     1005     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0                                N.N.[Cl-].[Cl-].[Pt+2]
13    Cytarabine     1006     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  C1=CN(C(=O)N=C1N)[C@H]2[C@H]([C@@H]([C@H](O2)C...
14     Docetaxel     1007     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CC1=C2[C@H](C(=O)[C@@]3([C@H](C[C@@H]4[C@]([C@...
15  Methotrexate     1008     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CN(CC1=CN=C2C(=N1)C(=NC(=N2)N)N)C3=CC=C(C=C3)C...
16     Gefitinib     1010     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  COC1=C(C=C2C(=C1)N=CN=C2NC3=CC(=C(C=C3)F)Cl)OC...
17    Navitoclax     1011     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CC1(CCC(=C(C1)CN2CCN(CC2)C3=CC=C(C=C3)C(=O)NS(...
18    Vorinostat     1012     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0                  C1=CC=C(C=C1)NC(=O)CCCCCCC(=O)NO
19     Nilotinib     1013     906801      BT20_BREAST  ...           0.014355  0.01435529297707          0.0  CC1=C(C=C(C=C1)C(=O)NC2=CC(=CC(=C2)C(F)(F)F)N3...

[20 rows x 19226 columns]
Error during model prediction: could not convert string to float: 'abc'
127.0.0.1 - - [17/May/2024 01:45:45] "POST /predict HTTP/1.1" 500 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:45:45] "POST /predict HTTP/1.1" 500 -
```

Figure 67: Terminal output of Scenario 3

Figure 68: Web Interface Output of Scenario 3

Scenario 4:

In this scenario, since the user-uploaded data has the correct input data dimensions, no missing values, and no non-numeric values in the gene expression columns, the terminal output shown in **Figure 69** confirms that the input file has the correct number of 19,226 columns. **Figure 70** shows a successful prediction on the input data, with the prediction output containing the same number of rows as the input data. **Figure 71** demonstrates the successful display of the prediction results in the web application, with the message “Prediction completed successfully.” This outcome confirms that the scenario has passed the test case, proving that if the data adheres to the format outlined in Section 1.3: User Upload Data Format Requirements, a successful prediction will be made, with no missing data triggered.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_2.csv'}
Constructed CSV path: user_file\sample_user_upload_2.csv
        DRUG_NAME  DRUG_ID  COSMIC_ID   CELL_LINE_NAME  ...  ELOA3D (100506888)  ELOA3 (162699)  CDR1 (1038)                                          isosmiles
0       Docetaxel     1007     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  CC1=C2[C@H](C(=O)[C@@]3([C@H](C[C@@H]4[C@](C@...
1       Gefitinib     1010     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  COC1=C(C=C2C(=C1)N=CN=C2NC3=CC(=C(C=C3)F)Cl)OC...
2        KU-55933     1030     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  C1COCCN1C2=CC(=O)C=C(O2)C3=C4C(=CC=C3)SC5=CC=C...
3        Afatinib     1032     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  CN(C)C/C=C/C(=O)NC1=C(C=C2C(=C1)C(=NC=N2)NC3=C...
4  Wee1 Inhibitor     1046     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  C1=CC=C(C(=C1)C2=CC3=C(C4=C(N3)C=CC(=C4)O)C5=C...
5       PD0325901     1060     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  C1=CC(=C(C=C1I)F)NC2=C(C=CC(=C2F)F)C(=O)NOC[C@...
6     Ruxolitinib     1507     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  C1CCC(C1)[C@@H](CC#N)N2C=C(C=N2)C3=C4C=CNC4=NC=N3
7       Cediranib     1922     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  CC1=CC2=C(N1)C=CC(=C2F)OC3=NC=NC4=CC(=C(C=C43)...
8      NVP-ADW742     1932     909778   UACC893_BREAST  ...                 0.0             0.0     0.000000  C1CCN(C1)CC2CC(C2)N3C=C(C4=C(N=CN=C43)N)C5=CC(...
9       Docetaxel     1007     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  CC1=C2[C@H](C(=O)[C@@]3([C@H](C[C@@H]4[C@](C@...
10      Gefitinib     1010     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  COC1=C(C=C2C(=C1)N=CN=C2NC3=CC(=C(C=C3)F)Cl)OC...
11       KU-55933     1030     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  C1COCCN1C2=CC(=O)C=C(O2)C3=C4C(=CC=C3)SC5=CC=C...
12       Afatinib     1032     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  CN(C)C/C=C/C(=O)NC1=C(C=C2C(=C1)C(=NC=N2)NC3=C...
13 Wee1 Inhibitor     1046     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  C1=CC=C(C(=C1)C2=CC3=C(C4=C(N3)C=CC(=C4)O)C5=C...
17     NVP-ADW742     1932     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  C1CCN(C1)CC2CC(C2)N3C=C(C4=C(N=CN=C43)N)C5=CC(...
18     GSK2830371     2359     905960  MDAMB231_BREAST  ...                 0.0             0.0     0.097611  CC1=C(C=C(C=N1)C1)NCC2=CC=C(S2)C(=O)N[C@@H](CC...

[19 rows x 19226 columns]
```

Figure 69: Terminal Output of Data Reading of Scenario 4

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
    COSMIC_ID   CELL_LINE_NAME  DRUG_ID       DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0      909778   UACC893_BREAST     1007       Docetaxel     -3.387078               Low
1      909778   UACC893_BREAST     1010       Gefitinib      2.238444               Low
2      909778   UACC893_BREAST     1030        KU-55933      2.742792               Low
3      909778   UACC893_BREAST     1032        Afatinib      0.728675               Low
4      909778   UACC893_BREAST     1046  Wee1 Inhibitor      2.679244               Low
5      909778   UACC893_BREAST     1060       PD0325901      2.148164               Low
6      909778   UACC893_BREAST     1507     Ruxolitinib      3.135053               Low
7      909778   UACC893_BREAST     1922       Cediranib      2.229212               Low
8      909778   UACC893_BREAST     1932      NVP-ADW742      1.784981               Low
9      905960  MDAMB231_BREAST     1007       Docetaxel     -2.212940               Low
10     905960  MDAMB231_BREAST     1010       Gefitinib      3.665546               Low
11     905960  MDAMB231_BREAST     1030        KU-55933      3.778869              High
12     905960  MDAMB231_BREAST     1032        Afatinib      3.164263               Low
13     905960  MDAMB231_BREAST     1046  Wee1 Inhibitor      3.627371               Low
14     905960  MDAMB231_BREAST     1060       PD0325901      3.495899               Low
15     905960  MDAMB231_BREAST     1507     Ruxolitinib      4.127828              High
16     905960  MDAMB231_BREAST     1922       Cediranib      3.580712               Low
17     905960  MDAMB231_BREAST     1932      NVP-ADW742      3.389481               Low
18     905960  MDAMB231_BREAST     2359      GSK2830371      3.784025              High
127.0.0.1 - - [17/May/2024 01:35:15] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:35:15] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 01:35:15] "GET /results/user_prediction_result_20240517013515.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:35:15] "GET /results/user_prediction_result_20240517013515.csv HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 01:35:15] "GET /results/user_prediction_result_20240517013515.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 01:35:15] "GET /results/user_prediction_result_20240517013515.csv HTTP/1.1" 304 -
```

Figure 70: Terminal Output of Prediction Result of Scenario 4

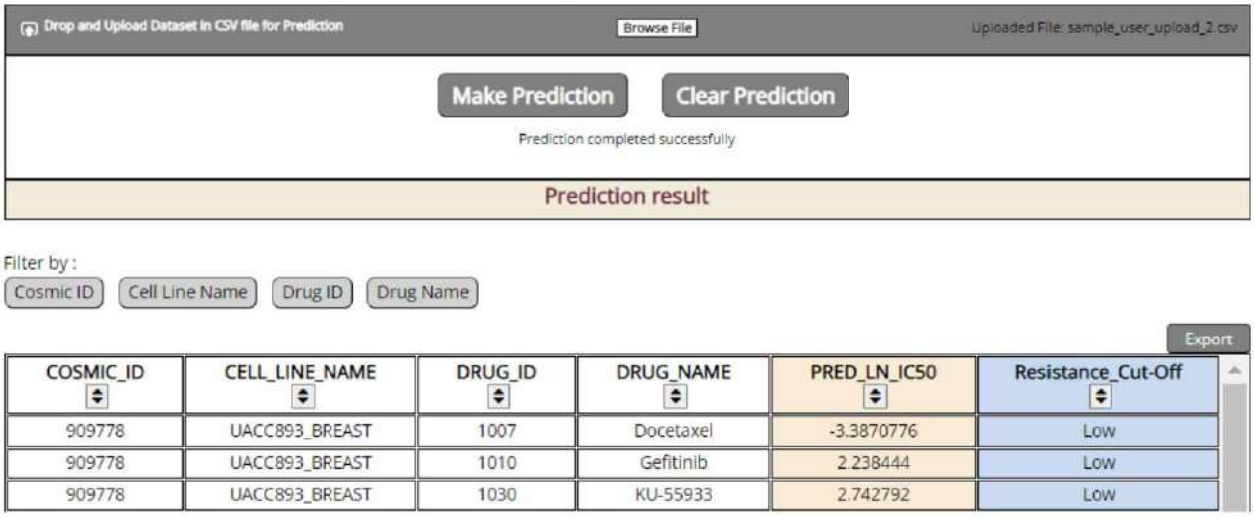


Figure 71: Web Interface Output of Scenario 4

Section 4.2: Upload Testing

Section 4.2.1: Simulated Delay Testing

(a) Test Description (What is being tested)

This test is designed to evaluate the application's handling of delays during the dataset upload and prediction process. Specifically, it aims to ensure that the **prediction model only runs after the dataset is fully uploaded** and that the application **provides a "loading" indicator while the prediction is being processed**. This is crucial for large file uploads, which may take significant time. The test will verify that if a user attempts to trigger a prediction before the upload is complete, the system correctly prevents the prediction process from starting. Additionally, the test will check that the user is informed of the ongoing prediction process through a loading message, ensuring clarity about when the prediction is complete.

(b) Test Set-Up/ Methodology (How it is being tested)

To set up this test, we will start by uploading a large dataset to the application and immediately pressing the "Make Prediction" button to see if the application attempts to predict before the file is fully uploaded. We will observe whether the application correctly prevents the prediction and provides feedback indicating the file is not being uploaded. Once the file is fully uploaded, we will press the "Make Prediction" button again and monitor the application for a "loading" message that indicates the prediction process is underway. We will ensure this message remains visible until the prediction is complete and the results are displayed. This test will help verify that the application handles the dataset upload and prediction processes efficiently, providing feedback to users and ensuring predictions are only made on fully uploaded datasets.

(c) Inputs to the code or part of code being tested

Figure 72 shows the function being used to read the user uploaded file. In this case, it will retrieve the filename and file path of the user uploaded file, and check if the file is successfully being uploaded.

```
// Function to handle and upload files
function handleFiles(files) {
    var formData = new FormData();
    for (var i = 0; i < files.length; i++) {
        formData.append('file', files[i]);
    }

    fetch('/upload', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {

        uploadedFileName.textContent = data.message;
        console.log(uploadedFileName.textContent);
    })
    .catch(error => {
        uploadedFileName.textContent = 'Error uploading file: ' + error.message;
    });
```

Figure 72: "handleFiles" function in web.js

Figure 73 shows the flask route for "upload" which stores the user uploaded file, if the data has not been successfully uploaded, there will be no file being uploaded to this route, therefore, the handleFiles function will be unable to fetch the file.

```python
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'message': 'No file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'message': 'No selected file'}), 400

    # filename = secure_filename(file.filename)
    filename = file.filename
    # Basic security check to remove path information
    filename = os.path.basename(filename)
    file_path = os.path.join(app.config['USER_FOLDER'], filename)
    file.save(file_path)

     # Save the filename in the user's session
    session['uploaded_filename'] = filename

    return jsonify({'message': f'Uploaded file: {filename}', 'filename': filename})
```

Figure 73: flask route for storing user uploaded data

(d) Expected Outputs

If the user presses "Make Prediction" immediately after selecting a file, before allowing sufficient time for the upload to complete, the software should display an error message indicating that the file has not yet been detected. After waiting a few more seconds, the user should press the "Make Prediction" button again. If the file has been successfully uploaded, the application should indicate that the prediction process has started by displaying a "LOADING…" message. This message informs the user that the application is processing the data. Once the prediction is complete, a "Prediction Completed" message should be displayed, confirming the successful completion of the process and updating the user on the prediction status. The entire process should show the effect of simulated delay, and acknowledge the users about the corresponding delays.

(e) Actual Outputs and Discussions

The file name is printed in the grey area as "uploaded file" as shown in **Figure 74** even though the file has not actually been uploaded yet. This results in an error message, "Error during prediction, file not found."

Figure 74: Web Interface when Prediction is made before file is uploaded

After waiting a few more seconds and clicking the "Make Prediction" button again, the prediction runs, displaying 'LOADING…' as shown in **Figure 75**. Once the prediction is completed, a message "Prediction completed successfully" is printed, indicating the successful status update of the predictions, as shown in **Figure 76**.

Figure 75: Showing Loading Status when prediction is being processing

Figure 76: Showing Prediction Completed Status when prediction is done

(f) Bug Identified and Limitations

A significant bug was identified when uploading large datasets. Although the uploaded file's name appears on the screen almost immediately, it takes several extra seconds for the file to be fully loaded and stored in the "user_file" folder. As shown in **Figure 77**, no file has been uploaded to the user_file yet. This indicates a lack of a delay message or action acknowledging that the file has not been fully uploaded. Currently, this is not a critical issue because users encountering this problem can wait a few more seconds for the file to upload before attempting to make a prediction again. The absence of a status update on the file's upload progress means users are unaware of whether the file has been successfully uploaded. This can lead to frustration and potential errors, as users might assume the file is ready for processing when it is not. To address this bug, we need to improve the upload process by ensuring the uploaded filename is only printed once the upload is complete. Although the software still functions if users wait a few extra seconds, refining this aspect will enhance user experience and reliability. We aim to resolve this issue and improve our software before the semester ends.

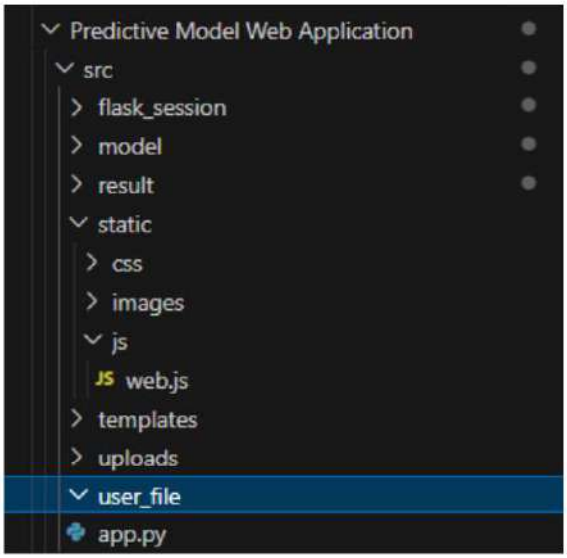


Figure 77: Empty folder after immediate upload

Section 4.2.2: Upload Logical Testing

(a) Test Description (What is being tested)

The Upload Logical Testing ensures the software application **processes and runs predictions on the most recently uploaded file.** This test verifies that predictions are made on the latest uploaded dataset, even if it shares the same filename as a previously uploaded file but contains different content. The objective is to **confirm the application's ability to differentiate between files with identical names and accurately select the most recent file for prediction**, thereby ensuring reliable and accurate results. This includes testing multiple uploads and predictions by the same user to ascertain that the latest upload is always used.

(b) Test Set-Up/ Methodology (How it is being tested)

To set up the Upload Logical Testing, begin by uploading an **initial** dataset named "same_name_dataset.csv"with **no NA values and correct dimension** to the application. Next, initiate the prediction process by clicking the "Make Prediction" button and carefully observe and record the terminal output and front-end outputs. Without clearing the initial prediction results, upload a new dataset with the same name, "same_name_dataset.csv", but containing **different content**, such that it **contains NA values, but the correct dimension**. Execute a new prediction by clicking the "Make Prediction" button again and observe the resulting terminal output and front-end outputs. Finally, compare the results of both predictions to verify that the software correctly selected the most recent dataset for execution, ensuring that predictions are based on the latest uploaded file even when filenames are identical.

(c) Inputs to the code or part of code being tested

Below are the codes utilised to conduct uploaded data checks. In the "app.py" file illustrated in **Figure 78**, we import a flask session (see line 2) and create a Session (see line 38). Do note that we import datetime (see line 8 & 9).

```
from flask import Flask, session, render_template, send_from_directory, jsonify, request
from flask_session import Session
from werkzeug.utils import secure_filename
import csv
import pandas as pd
from model.model import run_model
import os
from datetime import datetime, timedelta
import datetime
import shutil

app.secret_key = 'your_secret_key_here'  # Change to a strong secret key
app.config['SESSION_TYPE'] = 'filesystem'
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(days=1)

Session(app)
```

Figure 78: Code segment of "app.py" file where libraries are imported and session is created

In **Figure 79**, we have an "upload_file" function in "web.js" which calls session (see line 251) so that each user will have their own session where their own data will be stored in their respective session, that is each prediction will be uniquely stored. Whatever latest uploaded data will be the one that will be taken for prediction.

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'message': 'No file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'message': 'No selected file'}), 400

    # filename = secure_filename(file.filename)
    filename = file.filename
    # Basic security check to remove path information
    filename = os.path.basename(filename)
    file_path = os.path.join(app.config['USER_FOLDER'], filename)
    file.save(file_path)

    # Save the filename in the user's session
    session['uploaded_filename'] = filename

    return jsonify({'message': f'Uploaded file: {filename}', 'filename': filename})
```

Figure 79: Code segment of "upload_file" function in "web.js" which stacks the user's dataset

As mentioned above, datetime has been imported and used as shown in **Figure 80** (see line 84 & 85) in order to attach a specific timestamp to each file that is being uploaded.

```
def make_prediction():
    # Run the model and handle exceptions gracefully
    try:
        dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)

        timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  # Correctly access datetime class
        result_filename = f'user_prediction_result_{timestamp}.csv'
        result_path = os.path.join(app.config['RESULT_FOLDER'], result_filename)
        prediction_df.to_csv(result_path, index=False)
        if dropped:
            return jsonify({'message': 'Prediction completed, Rows with NA values have been dropped', 'result_filenam
        else:
            return jsonify({'message': 'Prediction completed successfully', 'result_filename': result_filename})
    except Exception as e:
        print(f"Error during model prediction: {e}")
        return jsonify({'message': 'Wrong Dimension Data or Non-numeric Values Detected, Please Input Correct Data',
```

Figure 80: Code segment of "make_prediction" function in "app.py" which runs the model and naming result files

By using unique sessions and timestamps, we can ensure that the latest uploaded files are read for making predictions, thus fulfilling the requirement of always making predictions on the most recent data.

(d) Expected Outputs

After the **first** upload and prediction, we expect the application to process the dataset and generate a successful prediction output. This output should be presented in the form of a table clearly displaying the predicted results based on the initial dataset. Additionally, the application should display a message, "Prediction Completed Successfully," to indicate that the prediction process has been completed without any issues. For the **second** upload and prediction, the expectations are slightly different. Once the second dataset is uploaded and the prediction process is initiated, the

application should again process all the data and produce a prediction output. This time, however, the application should display a message indicating that rows with NA values have been dropped. This differentiation ensures that despite having the same file name, the most recent dataset is correctly read and used for prediction.

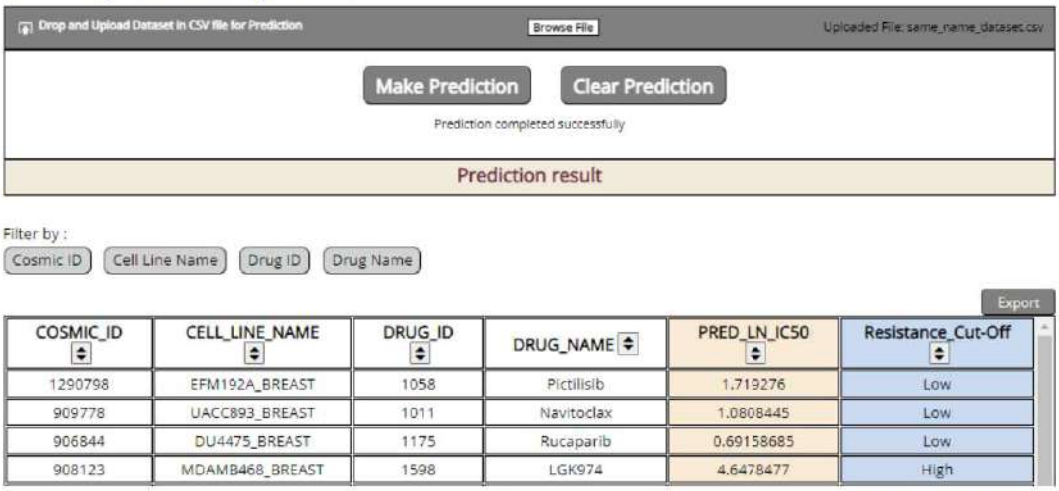
(e) Actual Outputs and Discussions

In **Figure 81,** the terminal output displays the expected and accurate output for the first dataset, "same_name_dataset.csv". It indicates that the first dataset comprises 21 rows, numbered from 0 to 20.

```
Received JSON data: {'uploaded_filename': 'same_name_dataset.csv'}
Constructed CSV path: user_file\same_name_dataset.csv
2024-05-17 00:29:48.475967: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
   COSMIC_ID   CELL_LINE_NAME  DRUG_ID         DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0    1290798    EFM192A_BREAST     1058       Pictilisib      1.719276               Low
1     909778    UACC893_BREAST     1011       Navitoclax      1.080845               Low
2     906844     DU4475_BREAST     1175        Rucaparib      0.691587               Low
3     908123  MDAMB468_BREAST     1598          LGK974      4.647848              High
4     908122  MDAMB453_BREAST     1059           AZD8055      1.692313               Low
5     910852     CAL851_BREAST     1997          WEHI-539      3.127102               Low
6     925338  MDAMB157_BREAST     1012       Vorinostat      4.659746              High
7     906801      BT20_BREAST     1512  Cyclophosphamide      3.328736               Low
8     908121  MDAMB361_BREAST     1373       Dabrafenib      4.311793              High
9     749713   HCC1599_BREAST     1039           SL0101      3.784155              High
10    946382      CAMA1_BREAST     1053           MK-2206      3.586741               Low
11    906851     EFM19_BREAST     1016      Temsirolimus      1.532363               Low
12    905951     BT549_BREAST     1089       Oxaliplatin      4.452836              High
13    905960  MDAMB231_BREAST     1051         Alisertib      3.297214               Low
14   1298157      JIMT1_BREAST     1069          EHT-1864      4.242426              High
15    910927     CAL51_BREAST     1915           AZD3759      2.517798               Low
16    749714   HCC1937_BREAST     1494             SN-38      2.680414               Low
17    749712   HCC1395_BREAST     1931             MIRA-1      4.358006              High
18   1240172  MDAMB436_BREAST     1578        Leflunomide      5.383609              High
19    906801      BT20_BREAST     1003      Camptothecin      0.798846               Low
20    910910    UACC812_BREAST     2048         Vinorelbine     -1.080480               Low
127.0.0.1 - - [17/May/2024 00:29:49] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:29:49] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 00:29:49] "GET /results/user_prediction_result_20240517002949.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:29:49] "GET /results/user_prediction_result_20240517002949.csv HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 00:29:49] "GET /results/user_prediction_result_20240517002949.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:29:49] "GET /results/user_prediction_result_20240517002949.csv HTTP/1.1" 304 -
```

Figure 81: Terminal output of first upload

Figure 82 illustrates the expected message printed and prediction output displayed for the first dataset in the web interface, which is aligned to the expectation.



COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_IC50	Resistance_Cut-Off
1290798	EFM192A_BREAST	1058	Pictilisib	1.719276	Low
909778	UACC893_BREAST	1011	Navitoclax	1.0808445	Low
906844	DU4475_BREAST	1175	Rucaparib	0.69158685	Low
908123	MDAMB468_BREAST	1598	LGK974	4.6478477	High

Figure 82: Web Interface Output of first upload

Figure 83 illustrates the terminal output, showcasing the expected and accurate results for the second dataset. Despite sharing the filename "same_name_dataset.csv," the distinct content of both files is demonstrated by the prediction output containing 631 rows.

```
Received JSON data: {'uploaded_filename': 'same_name_dataset.csv'}
Constructed CSV path: user_file\same_name_dataset.csv
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
     COSMIC_ID   CELL_LINE_NAME  DRUG_ID     DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
1      1290922     HDQP1_BREAST     1004    Vinblastine     -3.285440               Low
2      1290922     HDQP1_BREAST     1005      Cisplatin      2.992935               Low
3      1290922     HDQP1_BREAST     1006     Cytarabine      1.399643               Low
4      1290922     HDQP1_BREAST     1007      Docetaxel     -3.996063               Low
5      1290922     HDQP1_BREAST     1008    Methotrexate     -1.759563               Low
..         ...              ...      ...            ...           ...               ...
628     905960  MDAMB231_BREAST     2169         AZD6482      2.149143               Low
629     905960  MDAMB231_BREAST     2172             JQ1      2.388362               Low
630     905960  MDAMB231_BREAST     2173           PFI-1      3.019476               Low
631     905960  MDAMB231_BREAST     2177         SGC0946      3.397460               Low
632     905960  MDAMB231_BREAST     2359     GSK2830371      3.562152               Low

[631 rows x 6 columns]
127.0.0.1 - - [17/May/2024 00:44:42] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:44:42] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 00:44:42] "GET /results/user_prediction_result_20240517004442.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:44:42] "GET /results/user_prediction_result_20240517004442.csv HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 00:44:42] "GET /results/user_prediction_result_20240517004442.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [17/May/2024 00:44:42] "GET /results/user_prediction_result_20240517004442.csv HTTP/1.1" 304 -
```

Figure 83: Terminal output of second upload

Figure 84 illustrates the expected message printed and prediction output displayed for the second dataset in the web interface, which is aligned to the expectation.

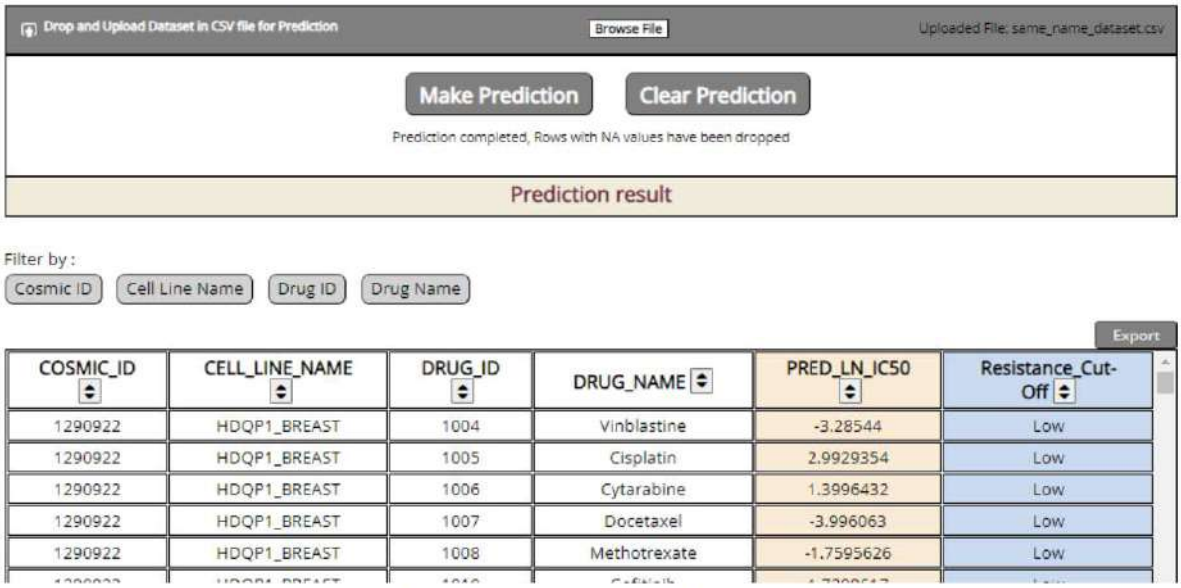


Figure 84: Web Interface Output of second upload

Based on the provided outputs, the Upload Logic Test successfully **PASSES**. The prediction output of the second file distinctly differs from that of the first file, despite sharing the same filename. This outcome serves to validate the upload logic, ensuring that predictions are executed based on the most recently uploaded file.

Section 4.3: Prediction and Output Testing

Section 4.3.1: User Manual Prediction Functionality Testing

(a) Test Description (What is being tested)

The User Manual Prediction Functionality Testing evaluates the functionality of the "Make Prediction" button, which is designed to initiate the prediction process. This test aims to verify whether users can successfully make predictions with a single click of the button. Specifically, it **assesses whether clicking on the button triggers the loading of the predictive model and initiates the prediction action**.

(b) Test Set-Up/ Methodology (How it is being tested)

To conduct this test, we will first launch the Software and navigate to the "User Manual Prediction" section. Next, we will upload a dataset titled "Sample_Dataset.csv" using the application's interface. Upon successful upload, we will proceed to click the "Make Prediction" button. Throughout the test, we will closely monitor the terminal console to verify whether the predictive model is invoked correctly. To enhance the test set-up, we will insert intermediate print statements within the code to indicate the execution of the "Make Prediction" button and the initiation of the Predictive Model Function. These print statements will be detailed in part (c). After the prediction process is completed, we will review the output displayed in the web application interface to ensure that the results generated by invoking the "Make Prediction" button are accurately reflected.

(c) Inputs to the code or part of code being tested

The makePredictButton function shown in **Figure 85** within "web.js" is responsible for capturing the user clicks on the "Make Prediction" button on the web interface, which therefore triggers the model loading action in "app.py".

```
323  makePredictionButton.addEventListener('click', function () {
324      if (!uploadedFileName) {
325          alert('Please upload a file before making a prediction.');
326          return;
327      }
328
329      // Show a loading message
330      predictionStatus.textContent = "LOADING...";
```

Figure 85: Intermediate Printing Command added to makePredictionButton in "web.js"

The command "**Make Prediction button triggered run_model function**" is added to the "make_prediction" function in "app.py" in line 78. It will be printed once the make_prediction function is called upon the trigger of "Make Prediction" button

```
55  def make_prediction():
75
76      # Run the model and handle exceptions gracefully
77      try:
78          print("Make Prediction button triggered run_model function")
79          dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)
```

Figure 86: Intermediate Printing Command added to "make_prediction" function in "app.py"

The intermediate printing commands "**Predictive Model has been loaded**" (line 20), "**Prediction process is now started**" (line 21) and "**Prediction Completed**" (line 119) is added to the "run_model" function as shown in **Figure 87** to show the status of calling of the predictive model to make a prediction.

```
18  def run_model(model, df_upload):
19
20      print("Predictive Model has been loaded")
21      print("Prediction process is now started")
22      user_df = pd.read_csv(df_upload)
23
118     print(prediction_df)
119     print("Prediction Completed")
```

Figure 87: Intermediate Printing Commands added to "run_model" function in "model.py"

By observing the terminal output containing these intermediate printing commands, we can determine whether the predictive model is successfully invoked to make predictions after the "Make Prediction" button is clicked.

(d) Expected Outputs

Upon uploading the dataset and clicking on the "Make Prediction" button, the terminal console should display messages indicating the triggering of the "run_model" function, the successful loading of the predictive model, and the initiation of the prediction process. Subsequently, upon completion of the prediction, the terminal should print the prediction result along with a message confirming the completion of the prediction process. This sequence of messages serves as evidence that the model has been successfully loaded and that the prediction has been successfully executed. Furthermore, the prediction output, presented in both tabular and visual formats, should be accurately displayed in the web interface, confirming the successful completion of the prediction process.

(e) Actual Outputs and Discussions

Figure 88 confirms the successful execution of the "Make Prediction" button functionality as intended. The terminal console displays the expected messages, including "Make Prediction button triggered run_model function", "Predictive Model has been loaded", "Prediction process is now started", and "Prediction Completed". These messages validate the correct invocation of the "run_model" and "make_prediction" functions, demonstrating the successful execution of the prediction action.

```
Received JSON data: {'uploaded_filename': 'Sample_Dataset.csv'}
Constructed CSV path: user_file\Sample_Dataset.csv
Make Prediction button triggered run_model function
Predictive Model has been loaded
Prediction process is now started
2024-05-20 13:31:52.382039: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performanc
tions.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
    COSMIC_ID   CELL_LINE_NAME  DRUG_ID       DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0      909778   UACC893_BREAST     1007       Docetaxel     -3.387078               Low
1      909778   UACC893_BREAST     1010       Gefitinib      2.238444               Low
2      909778   UACC893_BREAST     1030        KU-55933      2.742792               Low
3      909778   UACC893_BREAST     1032        Afatinib      0.728675               Low
4      909778   UACC893_BREAST     1046  Wee1 Inhibitor      2.679244               Low
5      909778   UACC893_BREAST     1060       PD0325901      2.148164               Low
6      909778   UACC893_BREAST     1507     Ruxolitinib      3.135053               Low
7      909778   UACC893_BREAST     1922       Cediranib      2.229212               Low
8      909778   UACC893_BREAST     1932      NVP-ADW742      1.784981               Low
9      905960  MDAMB231_BREAST     1007       Docetaxel     -2.212948               Low
10     905960  MDAMB231_BREAST     1010       Gefitinib      3.665546               Low
11     905960  MDAMB231_BREAST     1030        KU-55933      3.778869              High
12     905960  MDAMB231_BREAST     1032        Afatinib      3.164263               Low
13     905960  MDAMB231_BREAST     1046  Wee1 Inhibitor      3.627371               Low
14     905960  MDAMB231_BREAST     1060       PD0325901      3.495899               Low
15     905960  MDAMB231_BREAST     1507     Ruxolitinib      4.127828              High
16     905960  MDAMB231_BREAST     1922       Cediranib      3.580712               Low
17     905960  MDAMB231_BREAST     1932      NVP-ADW742      3.389481               Low
18     905960  MDAMB231_BREAST     2359      GSK2830371      3.784025              High
Prediction Completed
```

Figure 88: Terminal Console Output for Identification of effect of "Make Prediction" button

As shown in **Figure 89**, the prediction output is successfully produced and displayed in the web interface after the "Make Prediction" button is clicked.

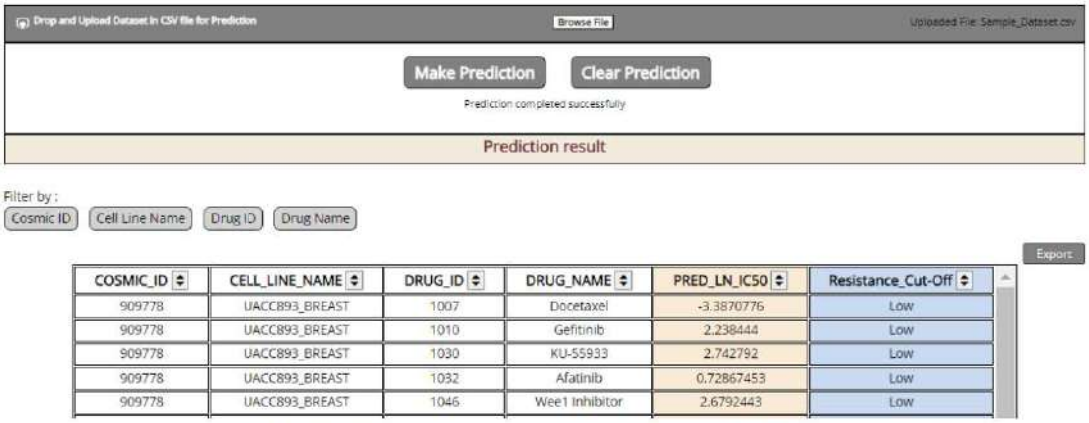


Figure 89: Web Interface Displayed after "Make Prediction" button is clicked

The outputs provided confirm the **PASS** of the User Manual Prediction Functionality Testing, indicating that the "Make Prediction" button effectively initiates the predictive model to perform the prediction action on the user-uploaded data.

Section 4.3.2: Predictive Model Output Data Testing

(a) Test Description (What is being tested)

The Predictive Model Output Data Testing aims to examine the process of generating and storing prediction outputs within our application. When a prediction is made, the application generates a result in the form of a dataframe, which is then converted into a CSV file and stored in a specific folder. This CSV file is subsequently displayed in the result section of the web application and can be exported using the export button. The objective of this test is to **ensure that the predictive model's output data is accurately saved into the CSV file and is accessible** for viewing and export within the application.

(b) Test Set-Up/ Methodology (How it is being tested)

To conduct this test effectively, we will upload a dataset and initiate a prediction process. Following the prediction, we will examine the storage location and format of the output result. By inspecting the application's file structure and directory paths, we will verify where the result is stored and how it is saved. Additionally, we will compare the content of the generated CSV file with the expected output to ensure accuracy. Through this process, we aim to confirm that the predictive model's output data is correctly generated, stored, and accessible within the application.

(c) Inputs to the code or part of code being tested

After executing the "run_model" function, the prediction result is returned in a dataframe, denoted by the variable "prediction_df," as depicted in **Figure 90**.

```
18   def run_model(model, df_upload):
120
121      return dropped, prediction_df
```

Figure 90: Code Snippet of "run_model" function in "model.py" that outputs prediction result as dataframe

Subsequently, in the "app.py" file housing the "make_prediction" function, as illustrated in **Figure 91**, the code is responsible for converting the data frame into a CSV file with a specific name (lines 80-83). This CSV file is then outputted to the folder path representing the directory 'result', identified as the 'RESULT_FOLDER' shown in **Figure 92.**

```
55  def make_prediction():
77      try:
78          dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)
79
80          timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  # Correctly access datetime class
81          result_filename = f'user_prediction_result_{timestamp}.csv'
82          result_path = os.path.join(app.config['RESULT_FOLDER'], result_filename)
83          prediction_df.to_csv(result_path, index=False)
84          if dropped:
85              return jsonify({'message': 'Prediction completed, Rows with NA values have been dropped', 'result_filename': result_filename})
86          else:
87              return jsonify({'message': 'Prediction completed successfully', 'result_filename': result_filename})
88      except Exception as e:
89          print(f"Error during model prediction: {e}")
90          return jsonify({'message': 'Wrong Dimension Data or Non-numeric Values Detected, Please Input Correct Data', 'details': str(e)}), 500
```

Figure 91: Code Snippet of "make_prediction" function in "app.py" for conversion of result dataframe into csv and storage

```
25  RESULT_FOLDER = 'result'
26  app.config['RESULT_FOLDER'] = RESULT_FOLDER
```

Figure 92: RESULT_FOLDER path

(d) Expected Outputs

The anticipated outcome entails verifying that the **number of rows in the input dataset corresponds to the number of rows in the output CSV file** stored within the 'RESULT_FOLDER.' Additionally, the **drug-cell pair content** should **mirror** the prediction output in the terminal console, serving as evidence of the successful storage of the prediction result dataframe into the CSV file.

(e) Actual Outputs and Discussions

Compared to the anticipated outcome, it is evident that the number of rows in the input data matches precisely with the number of rows in the output data. Additionally, the drug-cell pairs in both the input and output are identical, with the CSV file containing the same content as the prediction result showcased in the terminal console. Consequently, the Predictive Model Output Data Testing PASSED.

Figure 93 below shows the input sample dataset "sample_user_upload_data_7.csv", having 21 rows.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_data_7.csv'}
Constructed CSV path: user_file\sample_user_upload_data_7.csv
          DRUG_NAME  DRUG_ID  COSMIC_ID  ...  ELOA3 (162699)  CDR1 (1038)                                                 isosmiles
0         Pictilisib     1058    1290798  ...        0.000000     0.084064  CS(=O)(=O)N1CCN(CC1)CC2=CC3=C(S2)C(=NC(=N3)C4=...
1         Navitoclax     1011     909778  ...        0.000000     0.000000  CC1(CCC(=C(C1)CN2CCN(CC2)C3=CC=C(C=C3)C(=O)NS(...
2          Rucaparib     1175     906844  ...        0.000000     0.000000  CNCC1=CC=C(C=C1)C2=C3CCNC(=O)C4=C3C(=CC(=C4)F)...
3            LGK974     1598     908123  ...        0.000000     0.000000  CC1=CC(=CN=C1C2=CC(=NC=C2)C)CC(=O)NC3=NC=C(C=C...
4            AZD8055     1059     908122  ...        0.000000     0.028569  C[C@H]1COCCN1C2=NC(=NC3=C2C=CC(=N3)C4=CC(=C(C=...
5           WEHI-539     1997     910852  ...        0.000000     0.000000  C1CC2=C(C=C(C=C2)C3=NC(=C(S3)CCCOC4=CC=C(C=C4)...
6          Vorinostat     1012     925338  ...        0.124328     0.275087                  C1=CC=C(C=C1)NC(=O)CCCCCCC(=O)NO
7     Cyclophosphamide     1512     906801  ...        0.014355     0.000000                          C1CNP(=O)(OC1)N(CCCl)CCCl
8          Dabrafenib     1373     908121  ...        0.000000     0.000000  CC(C)(C)C1=NC(=C(S1)C2=NC(=NC=C2)N)C3=C(C(=CC=...
9               SL0101     1039     749713  ...        0.042644     0.475085  C[C@H]1[C@@H]([C@H]([C@H]([C@@H](O1)OC2=C(OC3=...
10           MK-2206     1053     946382  ...        0.000000     0.000000  C1CCC(C1)(C2=CC=C(C=C2)C3=C(C=C4C(=N3)C=CNSC4=N...
11        Temsirolimus     1016     906851  ...        0.000000     0.000000  C[C@@H]1CC[C@H]2C[C@@H](/C(=C/C=C/C=C/C[C@H](C[...
12         Oxaliplatin     1089     905951  ...        0.000000     0.000000                  C1CC[C@H]([C@@H](C1)N)N.C(=O)(C(=O)O)O.[Pt]
13           Alisertib     1051     905960  ...        0.000000     0.097611  COC1=C(C(=CC=C1)F)C2=NCC3=CN=C(N=C3C4=C2C=C(C=...
14            EHT-1864     1069    1298157  ...        0.000000     0.000000  C1COCCN1CC2=CC(=O)C(=CO2)OCCCCCSC3=C4C=CC(=CC4...
15             AZD3759     1915     910927  ...        0.124328     0.000000  C[C@@H]1CN(CCN1C(=O)OC2=C(C=C3C(=C2)C(=NC=N3)N...
16               SN-38     1494     749714  ...        0.000000     0.000000  CCC1=C2CN3C(=CC4=C(C3=O)COC(=O)[C@@]4(CC)O)C2=...
17             MIRA-1     1931     749712  ...        0.084064     0.000000                          CCC(=O)OCN1C(=O)C=CC1=O
18         Leflunomide     1578    1240172  ...        0.056584     0.799087              CC1=C(C=NO1)C(=O)NC2=CC=C(C=C2)C(F)(F)F
19       Camptothecin      1003     906801  ...        0.014355     0.000000  CC[C@@]1(C2=C(COC1=O)C(=O)N3CC4=CC5=CC=CC=C5N=...
20        Vinorelbine      2048     910910  ...        0.000000     0.000000  CCC1=C[C@H]2C[C@@](C3=C(CN(C2)C1)C4=CC=CC=C4N3...

[21 rows x 19226 columns]
```

Figure 93: Input Dataset

In **Figure 94**, the terminal output post-prediction reveals a range of 0 to 21 rows, totaling 21 rows excluding the header, consistent with the input data. The prediction output is directed to the path: "/results/user_prediction_result_20240520145308.csv".

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
    COSMIC_ID   CELL_LINE_NAME  DRUG_ID         DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0     1290798    EFM192A_BREAST     1058        Pictilisib      1.719276                Low
1      909778    UACC893_BREAST     1011        Navitoclax      1.080844                Low
2      906844     DU4475_BREAST     1175         Rucaparib      0.691587                Low
3      908123   MDAMB468_BREAST     1598           LGK974      4.647848               High
4      908122   MDAMB453_BREAST     1059          AZD8055      1.692313                Low
5      910852     CAL851_BREAST     1997          WEHI-539      3.127102                Low
6      925338   MDAMB157_BREAST     1012         Vorinostat      4.659746               High
7      906801      BT20_BREAST     1512    Cyclophosphamide      3.328736                Low
8      908121   MDAMB361_BREAST     1373         Dabrafenib      4.311793               High
9      749713   HCC1599_BREAST     1039              SL0101      3.784155               High
10     946382      CAMA1_BREAST     1053           MK-2206      3.586742                Low
11     906851      EFM19_BREAST     1016        Temsirolimus      1.532362                Low
12     905951     BT549_BREAST     1089         Oxaliplatin      4.452836               High
13     905960   MDAMB231_BREAST     1051           Alisertib      3.297214                Low
14    1298157      JIMT1_BREAST     1069            EHT-1864      4.242426               High
15     910927     CAL51_BREAST     1915             AZD3759      2.517798                Low
16     749714    HCC1937_BREAST     1494               SN-38      2.600414                Low
17     749712    HCC1395_BREAST     1931              MIRA-1      4.358007               High
18    1240172   MDAMB436_BREAST     1578          Leflunomide      5.383609               High
19     906801      BT20_BREAST     1003       Camptothecin       0.798847                Low
20     910910    UACC812_BREAST     2048         Vinorelbine     -1.080480                Low
result\user_prediction_result_20240520145308.csv
127.0.0.1 - - [20/May/2024 14:53:08] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [20/May/2024 14:53:08] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [20/May/2024 14:53:08] "GET /results/user_prediction_result_20240520145308.csv HTTP/1.1" 200 -
```

Figure 94: Terminal Prediction Output

Upon examination of the result folder in **Figure 95**, the presence of the result CSV file with the specified path is evident.

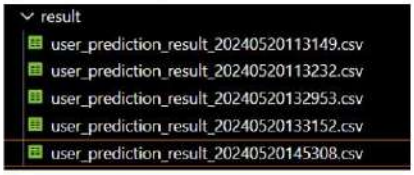


Figure 95: Files in "result" folder

Further analysis of the content within the output CSV file confirms alignment with the prediction result displayed in the terminal console. This affirms the successful conversion of the prediction output dataframe into CSV file format and the effective storage of the result CSV.

```
src > result > user_prediction_result_20240520145308.csv
1   COSMIC_ID,CELL_LINE_NAME,DRUG_ID,DRUG_NAME,PRED_LN_IC50,Resistance_Cut-Off
2   1290798,EFM192A_BREAST,1058,Pictilisib,1.7192758,Low
3   909778,UACC893_BREAST,1011,Navitoclax,1.080844,Low
4   906844,DU4475_BREAST,1175,Rucaparib,0.69158685,Low
5   908123,MDAMB468_BREAST,1598,LGK974,4.6478477,High
6   908122,MDAMB453_BREAST,1059,AZD8055,1.6923134,Low
7   910852,CAL851_BREAST,1997,WEHI-539,3.127102,Low
8   925338,MDAMB157_BREAST,1012,Vorinostat,4.659746,High
9   906801,BT20_BREAST,1512,Cyclophosphamide,3.328736,Low
10  908121,MDAMB361_BREAST,1373,Dabrafenib,4.311793,High
11  749713,HCC1599_BREAST,1039,SL0101,3.7841551,High
12  946382,CAMA1_BREAST,1053,MK-2206,3.5867417,Low
13  906851,EFM19_BREAST,1016,Temsirolimus,1.5323622,Low
14  905951,BT549_BREAST,1089,Oxaliplatin,4.4528356,High
15  905960,MDAMB231_BREAST,1051,Alisertib,3.2972143,Low
16  1298157,JIMT1_BREAST,1069,EHT-1864,4.242426,High
17  910927,CAL51_BREAST,1915,AZD3759,2.5177984,Low
18  749714,HCC1937_BREAST,1494,SN-38,2.6004143,Low
19  749712,HCC1395_BREAST,1931,MIRA-1,4.358007,High
20  1240172,MDAMB436_BREAST,1578,Leflunomide,5.3836093,High
21  906801,BT20_BREAST,1003,Camptothecin,0.7988467,Low
22  910910,UACC812_BREAST,2048,Vinorelbine,-1.0804796,Low
```

Figure 96: Observation of prediction result csv file

Section 4.4: Predictive Model Performance Validation Testing

(a) Test Description (What is being tested)

The objective of this test is to **validate the predictive model's ability to generalise and perform accurately when faced with new, unseen data**. This is critical for ensuring the model's utility in real-world scenarios where users may upload data for prediction that the model has not previously encountered. Specifically, we aim to determine whether the model can make valid predictions for new cell lines or drugs that were not included in the training dataset. This involves assessing the model's robustness, reliability, and accuracy in predicting outcomes based on novel inputs. The overarching goal is to confirm that the model is not simply memorising the training data but is capable of applying learned patterns to generate accurate predictions in diverse, real-world contexts. This test is crucial for establishing confidence in the model's performance and its practical applicability in predicting responses to new drugs or understanding new cell lines.

(b) Test Set-Up/ Methodology (How it is being tested)

The methodology for validating the model's performance involves a structured and systematic approach to data preparation, training, and evaluation. The test begins by splitting the entire dataset, which includes gene expression data, drug information, and chemical compound structures, into two main segments: **90% for training** and **10% for testing**. This initial division ensures that a portion of the data remains unseen by the model during training, allowing for a genuine evaluation of its performance on new data. The **90% training data** is **further subdivided** into **85% for actual training** (X_train) and **15% for validation** (X_val). This secondary split is crucial for fine-tuning the model and preventing overfitting. The model is first trained on the X_train subset, where it learns to identify patterns and relationships within the data. During this phase, metrics such as loss are monitored to optimise the model's parameters. The validation data (X_val) is then used to test the model's performance during training. This helps in adjusting hyperparameters and ensuring that the model is not overfitting the training data. Validation provides an intermediary check on the model's generalizability before it is tested on the completely unseen test data. Once the model has been trained and validated, it is tested on the remaining 10% of the data (X_test). This test set represents new, unseen data that simulates real-world conditions. The model's predictions on this data are compared against the actual outcomes to evaluate its accuracy and reliability. Various performance metrics, including Root Mean Squared Error (RMSE), Pearson Correlation and R-squared (R^2), are calculated to quantify the model's predictive accuracy. **Figure 97** shows the set-up of the splitting of training, validation and testing data.

In [5]:
```
# Split data into features (X) and target variable (y)
X = df.drop(columns=['DRUG_NAME', 'CCLE_Name', 'DRUG_ID', 'COSMIC_ID', 'LN_IC50'])
y = df['LN_IC50']
```

In [6]:
```
# Step 1: Split into 90% training and 10% testing
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

# Step 2: Split the 90% training data into 85% training and 15% validation
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.15, random_state=42)
```

In [8]:
```
# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Figure 97: Splitting of X and y into training, validation and testing data

(c) Inputs to the code or part of code being tested

The code provided in **Figure 98** focuses on training a predictive model and evaluating its performance on a test set. The first part of the code, model.fit(), trains the model using the scaled training dataset (X_train_scaled) and the corresponding target values (y_train). The training process involves iterating over the dataset for 100 epochs with a batch size of 32. The training is monitored using the validation dataset (X_val and y_val), which helps in tuning the model and preventing overfitting through early stopping. Early stopping is facilitated by a callback function that halts the training if the model's performance on the validation data stops improving. The verbosity level is set to 1 to provide detailed logs of the training process. After training, the model's performance is evaluated on the test set using the model.predict() function, which takes X_test_scaled as input. X_test_scaled is the preprocessed and scaled test dataset that contains new, unseen data. The function outputs y_pred, which are the predicted values generated by the model. To assess the model's accuracy, the predicted values (y_pred) are compared to the actual target values (y_test) using the

Root Mean Squared Error (RMSE) metric. The RMSE provides a quantitative measure of the model's prediction accuracy, with lower values indicating better performance. **Figure 99** also shows the code for the performance evaluation of the model using different metrics, such as R-squared (R^2), Mean Absolute Error (MAE) and Pearson Correlation Coefficient.

```
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val), callbacks=[early_st
164/164 [==============================] - 2s 15ms/step - loss: 0.9228 - val_loss: 50.5486
Epoch 34/100
164/164 [==============================] - 2s 15ms/step - loss: 0.9884 - val_loss: 63.6155
Epoch 35/100
164/164 [==============================] - 3s 15ms/step - loss: 0.9561 - val_loss: 33.0401
Epoch 36/100
164/164 [==============================] - 2s 14ms/step - loss: 0.9673 - val_loss: 19.9122
Epoch 37/100
164/164 [==============================] - 2s 15ms/step - loss: 0.9222 - val_loss: 15.9544
Epoch 38/100
164/164 [==============================] - 2s 14ms/step - loss: 0.9368 - val_loss: 50.9558
Epoch 39/100
164/164 [==============================] - 3s 16ms/step - loss: 0.9415 - val_loss: 30.9330
Epoch 40/100
164/164 [==============================] - 3s 17ms/step - loss: 0.8993 - val_loss: 24.9960
Epoch 41/100
164/164 [==============================] - 2s 15ms/step - loss: 0.9442 - val_loss: 59.7728
Epoch 42/100
164/164 [==============================] - 2s 14ms/step - loss: 0.9126 - val_loss: 50.5569
Epoch 43/100
```

```
# Predict on the test set
y_pred = model.predict(X_test_scaled)
```

Figure 98: Process of fitting training and validation dataset to model, and make prediction using testing dataset

```
import numpy as np
from scipy.stats import pearsonr
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Flatten arrays
y_test_flat = y_test_np.flatten()
y_pred_flat = y_pred_np.flatten()

# Evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test_flat, y_pred_flat))
mae = mean_absolute_error(y_test_flat, y_pred_flat)

try:
    pearson_corr, _ = pearsonr(y_test_flat, y_pred_flat)
except ValueError as e:
    print("Warning:", e)
    pearson_corr = np.nan

r2 = r2_score(y_test_flat, y_pred_flat)

print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("Pearson Correlation:", pearson_corr)
print("R-squared (R2):", r2)
```

Figure 99: Metrics Evaluation Codes on Model's Performance

(d) Expected Outputs

The ideal expected output for the model performance validation is that the **RMSE should be below 1**. This indicates that, on average, the difference between the predicted values and the actual values is less than 1 unit, reflecting high accuracy in the model's predictions. Achieving an RMSE below 1 would demonstrate that the model has successfully learned the underlying patterns in the data and can generalise well to new, unseen data, thereby ensuring reliable and precise predictions in real-world scenarios. The ideal **Pearson correlation coefficient value would be close to 1**. A high positive correlation (close to 1) indicates a strong linear relationship between the predicted and actual values, suggesting that the model's predictions closely follow the true values. The ideal **R^2 value would be as close to 1** as possible because an R^2 value of 1 indicates that the model explains all of the variability of the response data around its mean. The ideal **MAE value would be as low as possible**, ideally approaching zero. A lower MAE indicates that the model's predictions are closer to the actual values on average, reflecting better accuracy and precision in the predictions.

(e) Actual Outputs and Discussions

Figure 100 shows the comparisons between the actual y_test value (y_test_np) and the predicted y_test (y_pred_np). It is shown that overall, the predicted value is somehow consistent with the actual value (for example, the last three actual y_test are 2.25, 4.43 and 5.49, whereas the last three predicted y_test are 2.13, 2.68 and 4.67). The LN_IC50 value predicted does not show a great variant from the actual value.

```
In [42]: 1 y_test_np

Out[42]: array([[3.163133],
                [1.343452],
                [5.623427],
                ...,
                [2.248526],
                [4.42983 ],
                [5.494281]])

In [43]: 1 y_pred_np

Out[43]: array([5.085401 , 1.5110184, 4.928393 , ..., 2.127267 , 2.678237 ,
                4.4697785], dtype=float32)
```

Figure 100: Comparisons between actual y_test value and predicted y_test

Figure 101 shows the metrics evaluation output. It is shown that the **RMSE** value is approximately **0.937,** indicating that, on average, the model's predictions deviate from the actual values by approximately 0.937 units. It **PASSED the expected output of having RMSE < 1**. For MAE, the **MAE** value is approximately **0.702**, which represents the average absolute difference between the model's predictions and the actual values, which does not align with the expectation of the lower it is, the better. The **Pearson correlation coefficient** is approximately **0.929**, indicating a strong positive linear relationship between the predicted and actual values, therefore **PASSED the expected output**. The **R-squared** value is approximately **0.862**, which indicates that approximately 86.2% of the variability in the dependent variable (target) is explained by the independent variables (features) included in the model., therefore **PASSED the expected output**.

```
In [44]:
import numpy as np
from scipy.stats import pearsonr
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Flatten arrays
y_test_flat = y_test_np.flatten()
y_pred_flat = y_pred_np.flatten()

# Evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test_flat, y_pred_flat))
mae = mean_absolute_error(y_test_flat, y_pred_flat)

try:
    pearson_corr, _ = pearsonr(y_test_flat, y_pred_flat)
except ValueError as e:
    print("Warning:", e)
    pearson_corr = np.nan

r2 = r2_score(y_test_flat, y_pred_flat)

print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("Pearson Correlation:", pearson_corr)
print("R-squared (R2):", r2)

Root Mean Squared Error (RMSE): 0.9371232128647476
Mean Absolute Error (MAE): 0.7018996115121721
Pearson Correlation: 0.9290970085365322
R-squared (R2): 0.8616670940359872
```

Figure 101: Results of Model's Performance Evaluation

Overall, the evaluation metrics indicate that the predictive model performs well, with relatively low RMSE, high Pearson correlation coefficient, and a high R-squared value. These results suggest that the model's predictions closely align with the actual values and demonstrate good predictive accuracy and performance. However, the MAE is not approaching 0, showing the potential of our model to be further improved to further reduce the RMSE. Overall, this testing has **PASSED**, showing that the **prediction made by the model is VALID.**

Section 5: Integration Testing

Section 5.1: Integration of Predictive Model into Web Application Testing

(a) Test Description (What is being tested)

The Integration of Predictive Model into Web Application Testing assesses the seamless integration of a predictive model, coded in Python, into the backend of a web application. This integration involves connecting the predictive model's functionality to the frontend user interface, which is typically designed using HTML for structure and CSS for styling. The interaction and functionality of buttons to trigger the prediction action within the web application are managed using JavaScript. To bridge the gap between the frontend and the predictive model, Flask, a Python web framework, is employed, providing necessary tools and features for easy web application development. The objective of this testing is to evaluate how effectively the predictive model is loaded and called from the backend, and executed within the web application environment.

(b) Test Set-Up/ Methodology (How it is being tested)

The test initiates when the user triggers the "Make Prediction" button within the web application interface. Upon activation, the predictive model code is executed, generating a print message upon completion, and displaying the resulting dataset output. The evaluation involves a physical examination of both Flask and JavaScript code to comprehend the flow of file serving and execution calls. By uploading a dataset and triggering the "Make Prediction" button, the test scrutinises the sequence of actions, and observing the calls made to the predictive model.

(c) Inputs to the code or part of code being tested

Previously, we have saved the trained model under the name of "**LATEST_PREDICTION.h5**" as shown in **Figure 102**. This model is saved under a file called "uploads" and the "run_model" function takes the model represented by "MODEL_FOLDER" storing the path of the model as the first argument.

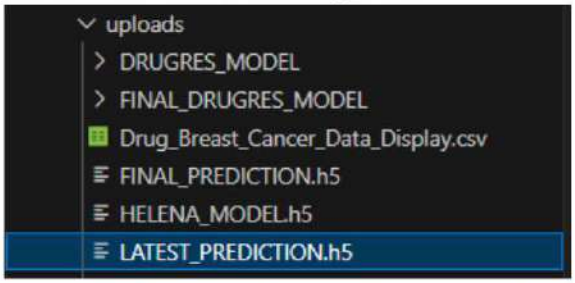


Figure 102: Folder uploads that contains the model saved LATEST_PREDICTION.h5

```
21
22  MODEL_FOLDER = 'uploads/LATEST_PREDICTION.h5'
23  app.config['MODEL_FOLDER'] = MODEL_FOLDER
24
```

Figure 103: Code segment shows model path

In the backend Python code, specifically in the "model.py" file located within the "model" folder, the function "run_model" is imported and invoked in "app.py" to execute the predictive model as shown in **Figure 104**.

```
1  from flask import Flask, session, render_template, send_from_directory, jsonify,
2  from flask_session import Session
3  from werkzeug.utils import secure_filename
4  import csv
5  import pandas as pd
6  from model.model import run_model
7  import os
8  from datetime import datetime, timedelta
9  import datetime
10 import shutil
```

Figure 104: Code segment of "app.py" to show function arguments to show how we call the model

Flask application is used to create /predict routes with a POST method for handling prediction requests as shown in **Figure 105**. It gets the filename from the JSON request body and then it makes up a path to the uploaded CSV file. If the file is not found or invalid, it returns a 400 error; if the file does not exist, it returns a 404 error. The **run_model**

function is finally called in **Line 78** with the by passing in the saved model path (LATEST_PREDICTION.h5) and input data CSV path as input parameters to the run_model function, triggering the action of executing the predictive model). Any errors during model execution will be communicated to the user through a 500 error. This is the procedure we followed to load the saved model and execute it for making predictions. Once the model is loaded, the retrieval of prediction output of the model and integration of the output into the web application will be demonstrated below in Section 5.3: Integration of Predictive Model into Web Application Testing.

```
54  @app.route('/predict', methods=['POST'])
55  def make_prediction():
56      data = request.json
57      print("Received JSON data:", data)  # Debugging information
58
59      # Extract the filename and ensure it's a valid string
60      uploaded_filename = data.get('uploaded_filename', '')
61      if not uploaded_filename or not isinstance(uploaded_filename, str):
62          print("Error: Invalid or missing filename.")
63          return jsonify({'message': 'Invalid or missing filename'}), 400
64
65      filename = session['uploaded_filename']
66
67      # Construct the path for the uploaded file
68      csv_path = os.path.join(app.config['USER_FOLDER'], filename)
69      print("Constructed CSV path:", csv_path)  # Debugging information
70
71      # Check if the file exists
72      if not os.path.exists(csv_path):
73          print("Error: File not found.")
74          return jsonify({'message': 'File not found'}), 404
75
76      # Run the model and handle exceptions gracefully
77      try:
78          dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)
```

Figure 105: Code Segment of "make_prediction" function in "app.py" to call the predictive model

(d) Expected Outputs

Upon uploading a dataset and triggering the "Make Prediction" button, the predictive model code should execute flawlessly, producing correct output displayed in the terminal output, proving the successful loading and calling of the saved model.

(e) Actual Outputs and Discussions

Successful prediction has been made, a message showing "Compiled the loaded model" is displayed in the terminal output, showing successful loading and calling of the saved mode "LATEST_PREDICTION.h5"

```
127.0.0.1 - - [17/May/2024 14:21:09] "GET /static/images/upload-logo.png HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 14:21:53] "POST /upload HTTP/1.1" 200 -
Received JSON data: {'uploaded_filename': 'sample_user_upload_data_7.csv'}
Constructed CSV path: user_file\sample_user_upload_data_7.csv
2024-05-17 14:21:54.361500: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
   COSMIC_ID   CELL_LINE_NAME  DRUG_ID        DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0    1290798   EFM192A_BREAST     1058       Pictilisib      1.719276               Low
1     909778   UACC893_BREAST     1011       Navitoclax      1.080845               Low
2     906844    DU4475_BREAST     1175        Rucaparib      0.691587               Low
3     908123  MDAMB468_BREAST     1598            LGK974      4.647848              High
4     908122  MDAMB453_BREAST     1059           AZD8055      1.692313               Low
5     910852    CAL851_BREAST     1997          WEHI-539      3.127102               Low
6     925338  MDAMB157_BREAST     1012       Vorinostat      4.659746              High
7     906801      BT20_BREAST     1512  Cyclophosphamide      3.328736               Low
8     908121  MDAMB361_BREAST     1373       Dabrafenib      4.311793              High
9     749713   HCC1599_BREAST     1039           SL0101      3.784155              High
```

Figure 106: Terminal Output showing that model has been taking the file and run in backend

Based on the output, the Integration of Predictive Model into Web Application Testing **PASSES**.

Section 5.2: User Upload Data Preprocessing Integration Testing

(a) Test Description (What is being tested)

Given that the input **user data** contains **19,226 columns** compared to the **19,481 columns (excluding the one column y-feature, LN_IC50)** in the **training dataset** of the predictive model shown in **Figure 107**, which includes an additional 256 columns representing isosmiles converted to morgan fingerprints of the chemical compound structure of drugs, preprocessing steps are therefore necessary for the user uploaded dataset before passing it into the predictive model. Additionally, since the X-features of the training dataset have been scaled and normalised, it is imperative to perform scaling on the user uploaded dataset to ensure consistency in the range of values for each column, aligning it with X_train and X_valid. This scaling is essential for the user uploaded dataset, equivalent to scaling the X_test value, as it will be utilised as such in the predictive model for making predictions. Therefore, conducting preprocessing on the user upload dataset is crucial to facilitate successful prediction. The testing will focus on ensuring that the preprocessing steps, including feature scaling and normalisation, are effectively applied to the user uploaded dataset to prepare it for prediction. Additionally, it will verify that the processed dataset maintains compatibility with the predictive model's input requirements and does not introduce any inconsistencies or errors that could affect prediction accuracy.

```
In [2]: 1 df = pd.read_csv("Drug_Breast_Cancer_Data.csv")
```

```
In [3]: 1 df
```

Out[3]:

	DRUG_NAME	DRUG_ID	COSMIC_ID	CCLE_Name	TSPAN6 (7105)	TNMD (64102)	DPM1 (8813)	SC' (571
0	Camptothecin	1003	909907	ZR7530_BREAST	3.472488	0.0	5.959306	3.878
1	Vinblastine	1004	909907	ZR7530_BREAST	3.472488	0.0	5.959306	3.878
2	Cisplatin	1005	909907	ZR7530_BREAST	3.472488	0.0	5.959306	3.878
3	Cytarabine	1006	909907	ZR7530_BREAST	3.472488	0.0	5.959306	3.878
4	Docetaxel	1007	909907	ZR7530_BREAST	3.472488	0.0	5.959306	3.878
...	...	...	...	...	...	...	...	...
6842	AZD6482	2169	908121	MDAMB361_BREAST	1.855990	0.0	6.736740	2.885
6843	JQ1	2172	908121	MDAMB361_BREAST	1.855990	0.0	6.736740	2.885
6844	PFI-1	2173	908121	MDAMB361_BREAST	1.855990	0.0	6.736740	2.885
6845	SGC0946	2177	908121	MDAMB361_BREAST	1.855990	0.0	6.736740	2.885
6846	GSK2830371	2359	908121	MDAMB361_BREAST	1.855990	0.0	6.736740	2.885

6847 rows × 19482 columns

Figure 107: Dimensions of Training Data "Drug_Breast_Cancer_Data.csv"

(b) Test Set-Up/ Methodology (How it is being tested)

It is understood that when the user clicks on the "Make Prediction" button, it triggers the execution of the "run_model" function, which takes the saved predictive model and the user upload data CSV file as input parameters. To ensure that the code includes the necessary preprocessing steps for the user uploaded dataset, two approaches will be employed for testing: **physical code observation methodology** and **intermediate output printing methodology**. Upon receiving the input user uploaded data, the dimensions of the data before preprocessing will be printed. Subsequently, after the conversion of isosmiles into 256-bit morgan fingerprint binary representations and dropping the "isosmiles" column replaced by the new 256 columns, additional commands will be added to print the dimensions of the dataset again. For testing whether normalisation or scaling is performed on the input user data, the initial X_test value (represented by the user upload dataset after dropping the columns "DRUG_ID", "DRUG_NAME", "COSMIC_ID", "CELL_LINE_NAME") will be printed prior to applying the scaler function. The code segment performing the StandardScaler() action on the X_test value will be observed, and a printing command will be added after the X_test is scaled to compare the initial X_test value with the scaled X_test value, thus verifying if the scaling action has been successfully executed.

Below are the steps required to perform the user upload data preprocessing integration testing:

Step 1: Conduct physical observations on the "run_model" function, identify if there is any code which performs preprocessing actions on the input user upload dataset.

Step 2: Add on intermediate printing commands to print the dimension of user upload dataset (before and after preprocessing). Next, add on intermediate printing commands to print the X_test value (before and after scaling)

Step 3: In the terminal of Visual Studio Code, run the command: python app.py shown in **Figure 108**.

```
PS C:\Users\Acer\Desktop\monash_data_science\2024_sem1\FIT3164\DRUGRES\MDS6_FIT3164\Predictive Model Web Application\src> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
```

Figure 108: Terminal Launching of Software Application

Step 4: Select "User Manual Prediction", upload the file "sample_user_upload_usability_test1.csv", click on "Make Prediction"

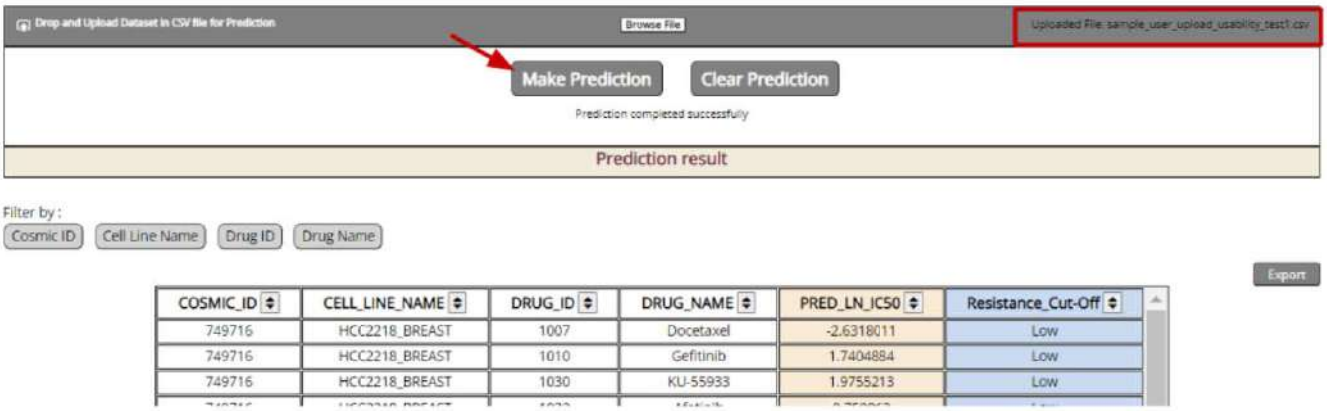


Figure 109: Action to Make Prediction in Web Application

Step 5: Observe the printings output in the terminal

(c) Inputs to the code or part of code being tested

A sample testing file, "sample_user_upload_usability_test1.csv" is uploaded to the web software via "Browse File" and "Make Prediction " button will be clicked to make a prediction. Upon executing the "Make Prediction" action, the function "**run_model**" in the file "**model.py**" shown in **Figure 110** below will be executed, having "sample_user_upload_usability_test1.csv" as the df_upload parameter, and the saved predictive model as the model parameter.

The commands used (**intermediate output printing** methodology) for the user upload data preprocessing integration testing are specified below:

Line 28: print(user_df.shape)
Line 61: print(user_df_morgan)
Line 62: print(len(user_df_morgan.columns))
Line 63: print(user_df_morgan.shape)
Line 78: print(X_test)
Line 79: print(X_test_scaled)

Upon **physical observation**, it is evident that **lines 30 to 59** shown in **Figure 110** execute the preprocessing steps, which involve generating morgan fingerprints of the drugs' chemical compound structures and converting them into 256-bit representations. Additionally, these lines drop the "isosmiles" column and append the generated binary morgan fingerprints to the original user uploaded dataframe. Furthermore, **lines 75 to 76** demonstrate the application of StandardScaler() on x_test_scaled, indicating that the normalisation of the preprocessed user upload data (X_test) is effectively carried out.

```
18  def run_model(model, df_upload):
19
20      user_df = pd.read_csv(df_upload)
21
22      # Remove rows with any missing values
23      # user_df.dropna(inplace=True)
24
25      #Remove unneccesary cols
26      # to_drop = ['Unnamed: 0']
27      # user_df.drop(to_drop, inplace=True, axis=1)
28      print(user_df.shape)
```

```
    arr = []

    # Generate Morgan fingerprints for each compound and store them in arr list
    for i in range(len(user_df)):

        # Define the SMILES string for drug
        smiles = user_df['isosmiles'][i]

        # Generate Rockit molecule object from SMILES
        mol = Chem.MolFromSmiles(smiles)

        # Generate Morgan fingerprint with radius 2 and 256 bits
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius=2, nBits =256)

        # Convert fingerprint to a numpy array
        fp_array = np.zeros((1,), dtype = np.int64)

        # Display the hashed count Morgan fingerprint
        AllChem.DataStructs.ConvertToNumpyArray(fp, fp_array)
        arr.append(fp_array)

morgan_data =pd.DataFrame(arr)


# Join morgan_data DataFrame with user_df
user_df_morgan = user_df.join(morgan_data)

# Drop column 'isosmiles' from user_df_morgan
to_drop = ['isosmiles']
user_df_morgan.drop(to_drop, inplace=True, axis=1)
### for User Upload Data Preprocessing Integration Testing
print(user_df_morgan)
print(len(user_df_morgan.columns))
print(user_df_morgan.shape)

# Define or load a sample structure for X_train columns. This can be a hardcoded list of expec
# For example:
expected_columns = ['DRUG_NAME', 'CELL_LINE_NAME', 'DRUG_ID', 'COSMIC_ID']

# from user upload data
X_test = user_df_morgan.drop(columns=['DRUG_NAME', 'CELL_LINE_NAME', 'DRUG_ID', 'COSMIC_ID'])

X_test.columns = X_test.columns.astype(str)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_test_scaled = scaler.fit_transform(X_test)

print(X_test)
print(X_test_scaled)
```

Figure 110: Part of Code Being Tested for User Upload Data Preprocessing Integration Testing

(d) Expected Outputs

Assuming the validity of my input user upload data "sample_user_upload_usability_test1.csv", it should contain 19226 columns. Given that the output prediction yields 20 rows, the dimension of the input data should consequently be 20 rows by 19226 columns. Post-preprocessing, with the addition of 256 columns and the removal of one column (isosmiles), the expected total should be (19226 + 256 - 1 = 19481) columns. The anticipated outcome upon executing the commands is as follows:

Command: print(user_df.shape)

>> Expected Output: **(20, 19226)**

Command: print(user_df_morgan)

>> Expected Output: **The dataset last few columns are binary representation '0/1'**

Command: print(len(user_df_morgan.columns))

>> Expected Output: **19481**

Command: print(user_df_morgan.shape)

>> Expected Output: **(20, 19481)**

Command: print(X_test)

>> Expected Output: **Initial user upload data (without DRUG_ID, DRUG_NAME, COSMIC_ID, CELL_LINE_NAME, should be starting from a column of gene expressions)**

Command: print(X_test_scaled)

>> Expected Output: **scaled data of X_test, represented in an array of matrix**

(e) Actual Outputs and Discussions

Upon examination of the actual outputs, depicted in **Figure 111 to Figure 114**, it becomes evident that they closely correspond with the anticipated results outlined in part (d). The actual outputs align closely with the expected outputs outlined in part (d). The shape of the user upload data ("sample_user_upload_usability_test1.csv") matches the expected dimensions of (20, 19226), indicating the correct number of rows and columns. Furthermore, the dataset containing morgan fingerprints displays the expected number of columns, totaling 19481 after preprocessing. This confirms the successful addition of 256 columns and the removal of the "isosmiles" column.This alignment serves as compelling evidence that the preprocessing steps have been executed effectively on the input user upload dataset. The congruence between the expected and actual dimensions of the dataset, along with the correct representation of morgan fingerprints, affirms the successful addition and removal of columns as part of the preprocessing procedure.

Furthermore, the successful output of the prediction results further reinforces the efficacy of the preprocessing steps. The fact that valid values of LN_IC50 are produced indicates that the preprocessed dataset has been seamlessly integrated into the predictive model as X_test. This integration is crucial for the accurate execution of the model's predictions. Without this preprocessing, the model's ability to generate valid predictions would have been compromised, potentially rendering the prediction execution prohibited.

Therefore, we can conclude that all the "User Upload Data Preprocessing Integration Testing" has **PASSED**.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_usability_test1.csv'}
Constructed CSV path: user_file\sample_user_upload_usability_test1.csv
(20, 19226)
        DRUG_NAME  DRUG_ID  COSMIC_ID CELL_LINE_NAME  TSPAN6 (7105)  TNMD (64102)  DPM1 (8813)  SCYL3 (57147)  ...  248  249  250  251  252  253  254  255
0       Docetaxel     1007     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    1    1    1    1    0    0    0    0
1       Gefitinib     1010     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    1    0    0    0    0    0    1
2        KU-55933     1030     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    1    0    0    0    1    0    0    0
3        Afatinib     1032     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    0    0    1    0    0    0    0
4   Wee1 Inhibitor     1046     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    0    0    0    0    0    1    0
5       PD0325901     1060     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    1    0    0    0    0    0    0
6     Ruxolitinib     1507     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    0    0    1    0    0    0    1
7       Cediranib     1922     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    1    0    0    0    0    0    1
8      NVP-ADW742     1932     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    0    0    1    1    0    0    0    1
9      GSK2830371     2359     909907  ZR7530_BREAST       3.472488           0.0     5.959306       3.878725  ...    1    0    0    1    0    0    0    1
10      Docetaxel     1007     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    1    1    1    1    0    0    0    0
11      Gefitinib     1010     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    1    0    0    0    0    0    1
12       KU-55933     1030     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    1    0    0    0    1    0    0    0
13       Afatinib     1032     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    0    0    1    0    0    0    0
14  Wee1 Inhibitor     1046     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    0    0    0    0    0    1    0
15      PD0325901     1060     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    1    0    0    0    0    0    0
16    Ruxolitinib     1507     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    0    0    1    0    0    0    1
17      Cediranib     1922     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    1    0    0    0    0    0    1
18     NVP-ADW742     1932     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    0    0    1    1    0    0    0    1
19     GSK2830371     2359     905945    T47D_BREAST       3.310340           0.0     6.844737       2.790772  ...    1    0    0    1    0    0    0    1

[20 rows x 19481 columns]
19481
(20, 19481)
```

Figure 111: Actual Output of the Commands of Line 28, 61, 63: Dimension Before and After Preprocessing

	TSPAN6 (7105)	TNMD (64102)	DPM1 (8813)	SCYL3 (57147)	C1orf112 (55732)	FGR (2268)	CFH (3075)	...	249	250	251	252	253	254	255
0	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	1	1	1	0	0	0	0
1	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	1	0	0	0	0	0	1
2	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	0	0	1	0	0	0
3	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	0	1	0	0	0	0
4	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	0	0	0	0	1	0
5	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	1	0	0	0	0	0	0
6	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	0	1	0	0	0	1
7	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	1	0	0	0	0	0	1
8	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	1	1	0	0	0	1
9	3.472488	0.0	5.959306	3.878725	3.646163	0.000000	0.042644	...	0	0	1	0	0	0	1
10	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	1	1	1	0	0	0	0
11	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	1	0	0	0	0	0	1
12	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	0	0	1	0	0	0
13	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	0	1	0	0	0	0
14	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	0	0	0	0	1	0
15	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	1	0	0	0	0	0	0
16	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	0	1	0	0	0	1
17	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	1	0	0	0	0	0	1
18	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	1	1	0	0	0	1
19	3.310340	0.0	6.844737	2.790772	4.135042	0.028569	0.176323	...	0	0	1	0	0	0	1

[20 rows x 19477 columns]

Figure 112: Actual Output of the Commands of Line 78: X_test

```
[[ 1.         0.        -1.        ...  0.        -0.33333333
  -1.        ]
 [ 1.         0.        -1.        ...  0.        -0.33333333
   1.        ]
 [ 1.         0.        -1.        ...  0.        -0.33333333
  -1.        ]
 ...
 [-1.         0.         1.        ...  0.        -0.33333333
   1.        ]
 [-1.         0.         1.        ...  0.        -0.33333333
   1.        ]
 [-1.         0.         1.        ...  0.        -0.33333333
   1.        ]]
```

Figure 113: Actual Output of the Commands of Line 79: X_test_scaled

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step
```

	COSMIC_ID	CELL_LINE_NAME	...	PRED_LN_IC50	Resistance_Cut-Off
0	749716	HCC2218_BREAST	...	-2.631801	Low
1	749716	HCC2218_BREAST	...	1.740488	Low
2	749716	HCC2218_BREAST	...	1.975521	Low
3	749716	HCC2218_BREAST	...	0.759863	Low
4	749716	HCC2218_BREAST	...	1.911579	Low
5	749716	HCC2218_BREAST	...	1.806497	Low
6	749716	HCC2218_BREAST	...	2.303889	Low
7	749716	HCC2218_BREAST	...	1.690433	Low
8	749716	HCC2218_BREAST	...	1.522244	Low
9	908123	MDAMB468_BREAST	...	-3.145956	Low
10	908123	MDAMB468_BREAST	...	3.513914	Low
11	908123	MDAMB468_BREAST	...	4.015914	High
12	908123	MDAMB468_BREAST	...	2.305163	Low
13	908123	MDAMB468_BREAST	...	3.859415	High
14	908123	MDAMB468_BREAST	...	3.376752	Low
15	908123	MDAMB468_BREAST	...	4.411232	High
16	908123	MDAMB468_BREAST	...	3.451726	Low
17	908123	MDAMB468_BREAST	...	3.201281	Low

Figure 114: Actual Output of the Commands of Line 78

Section 5.3: Integration of Prediction Output into Web Application Testing

(a) Test Description (What is being tested)

This test focuses on verifying the seamless integration of prediction output into our web application interface. Specifically, we aim to ensure the accurate display of prediction results in tabular format within the web application. Our objective is to **examine how the software retrieves prediction output from the model** and utilises it to generate the result table component. This entails validating that the correct path of the result outputted by the "Make Prediction" action is passed as a parameter into the code responsible for result table csv output. In this case, the <table> element is employed for displaying data in tabular format (will be explained below). While the steps to read data from the CSV file and display it as table output may be correct, there is a possibility of encountering integration issues within the web application. Even if the terminal console displays the prediction result successfully, it does not guarantee seamless integration into our web application, as potential issues may arise from incorrect path passing or improper calling actions of the table. Therefore, this testing aims to scrutinise the integration process and **ensure that the prediction results are seamlessly displayed in the table format within the web application interface**. We will explore different paths for testing, **validating each to ensure that the correct table is displayed in the appropriate context**.

(b) Test Set-Up/ Methodology (How it is being tested)

For this testing phase, we will execute two distinct tests to ensure the seamless integration of table output into our web application interface.

Test 1: Correct table Input Path Reading

Refer to Section 5.4: Integration of Visualisation Output into Web Application Testing for details.

Test 2: Integration of Table Display into Web Application

In this test, we will evaluate the integration of the table display into our web application using the "**createTable**" function in "**web.js**". Initially, we will upload the "sample_user_upload_1.csv" file to the web application. Subsequently, we will initiate the "Make Prediction" action to generate predictions. We will closely monitor whether the table is successfully displayed after the message "Prediction completed successfully" is shown. Furthermore, we will observe whether the "createTable" function accurately reads the data from the CSV file containing the predicted results outputted by the predictive model. Our scrutiny will extend to ensuring that the data presented in the table corresponds to the predicted results generated by our model [displayed_table = prediction result csv]

(c) Inputs to the code or part of code being tested

To guarantee that the predicted result is effectively transmitted to the createTable function, it is imperative to ensure that the invocation of the "createTable" function occurs subsequent to the successful generation of prediction results. Upon physical inspection, it becomes apparent that **lines 334 and 347** of the "make_prediction" function shown in **Figure 115** are responsible for reading the file uploaded. The invocation of the createTable function is contingent upon two conditions: first, the status of the response must be 200, indicating the successful execution of the prediction action, and second, the data.result_filename must not be null, signifying the successful conversion of prediction results into the CSV file format. Only under these circumstances will the createTable function be called, ensuring the seamless integration of predicted results into the web application interface.

```
324  makePredictionButton.addEventListener('click', function () {
325      if (!uploadedFileName) {
326          alert('Please upload a file before making a prediction.');
327          return;
328      }
329
330      // Show a loading message
331      predictionStatus.textContent = "LOADING...";
332
333      // Send the uploaded filename in JSON format
334      fetch('/predict', {
335          method: 'POST',
336          headers: { 'Content-Type': 'application/json' },
337          body: JSON.stringify({ 'uploaded_filename': uploadedFileName })
338      })
339      .then(response => response.json().then(data => ({ status: response.status, data })))
340      .then(({ status, data }) => {
341          if (status !== 200) {
342              throw new Error(data.message || "Prediction request failed");
343          }
344
```

```
        predictionStatus.textContent = data.message;
        if (data.result_filename) {
            createTable(`/results/${data.result_filename}`);
        } else {
            predictionStatus.textContent = "Error: No result file returned.";
        }
        predictionMade = true;
    })
    .catch(error => {
        predictionStatus.textContent = 'Error during prediction: ' + error.message;
    });
});
```

Figure 115: Code Segment of function "make_prediction" in "app.py" for reading input user file

After successfully reading the uploaded file, the system will invoke the "run_model" function in "model.py" and provide the path used to store the uploaded dataset as input for prediction to the model as shown in **Figure 116**.

```
def make_prediction():
    try:
        dropped, prediction_df = run_model(MODEL_FOLDER, csv_path)

        # if prediction_df is None:
        #     return jsonify({'error': prediction_df}), 400  # Return a bad request with the error message

        # prediction_df = run_model(MODEL_FOLDER, csv_path)
        timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  # Correctly access datetime class
        result_filename = f'user_prediction_result_{timestamp}.csv'
        result_path = os.path.join(app.config['RESULT_FOLDER'], result_filename)
        prediction_df.to_csv(result_path, index=False)
```

Figure 116: Code Segment of function "make_prediction" in "app.py" for passing in the user file to the model

The printed prediction output facilitates a straightforward comparison between the actual predicted results and the data displayed in both the table and the visualisation. Below are the specified commands utilised for the Integration of Prediction Output into Web Application Testing, employing the **intermediate output printing** methodology as shown in **Figure 117**:

Line 116: print(prediction_df)

```
    prediction_df = pd.DataFrame({
        'COSMIC_ID': user_df_morgan.loc[X_test.index, 'COSMIC_ID'],
        'CELL_LINE_NAME': user_df_morgan.loc[X_test.index, 'CELL_LINE_NAME'],
        'DRUG_ID': user_df_morgan.loc[X_test.index, 'DRUG_ID'],  # Retrieve 'DRUG_ID' using the index of the original DataFrame
        'DRUG_NAME': user_df_morgan.loc[X_test.index, 'DRUG_NAME'],
        'PRED_LN_IC50': y_pred.flatten()  # Flatten the y_pred array to make it one-dimensional
    })

    prediction_df['Resistance_Cut-Off'] = prediction_df['PRED_LN_IC50'].apply(lambda x: 'High' if x > 3.77 else 'Low')

    # Reset the index of the prediction DataFrame
    prediction_df.reset_index(drop=True, inplace=True)

    initial_row_count = prediction_df.shape[0]

    prediction_df.dropna(inplace=True)

    final_row_count = prediction_df.shape[0]
    dropped = False
    rows_dropped = initial_row_count - final_row_count
    if rows_dropped > 0:
        dropped = True

    print(prediction_df)

    return dropped, prediction_df
```

Figure 117: Code Segment of function "run_model" in "model.py" for generating tabular prediction output

The code to display data is provided in **Figures 118 and 119** is located within the **createTable** function. It reads the "path" parameter, which is the path that stores the prediction result csv file as data input. For integration of the table into the web application (HTML) file, we utilised the **<table>** element shown in **Figure 120** by fetching the path, converting to text and separating data properly by **split function** with **comma** as a delimiter. The table.innerHTML shown in **Figure 119** will insert the html element which is <table> with the predicted result to the HTML file, and this indicates that the table with the predicted result, represented by "drug-table" id will be displayed in the web application at its corresponding container. Finally, the ids "table-container" and "drug-table" will be passed into the HTML to display the prediction result table in the web interface.

```
394  function createTable(path) {
395    var uniqueValues1 = new Set();
396    var DELIMITER = ',';
397    var NEWLINE = '\n';
398    var filePath = path;
399    var table = document.getElementById('table-container');
400
401    if (!table) {
402      return;
403    }
404
405    fetch(filePath)
406      .then(response => response.text())
407      .then(text => {
408        toTable(text);
409      })
410      .catch(error => console.error('Error fetching file:', error));
```

Figure 118: Code Segments of "createTable" function responsible for retrieving the prediction result from path

```
471      tableHTML += '</tbody>';
472      table.innerHTML = '<table id="drug-table">' + tableHTML + '</table>';
```

Figure 119: Code Segments of "createTable" function responsible for creating prediction output table

```
156      <div id="table-container">
157        <table id="drug-table"></table>
158      </div>
```

Figure 120: Code Segment in "index.html" to display the prediction result table via the id "table-container" and "drug-table"

(d) Expected Outputs

In the **first test**, the details of expected outputs are provided in Section 5.4: Integration of Visualisation Output into Web Application Testing. In the **second test**, it is anticipated that no table is displayed upon the "User Manual Prediction" button is clicked and before the prediction process is completed. The table of prediction results is expected to be aligned with the output generated by print(prediction_df) in the terminal and will only be displayed in the web application once the message "Prediction completed successfully" is shown, indicating the finish of execution of the predictive model.

(e) Actual Outputs and Discussions

After making a prediction with the "sample_user_upload_1.csv" file, the prediction result is effectively showcased in the terminal output, as depicted in **Figure 121**, affirming the completion of the prediction process. The response status is confirmed as 200, indicating success, and the "GET" action retrieves the path of the prediction result CSV file to be utilised as an input parameter for the "createTable" function. Consequently, the integration of the "drug-table" into the HTML file ensures the successful display of the table in the web application, facilitated by the reading of the prediction result CSV file by the "createTable" function.

```
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step
    COSMIC_ID   CELL_LINE_NAME  DRUG_ID       DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0      749716   HCC2218_BREAST     1007       Docetaxel     -2.631801                Low
1      749716   HCC2218_BREAST     1010       Gefitinib      1.740488                Low
2      749716   HCC2218_BREAST     1030        KU-55933      1.975521                Low
3      749716   HCC2218_BREAST     1032        Afatinib      0.759863                Low
4      749716   HCC2218_BREAST     1046  Wee1 Inhibitor      1.911579                Low
5      749716   HCC2218_BREAST     1060       PD0325901      1.806497                Low
6      749716   HCC2218_BREAST     1507     Ruxolitinib      2.303889                Low
7      749716   HCC2218_BREAST     1922       Cediranib      1.690433                Low
8      749716   HCC2218_BREAST     1932      NVP-ADW742      1.522244                Low
9      908123  MDAMB468_BREAST     1007       Docetaxel     -3.145956                Low
10     908123  MDAMB468_BREAST     1010       Gefitinib      3.513914                Low
11     908123  MDAMB468_BREAST     1030        KU-55933      4.015914               High
12     908123  MDAMB468_BREAST     1032        Afatinib      2.305163                Low
13     908123  MDAMB468_BREAST     1046  Wee1 Inhibitor      3.859415               High
14     908123  MDAMB468_BREAST     1060       PD0325901      3.376752                Low
15     908123  MDAMB468_BREAST     1507     Ruxolitinib      4.411232               High
16     908123  MDAMB468_BREAST     1922       Cediranib      3.451726                Low
17     908123  MDAMB468_BREAST     1932      NVP-ADW742      3.201281                Low
18     908123  MDAMB468_BREAST     2359      GSK2830371      4.035609               High
127.0.0.1 - - [18/May/2024 03:37:26] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2024 03:37:26] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2024 03:37:26] "GET /results/user_prediction_result_20240518033726.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [18/May/2024 03:37:26] "GET /results/user_prediction_result_20240518033726.csv HTTP/1.1" 200 -
```

Figure 121: Terminal Output after Prediction Made

The consistency between the table output presented in **Figure 122** and the corresponding prediction results illustrated in **Figure 121** has been verified. The displayed data precisely mirrors the prediction outcomes, affirming the successful presentation of the prediction output in tabular format within the web application. This alignment serves as confirmation that the prediction output has been seamlessly integrated into the web application's table component.

COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_IC50	Resistance_Cut-Off
749716	HCC2218_BREAST	1007	Docetaxel	-2.6318011	Low
749716	HCC2218_BREAST	1010	Gefitinib	1.7404884	Low
749716	HCC2218_BREAST	1030	KU-55933	1.9755213	Low
749716	HCC2218_BREAST	1032	Afatinib	0.759863	Low
749716	HCC2218_BREAST	1046	Wee1 Inhibitor	1.9115787	Low
749716	HCC2218_BREAST	1060	PD0325901	1.8064975	Low
749716	HCC2218_BREAST	1507	Ruxolitinib	2.3038886	Low
749716	HCC2218_BREAST	1922	Cediranib	1.6904327	Low
749716	HCC2218_BREAST	1932	NVP-ADW742	1.5222436	Low
908123	MDAMB468_BREAST	1007	Docetaxel	-3.145956	Low

Figure 122: Table Output of Prediction

In conclusion, the testing of integration of prediction output into the web application has been successful, as all test cases have been **PASSED**. This is evidenced by the data displayed in the table is aligned with the prediction output, and the matching ids in HTML shows successful displays of the result table in the web application.

Section 5.4: Integration of Visualisation Output into Web Application Testing

(a) Test Description (What is being tested)

The focus of this test is to ensure the seamless integration of visualisation output into our web application interface. Specifically, we aim to verify how the visualisation is displayed within the web application, ensuring that the correct visualisation is being called and rendered. To achieve this, we need to meticulously examine how the software reads the prediction output from the model and utilises it to generate the visualisation. This involves validating that the correct path of the result outputted by the "Make Prediction" action is passed as a parameter into the code responsible for visualisation development. As a background, our visualisation is built using Vega-Lite, a popular tool for creating interactive visualisations. While the visualisation codes may be correct, there is a possibility that the integration into the web application may encounter issues. Even if the visualisation displays correctly in the Vega-Lite online editor, it does not guarantee seamless integration into our web application. Potential issues could arise from incorrect path passing or improper calling actions of the visualisation within the web application. Therefore, this testing aims to meticulously scrutinise the integration process and **ensure that the visualisation is seamlessly displayed within the web application interface**. We will explore different paths for testing, **validating each to ensure that the correct visualisation is displayed in the appropriate context.**

(b) Test Set-Up/ Methodology (How it is being tested)

For this testing phase, we will conduct two distinct tests to ensure the seamless integration of visualisation output into our web application interface.

Test 1: Correct Visualisation Input Path Reading

In the first test, we will focus on validating the correct reading of the visualisation input path. To begin, we will **upload the first file** "sample_user_upload_data_1.csv" and execute the "Make Prediction" action. During this process, we will carefully observe the prediction output path and note which path the createTable() function reads. Subsequently, we will **upload the second file** "sample_user_upload_data_2.csv" and repeat the prediction process, ensuring that the correct prediction output path is read as input to the visualisation code segment. It is essential to **verify that the visualisation is updated with the results of the new prediction**, indicating that the **correct input path has been successfully passed** to the visualisation code segment, instead of reading the result of displaying the results of the first upload file upon executing the second file prediction.

Test 2: Integration of Visualisation Display into Web Application

In the second test, we will focus on testing the integration of the visualisation display into our web application using the **VegaEmbed function**. Initially, we will execute the visualisation code **without applying VegaEmbed** to observe whether the visualisation is being outputted correctly. Following this, we will **apply VegaEmbed** to embed the Vega-Lite JSON code into the HTML. We will closely monitor whether the visualisation is successfully displayed after applying VegaEmbed. To conduct this test effectively, we will add and remove the VegaEmbed line of code to assess its impact on the visualisation display.

(c) Inputs to the code or part of code being tested

The "web.js" file contains a function called createTable, shown in **Figure 123**, which is responsible for reading the path of the resulting CSV file generated when a prediction is made.

```
// create table and visualisation, just pass the file path as parameter
function createTable(path) {
  var uniqueValues1 = new Set();
  var DELIMITER = ',';
  var NEWLINE = '\n';
  var filePath = path;
  var table = document.getElementById('table-container');

  if (!table) {
    return;
  }

  fetch(filePath)
    .then(response => response.text())
    .then(text => {
      toTable(text);
    })
    .catch(error => console.error('Error fetching file:', error));
```

Figure 123: createTable function and its input parameter, "path"

The visualisation json code provided in **Figure 124** is located within the createTable function. It reads the "path" parameter of the function in Line 486, which is the path of the prediction result csv file as data input.

```
471         tableHTML += '</tbody>';
472         table.innerHTML = '<table id="drug-table">' + tableHTML + '</table>';
473         var arr1 = Array.from(uniqueValues1);
474         var jsonData = {
475           "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
476           "title": {
477             "text": "Bar Chart of Drug Resistance on Different Breast Cancer Cell \n",
478             "align": "center",
479             "fontSize": 31,
480             "offset": 20
481           },
482           "width": "container",
483           // "width": 2600,
484           "background": "white",
485           "data": {
486             "url": path
487           },
```

Figure 124: Code segment on how path is being passed in as input Data in the vega-lite json code for visualisation

Lastly, for integration of visualisation chart into the web application (HTML) file, we utilised the "vegaEmbed" function shown in **Figure 125** by passing in the "jsonData" variable, which contains the code of the visualisation development, into the container in the HTML having id as "bar_chart". This will therefore display the output in the web application at its corresponding container.

```
    vegaEmbed('#bar_chart', jsonData, { "actions": false }).then(function (result) {
      // Access the Vega view instance (https://vega.github.io/vega/docs/api/view/) as result.view
      var dropdownButton = document.querySelector('.vega-bindings');
      if (dropdownButton) {
        dropdownButton.style.position = 'absolute';
        dropdownButton.style.top = '10px'; // Adjust as needed
        dropdownButton.style.left = '10px'; // Adjust as needed
        dropdownButton.style.position = 'absolute';
        dropdownButton.style.top = '10px'; // Adjust as needed
        dropdownButton.style.left = '10px'; // Adjust as needed
        dropdownButton.style.padding = '10px'; // Add padding as needed
      }
    }
  ).catch(console.error);
```

Figure 125: Code segment of implementation of vegaEmbed

To facilitate testing, the vegaEmbed code will be commented out, which is demonstrated in **Figure 126,** to assess whether the visualisation is indeed reliant on vegaEmbed for display within the web application. Observations will be made upon with and without the execution of "vegaEmbed"

```
584         //   vegaEmbed('#bar_chart', jsonData, { "actions": false }).then(function (result) {
585         //     // Access the Vega view instance (https://vega.github.io/vega/docs/api/view/) as result.view
586         //     var dropdownButton = document.querySelector('.vega-bindings');
587         //     if (dropdownButton) {
588         //       dropdownButton.style.position = 'absolute';
589         //       dropdownButton.style.top = '10px'; // Adjust as needed
590         //       dropdownButton.style.left = '10px'; // Adjust as needed
591         //       dropdownButton.style.position = 'absolute';
592         //       dropdownButton.style.top = '10px'; // Adjust as needed
593         //       dropdownButton.style.left = '10px'; // Adjust as needed
594         //       dropdownButton.style.padding = '10px'; // Add padding as needed
595         //     }
596         //   }
597         // ).catch(console.error);
```

Figure 126: Commented parts of vegaEmbed for testing

(d) Expected Outputs

In the first test, upon making predictions on the initial testing file, the terminal output should display the filename of the uploaded file along with the path of the prediction output file. Subsequently, when predictions are made on the second testing file, the terminal should reflect the filename of the second uploaded file, and the **path of the prediction output file of the second prediction should differ from the first one**. Verification of the visualisations produced should **reveal distinct outputs for each test**, indicating the **uniqueness of the prediction results corresponding to the input files**. Furthermore, **alignment between the drug-cell pairs** depicted in the visualisations and those in the prediction output would confirm the correct data being passed in as input for visualisation.

In the second test, it is anticipated that **no visualisation will be generated in the absence of the vegaEmbed code**. This absence of visualisation would corroborate the necessity of vegaEmbed for successful display. This scenario will be contrasted with the outcome of the first test, where the involvement of vegaEmbed in the visualisation process was established.

(e) Actual Outputs and Discussions

Test 1:

Figure 127 shows that the "sample_user_upload_integration_test1.csv" is being read.

```
Received JSON data: {'uploaded_filename': 'sample_user_upload_integration_test1.csv'}
Constructed CSV path: user_file\sample_user_upload_integration_test1.csv
(19, 19226)
```

Figure 127: Terminal Output of Reading of First User Upload File

Upon making a prediction, the prediction output for the first prediction made is stored in the path **"/results/user_prediction_result_2024051617050**1**.csv HTTP/1/1"** as shown in **Figure 128.** There is a "GET" action to retrieve the path to be passed in as input parameter to the "createTable" function.

```
    COSMIC_ID   CELL_LINE_NAME  DRUG_ID       DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0      749716   HCC2218_BREAST     1007       Docetaxel     -2.631801                Low
1      749716   HCC2218_BREAST     1010       Gefitinib      1.740488                Low
2      749716   HCC2218_BREAST     1030        KU-55933      1.975521                Low
3      749716   HCC2218_BREAST     1032       Afatinib      0.759863                Low
4      749716   HCC2218_BREAST     1046  Wee1 Inhibitor      1.911579                Low
5      749716   HCC2218_BREAST     1060       PD0325901      1.806497                Low
6      749716   HCC2218_BREAST     1507     Ruxolitinib      2.303889                Low
7      749716   HCC2218_BREAST     1922       Cediranib      1.690433                Low
8      749716   HCC2218_BREAST     1932      NVP-ADW742      1.522244                Low
9      908123  MDAMB468_BREAST     1007       Docetaxel     -3.145956                Low
10     908123  MDAMB468_BREAST     1010       Gefitinib      3.513914                Low
11     908123  MDAMB468_BREAST     1030        KU-55933      4.015914               High
12     908123  MDAMB468_BREAST     1032       Afatinib      2.305163                Low
13     908123  MDAMB468_BREAST     1046  Wee1 Inhibitor      3.859415               High
14     908123  MDAMB468_BREAST     1060       PD0325901      3.376752                Low
15     908123  MDAMB468_BREAST     1507     Ruxolitinib      4.411232               High
16     908123  MDAMB468_BREAST     1922       Cediranib      3.451726                Low
17     908123  MDAMB468_BREAST     1932      NVP-ADW742      3.201281                Low
18     908123  MDAMB468_BREAST     2359      GSK2830371      4.035609               High
127.0.0.1 - - [16/May/2024 17:05:01] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:05:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [16/May/2024 17:05:01] "GET /results/user_prediction_result_20240516170501.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:05:01] "GET /results/user_prediction_result_20240516170501.csv HTTP/1.1" 200
-
127.0.0.1 - - [16/May/2024 17:05:01] "GET /results/user_prediction_result_20240516170501.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:05:01] "GET /results/user_prediction_result_20240516170501.csv HTTP/1.1" 304
```

Figure 128: Terminal Output after First Prediction Made

The alignment between the visualisation output of the initial prediction shown in **Figure 129** and the corresponding prediction results is confirmed by the matching default COSMIC_ID of "749716". This coherence underscores the accurate transmission of the result path into the visualisation code.

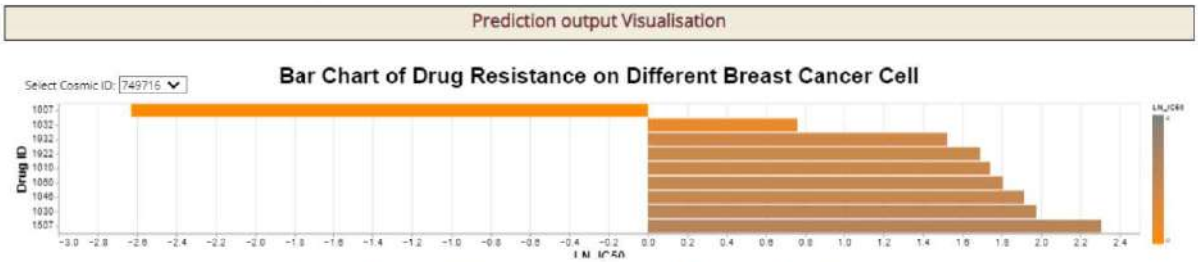


Figure 129: Visualisation Output of First Prediction Made

Next, when the second prediction file is uploaded, **Figure 130** shows that the "sample_user_upload_integration_test _2.csv" is being read.

```
Received JSON data: {'uploaded_filename': 'sample_user_uploadintegration_test_2.csv'}
Constructed CSV path: user_file\sample_user_uploadintegration_test_2.csv
(20, 19226)
```

Figure 130: Terminal Output of Reading of SecondUser Upload File

Upon making a prediction, the prediction output for the second prediction made is stored in the path **"/results/user_prediction_result_20240516170947.csv HTTP/1/1"** as shown in **Figure 131.** There is a "GET" action to retrieve the path to be passed in as input parameter to the "createTable" function. It is shown that the path of the output for the second prediction is **different from that of the first prediction.**

```
    COSMIC_ID  CELL_LINE_NAME  DRUG_ID     DRUG_NAME  PRED_LN_IC50 Resistance_Cut-Off
0      749710  HCC1143_BREAST     1003  Camptothecin      0.017473                Low
1      749710  HCC1143_BREAST     1004   Vinblastine     -2.010930                Low
2      749710  HCC1143_BREAST     1005     Cisplatin      3.075493                Low
3      749710  HCC1143_BREAST     1006    Cytarabine      2.170400                Low
4      749710  HCC1143_BREAST     1007     Docetaxel     -3.163481                Low
5      749710  HCC1143_BREAST     1008  Methotrexate     -0.134297                Low
6      749710  HCC1143_BREAST     1010     Gefitinib      2.043282                Low
7      749710  HCC1143_BREAST     1011    Navitoclax      1.044287                Low
8      749710  HCC1143_BREAST     1012     Vorinostat      2.234767                Low
9      749710  HCC1143_BREAST     1013     Nilotinib      2.067947                Low
10     906801      BT20_BREAST     1003  Camptothecin      1.995018                Low
11     906801      BT20_BREAST     1004   Vinblastine     -0.107481                Low
12     906801      BT20_BREAST     1005     Cisplatin      3.931827               High
13     906801      BT20_BREAST     1006    Cytarabine      3.070072                Low
14     906801      BT20_BREAST     1007     Docetaxel     -1.254485                Low
15     906801      BT20_BREAST     1008  Methotrexate      1.712903                Low
16     906801      BT20_BREAST     1010     Gefitinib      3.043389                Low
17     906801      BT20_BREAST     1011    Navitoclax      2.564304                Low
18     906801      BT20_BREAST     1012     Vorinostat      3.144807                Low
19     906801      BT20_BREAST     1013     Nilotinib      3.060961                Low
127.0.0.1 - - [16/May/2024 17:09:47] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:09:47] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [16/May/2024 17:09:47] "GET /results/user_prediction_result_20240516170947.csv HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:09:47] "GET /results/user_prediction_result_20240516170947.csv HTTP/1.1" 200
-
127.0.0.1 - - [16/May/2024 17:09:47] "GET /results/user_prediction_result_20240516170947.csv HTTP/1.1" 304 -
INFO:werkzeug:127.0.0.1 - - [16/May/2024 17:09:47] "GET /results/user_prediction_result_20240516170947.csv HTTP/1.1" 304
```

Figure 131: Terminal Output after Second Prediction Made

The alignment between the visualisation output of the second prediction shown in **Figure 132** and the corresponding prediction results is confirmed by the matching default COSMIC_ID of "749710". This indicates that the correct output file path has been successfully read, eliminating any errors associated with reading the previous output file. The resulting output is directly tied to the input user upload file, **affirming the specificity of the prediction outcome**.

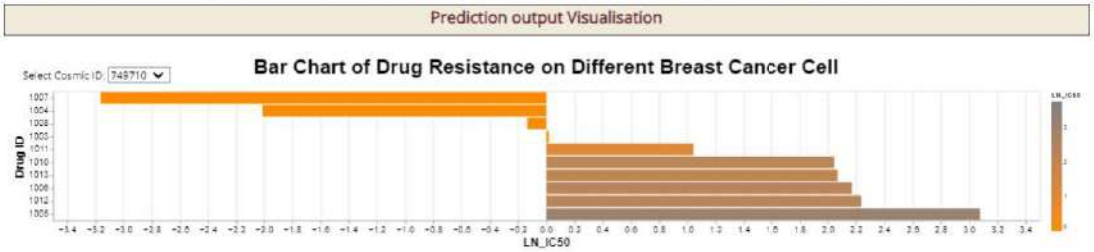


Figure 132: Visualisation Output of Second Prediction Made

Test 2:

After commenting out the vegaEmbed code, it becomes evident that no visualisation is displayed anymore, as no variable representing the visualisation is passed into the visualisation container via the visualisation id. This further confirms that we had previously successfully integrated the visualisation into the web application using vegaEmbed.

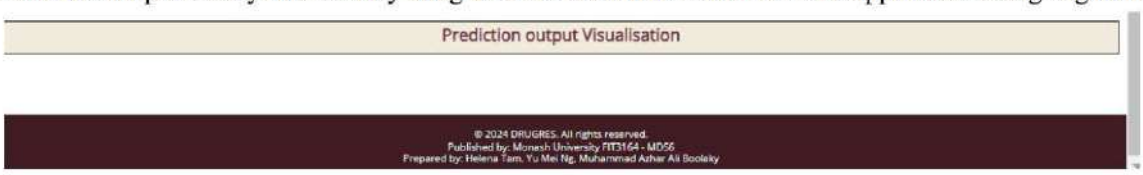


Figure 133: Visualisation Container Display after Commenting Out "VegaEmbed"

In conclusion, the testing of integrating visualisation output into the web application has been successful, as all test cases have been **PASSED**. This is evidenced by the correct reading of valid file paths specific to each uploaded input file, and the successful display of visualisations using vegaEmbed.

Section 6: Usability Testing

Section 6.1: User Interface/ User Experience Testing

Section 6.1.1: User Interface Navigation Testing

(a) Test Description (What is being tested)

The web application comprises two pages: the main page for prediction actions and a user guideline page for instruction. This separation aims to enhance user experience and application performance by allocating each page to a specific purpose. Navigation between these pages relies on button elements triggering navigation actions. Additionally, distinct user interface designs are displayed based on functions triggered by the "User Manual Prediction" and "Default Dataset" buttons. Testing **aims to ensure seamless transition between these interface designs upon button clicks**, ensuring a smooth user experience. All buttons are clicked to verify the expected interface changes.

(b) Test Set-Up/ Methodology (How it is being tested)

When users click various buttons, they trigger navigation actions that switch the display of the user interface. To ensure comprehensive testing of interface changes, three methodologies will be employed: physical code observation, intermediate output printing, and physical software output observation. Testing will encompass different button scenarios to observe outputs:

Case 1: Transition from the main page to the user guideline page.
Case 2: Return from the user guideline page to the main page.
Case 3: Change from the interface of default dataset interface to manual prediction interface.
Case 4: Shift from the manual prediction interface back to the default dataset interface.

Step 1: The code (app.py and web.js) undergoes physical observation to verify proper implementation of the navigation function.

Step 2: Intermediate printing commands, detailed in part (c), are added to print specific messages under corresponding function headers to confirm activation upon button clicks.
Case 1: Print "User Guideline page is call" text under the function header of user_guideline()
Case 2: No function can be called, a default back button of web page
Case 3: Print "User Manual Prediction is call" text under the function header of manualtButton()
Case 4: Print "Default Dataset is call" text under the function header of defaultButton()

Step 3: In the terminal of Visual Studio Code, run the command: python app.py

Step 4: Click on the specific button for different cases:
Case 1: Click on the "Manual User Guideline" button as shown in **Figure 134**

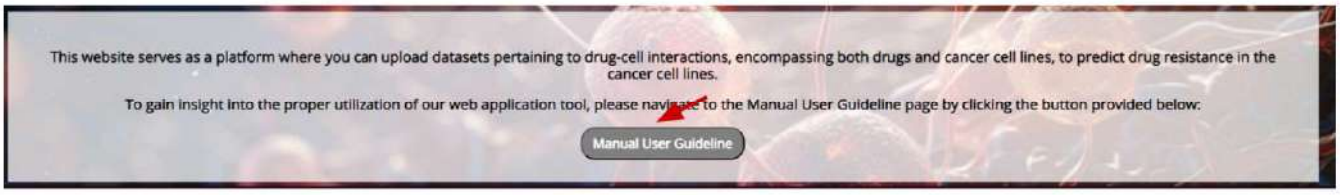


Figure 134: Action of Clicking on the Manual User Guideline Button

Case 2: Click on the "Back" button as shown in **Figure 135**

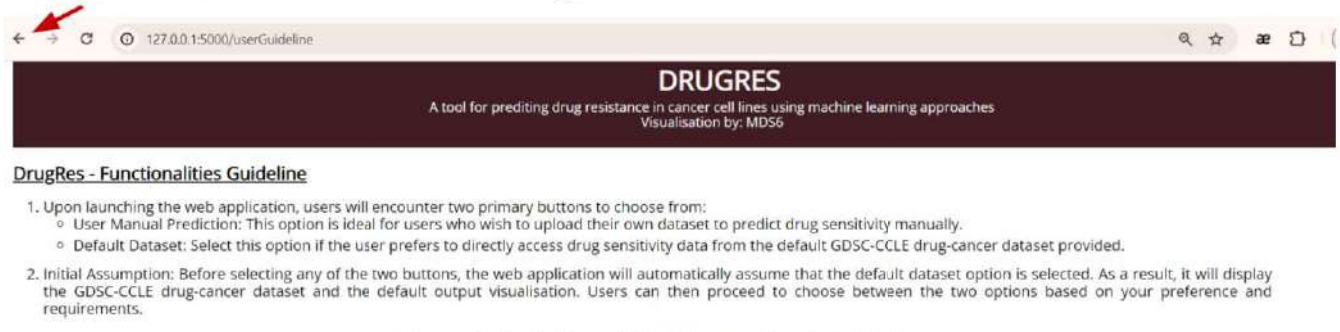


Figure 135: Action of Clicking on the Back Button

Case 3: Click on the "User Manual Prediction" button as shown in **Figure 136**

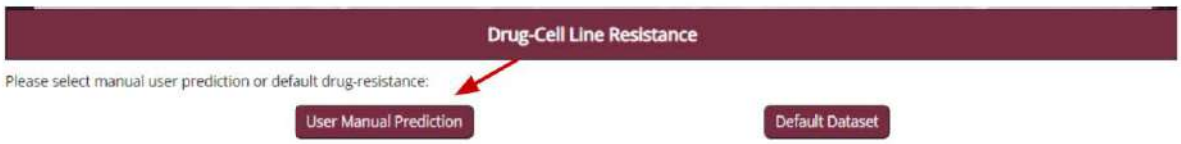


Figure 136: Action of Clicking on the User Manual Prediction Button

Case 4: Click on the "Default Dataset" button as shown in **Figure 137**

Figure 137: Action of Clicking on the Default Dataset Button

Step 5: Observe the printings output in the terminal

Step 6: Observe the user interface after the button is clicked

(c) Inputs to the code or part of code being tested

The commands used (**intermediate output printing** methodology) for the user interface navigation testing are specified below and shown in **Figures 138, 139 and 140**:

(Case 1): Line 214: print("User Guideline page is call") in app.py (shown in **Figure 138**)
(Case 3): Line 64: print("User Manual Prediction is call"); in web.js (shown in **Figure 139**)
(Case 4): Line 86: print("Default Dataset is call"); in web.js (shown in **Figure 140**)

When the application is launched, "Manual User Guideline" button will be clicked to navigate the user to the user guideline page. Upon executing the navigation action, the function "**user_guideline**" in the file "**app.py**" shown in **Figure 138** below will be executed and shows the printing of "User Guideline page is call" (Case 1). Besides, the default back button will be clicked to navigate the user back from the user guideline page to the main page (Case 2).

```
212  @app.route('/userGuideline')
213  def user_guideline():
214      print("User Guideline page is call")
215      return render_template('userGuideline.html')
```

Figure 138: Code Segment in "app.py" of the function "user_guideline" (case 1)

Next, "User Manual Prediction" button will be clicked to switch the design of the default dataset interface to the user manual prediction interface. Upon executing the navigation action, the function "**manualtButton**" in the file "**web.js**" shown in **Figure 139** below will be executed and shows the printing of "User Manual Prediction is call" (Case 3).

```
84  //use to choose manual prediction
85  function manualtButton(){
86    console.log("User Manual Prediction is call");
87    var fileUploadSec = document.getElementById("fileUploadSection");
88    var predictionSec = document.getElementById('predictionSection');
89    var visualLabel= document.getElementById("predictionOutputLabel");
90    var predictionLabel = document.getElementById('predictionLabel');
91    var table = document.getElementById('table-container');
92    var visual = document.getElementById('bar_chart');
```

Figure 139: Code Segment in "web.js" of the function "manualtButton" (case 3)

Lastly, the "**defaultButton**" function in "**web,js**" of the web page shown in **Figure 140** will be executed when the user clicked on the "Default Dataset" button to navigate back to the main page and shows the printing of "Default Dataset is call" (Case 4).

```
62  //display default dataset
63  function defaultButton(){
64    console.log("Default Dataset is call");
65    var fileUploadSec = document.getElementById("fileUploadSection");
66    var predictionSec = document.getElementById('predictionSection');
67    var visualLabel= document.getElementById("predictionOutputLabel");
68    var predictionLabel = document.getElementById('predictionLabel');
```

Figure 140: Code Segment in "web.js" of the function "defaultButton" (case 4)

(d) Expected Outputs

All the messages for different cases are expected to be printed out in the terminal when the correspondence button is clicked.

Case 1: Command: print("User Guideline page is call")

>> Expected Output: **"User Guideline page is call"** printed in the terminal prompt. The interface displayed should be the **user guideline interface**.

Case 2: Nothing is printed because it is the web browser default back action, the interface displayed should be the main page (**default dataset interface**).

Case 3: Command: console.log("User Manual Prediction is call")

>> Expected Output: "**User Manual Prediction is call**" printed in the terminal prompt and shown in the inspection console. The interface displayed should be the **user manual prediction interface**.

Case 4: Command: console.log("Default Dataset is call")

>> Expected Output: "**Default Dataset is call**" printed in the terminal prompt and shown in the inspection console. The interface displayed should be the main page (**default dataset interface**).

(e) Actual Outputs and Discussions

Upon examination of the actual outputs, it becomes evident that they all correspond with the anticipated results outlined in part (d). Different messages are printed successfully depending on which button is clicked.

For **Case 1**, "User Guideline page is call" has been successfully printed as shown in **Figure 141**, and the navigation to the user guideline page is successful as shown in **Figure 142**.

```
 * Detected change in 'C:\\FIT3164_Web_App\\MDS6_FIT3164\\Predictive Model Web Application\\src\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 132-591-143
User Guideline page is call
127.0.0.1 - - [17/May/2024 17:50:14] "GET /userGuideline HTTP/1.1" 200 -
127.0.0.1 - - [17/May/2024 17:50:14] "GET /static/css/userGuideStyle.css HTTP/1.1" 304 -
```

Figure 141: Terminal Output for User Interface Navigation Testing (case 1)

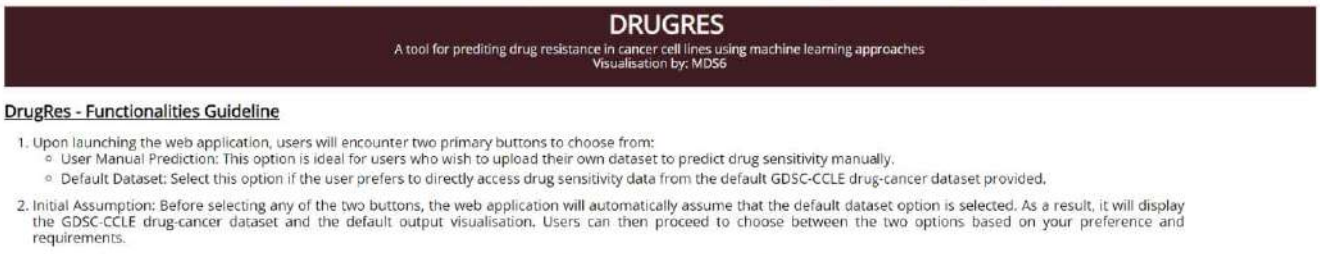


Figure 142: Interface Displayed for Case 1

For **Case 2,** it successfully directs back to the main page as shown in **Figure 143**.

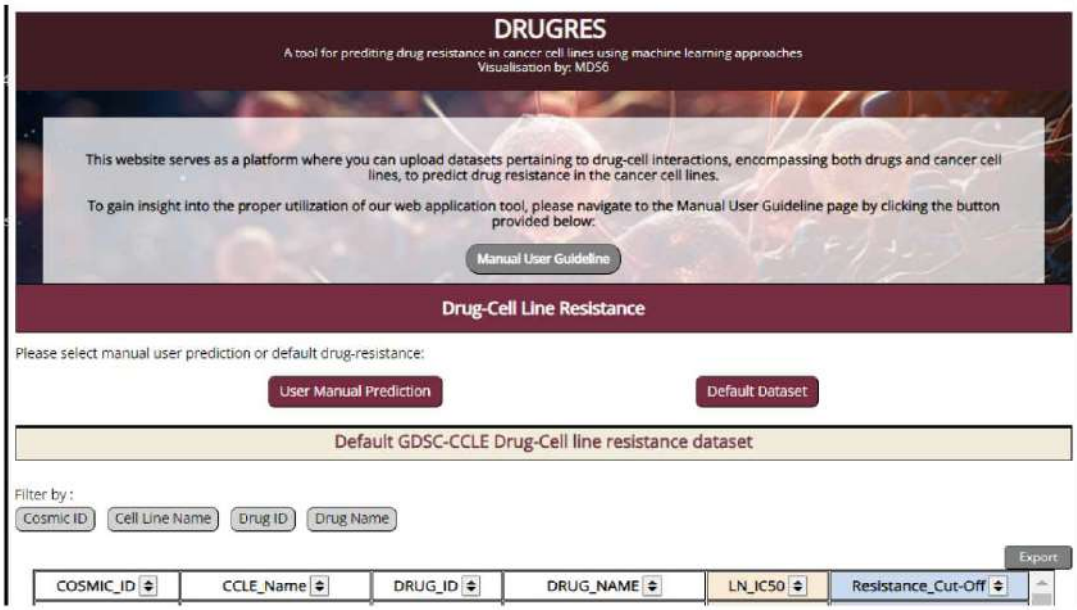


Figure 143: Interface Displayed for Case 2

For **Case 3**, once clicked on the "User Manual Prediction" button, the dropbox, "Make Prediction" button and the "Clear Prediction" button are displayed at the correct corresponding position, and the table and visualisation sections are empty, showing success navigation to the user manual prediction interface shown in **Figure 144**. The inspection console shows the printing of "User Manual Prediction is call", as shown in **Figure 145**.

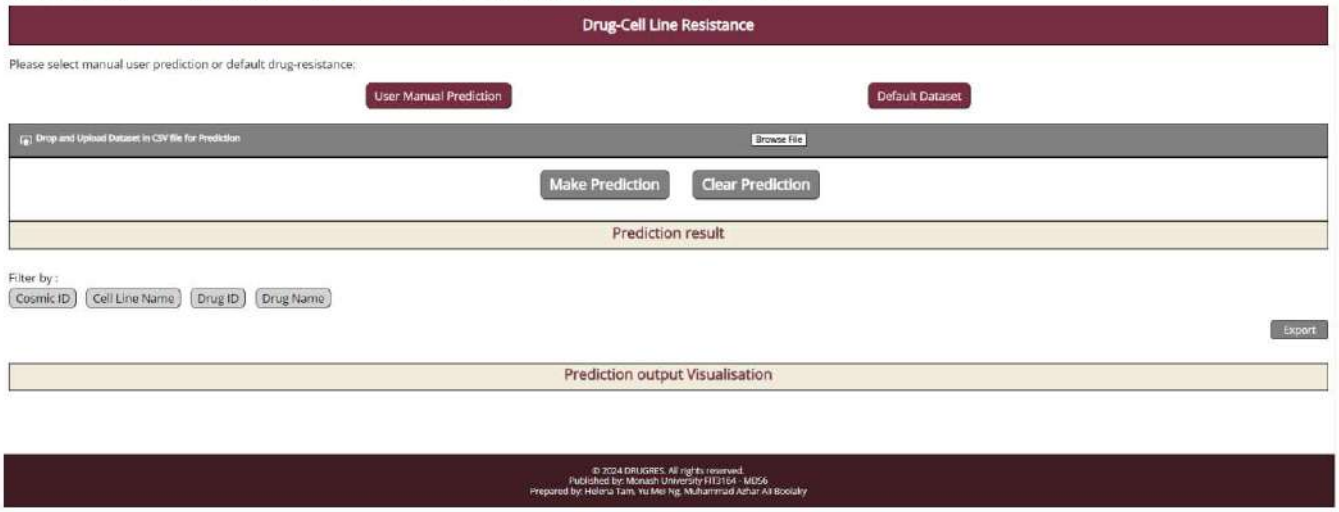


Figure 144: Interface Displayed for Case 3

Figure 145: Inspection Console Output for Case 3

Lastly, for **Case 4**, once clicked on the "Default Dataset" button, the dropbox, "Make Prediction" button and the "Clear Prediction" button are successfully hidden from the user interface, and the default table and visualisation are displayed as shown in **Figure 146**. The inspection console shows success printing of "Default Dataset is call", as shown in **Figure 147.**

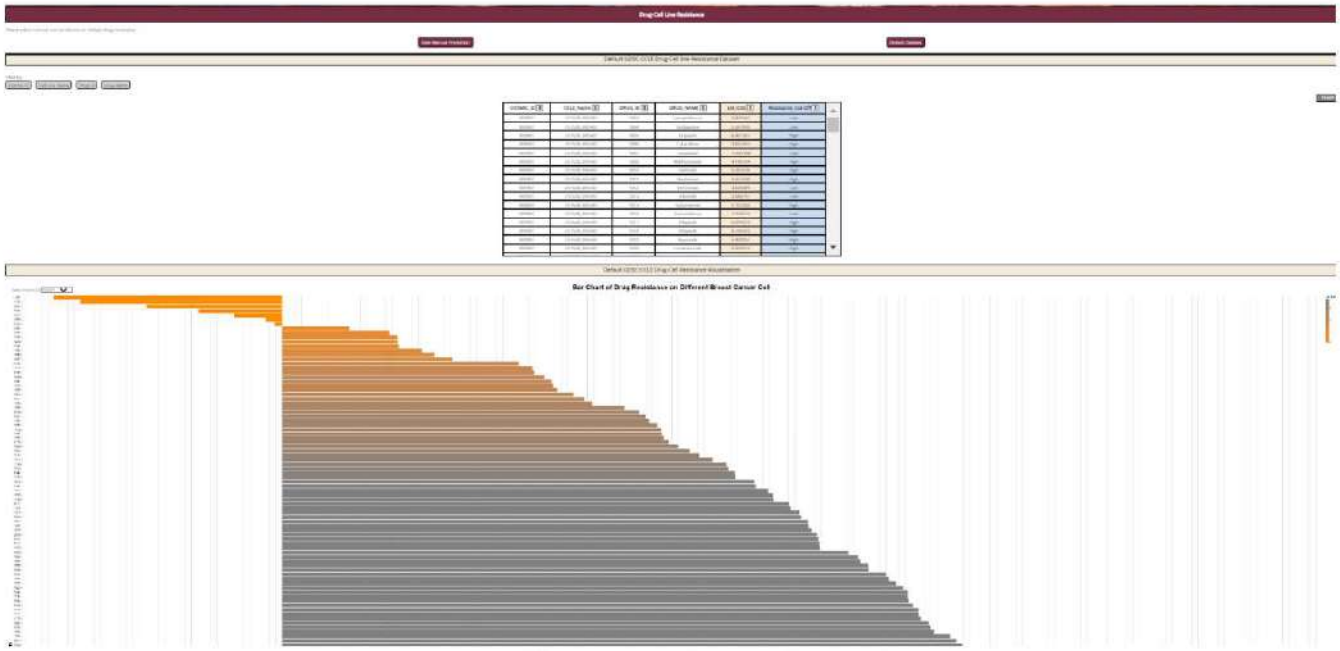


Figure 146: Interface Displayed for Case 4

Figure 147: Inspection Console Output for Case 4

Therefore, we can conclude that all the cases for "User Interface Navigation Testing" have **PASSED**.

Section 6.1.2: Upload and Export Dataset Testing

(a) Test Description (What is being tested)

The upload and export dataset testing aims to evaluate the functionality and user experience of the upload and export features in the application. Specifically, we are testing whether users can successfully upload a file for prediction and export the prediction results. The upload functionality is tested by allowing users to upload a file either by dragging and dropping it into the designated grey box or by clicking the "Browse File" button to select the file manually. The export functionality is tested by ensuring that once the prediction is completed, users can download the result dataset, verifying that the content of the downloaded file matches the displayed prediction results.

(b) Test Set-Up/ Methodology (How it is being tested)

To test the upload and export functionalities, we will use a dataset named "sample_user_upload_data_7.csv," which has correct dimensions and no missing values (NA). The testing procedure involves the following steps:

Step 1: Upload the dataset using the **drag-and-drop** method. (**Grey colour** area in **Figure 148**)
Step 2: Upload the dataset using the "**Browse File**" button.
Step 3: Initiate the prediction process and ensure the table is displayed correctly on the web interface.
Step 4: Click the "**Export**" button to download the prediction result file.
Step 5: Verify that the **content** of the downloaded file matches the prediction results displayed in the table.

During this process, we will monitor and record any issues encountered, such as upload failures, discrepancies in the displayed and exported data, or difficulties in using either upload method.

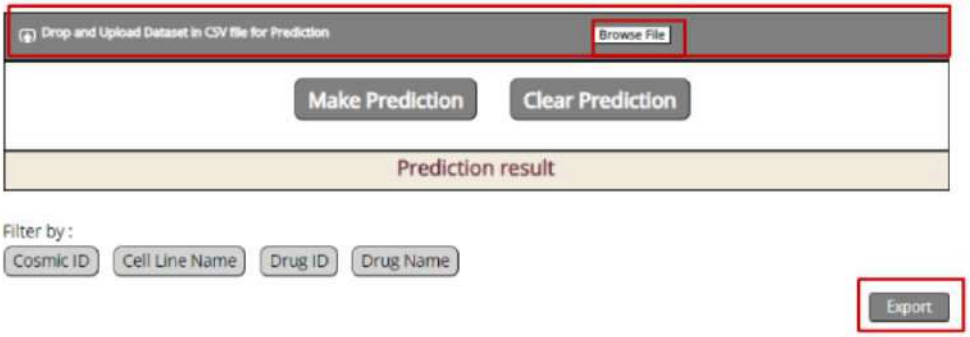


Figure 148: Buttons and Sections in Web Interface for Performing Upload and Export Tasks

(c) Inputs to the code or part of code being tested

Figure 149 shows the "upload_file" function in "app.py," which is responsible for reading the user-uploaded data and storing it in the "user_file" folder within the corresponding user's session.

```
234  @app.route('/upload', methods=['POST'])
235  def upload_file():
236      if 'file' not in request.files:
237          return jsonify({'message': 'No file part'}), 400
238
239      file = request.files['file']
240      if file.filename == '':
241          return jsonify({'message': 'No selected file'}), 400
242
243      # filename = secure_filename(file.filename)
244      filename = file.filename
245      # Basic security check to remove path information
246      filename = os.path.basename(filename)
247      file_path = os.path.join(app.config['USER_FOLDER'], filename)
248      file.save(file_path)
249
250       # Save the filename in the user's session
251      session['uploaded_filename'] = filename
252
253      return jsonify({'message': f'Uploaded file: {filename}', 'filename': filename})
254
```

Figure 149: upload_file function from app.py

Figure 150 and Figure 151 shows the sections of code in "web.js" responsible for handling the browse file action and the drag-and-drop action via the EventListener, which monitors and responds to button clicks.

```
84   function manualtButton(){
131
132    // browse file
133    uploadedFileName = document.getElementById("uploadedFileName");
134      var fileInput = document.getElementById("fileInput");
135      var browseButton = document.getElementById("browseButton");
136      var dropArea = document.querySelector(".drop-area");
137      var predictionMade = false;
138
139      // When the "Browse File" button is clicked, simulate a click on the hidden file input
140      browseButton.addEventListener('click', function () {
141        fileInput.click();
142      });
```

Figure 150: Code Segment of the "manualtButton" from "web.js" responsible for Browse File Actions

```
175    // Unhighlight drop area on drag leave and drop
176    ['dragleave', 'drop'].forEach(eventName => {
177      dropArea.addEventListener(eventName, () => dropArea.classList.remove('highlight'), false);
178    });
179
180    dropArea.addEventListener('drop', function (e) {
181      var files = e.dataTransfer.files;
182    e.preventDefault();
183    e.stopPropagation();
184    if (files.length > 0) {
185      const file = files[0];
186      uploadedFileName = file.name;
187      uploadedFileNameDisplay.textContent = `Uploaded File: ${uploadedFileName}`;  // Update display
188      handleFiles(files);  // Then call handleFiles
189    }
```

Figure 151: Code Segment of the "manualtButton" from "web.js" responsible for Drag and Drop Actions

Figure 152 shows the code segment in "web.js" responsible for handling export actions. It includes the retrieval via Event Listener and facilitates the calling of the "download" function.

```
556  exportBtn.addEventListener("click", function() {
557
558    var table = document.getElementById("drug-table");
559      var csv = [];
560      var rows = table.rows;
561      for (var i = 0; i < rows.length; i++) {
562        var row = [], cols = rows[i].cells;
563        for (var j = 0; j < cols.length; j++) {
564          row.push(cols[j].innerText);
565        }
566        csv.push(row.join(","));
567      }
568      csv = csv.join("\n");
569    var blob = new Blob([csv], { type: "text/csv;charset=utf-8" });
570    var link = document.createElement("a");
571    link.href = URL.createObjectURL(blob);
572    link.download = "LN_IC50.csv";
573    link.click();
574  });
575
576  function download(){
577    var link = document.getElementById("downloadSample");
578    link.download = "sample_user_data.csv";
579  }
```

Figure 152: export function from "web.js"

(d) Expected Outputs

The expected outcomes of this testing phase are:

(1) Users should be able to upload the dataset using both the **drag-and-drop** method and the "**Browse File**" button **without any errors**.

(2) The prediction process should execute smoothly, resulting in the correct display of prediction results in the table format on the web interface.

(3) Upon clicking the "**Export**" button, the output file should be **successfully downloaded** to the user's device.

(4) The **content** of the downloaded file should exactly **match** the data displayed in the web interface's prediction results section.

By ensuring these outcomes, we aim to validate the usability and reliability of the upload and export features, ensuring a seamless user experience from data input to result retrieval.

(e) Actual Outputs and Discussions

As shown in **Figure 153**, the file is successfully uploaded using both of the methods. The file name being printed at the grey area, indicating successful update action.

Figure 153: Web Interface showing File Uploaded Successfully

To further verify that the upload process has been successful, we can check the "user_file" folder for the presence of a new file named "sample_user_upload_data_7.csv," as shown in **Figure 154**.

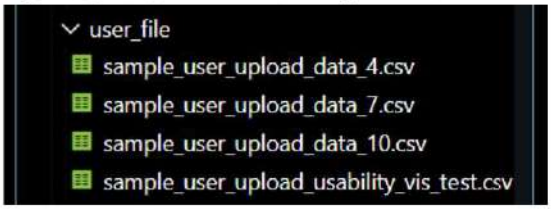


Figure 154: File goes into folder "user_file"

Lastly, upon clicking on the "export" button, a file "LN_IC50" is successfully downloaded as shown in **Figure 155**.

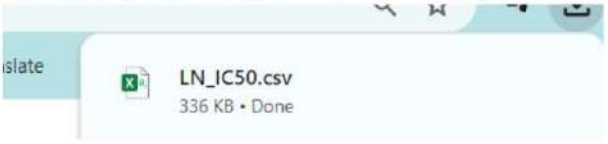


Figure 155: Exported/Downloaded output file

Upon verifying the content of the file, it is shown that the content matches the table being displayed in the web interface, showing success and correct export of the prediction result file as shown in **Figure 156** and **157**.

COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_IC50	Resistance_Cut-Off
1290798	EFM192A_BREAST	1058	Pictilisib	1.7192758	Low
909778	UACC893_BREAST	1011	Navitoclax	1.080844	Low
906844	DU4475_BREAST	1175	Rucaparib	0.69158685	Low
908123	MDAMB468_BREAST	1598	LGK974	4.6478477	High
908122	MDAMB453_BREAST	1059	AZD8055	1.6923134	Low
910852	CAL851_BREAST	1997	WEHI-539	3.127102	Low
925338	MDAMB157_BREAST	1012	Vorinostat	4.659746	High

Figure 156: Prediction Result Displayed in Web Interface

	A	B	C	D	E	F
1	COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_	Resistance_Cut-Off
2	1290798	EFM192A_BREAST	1058	Pictilisib	1.719276	Low
3	909778	UACC893_BREAST	1011	Navitoclax	1.080844	Low
4	906844	DU4475_BREAST	1175	Rucaparib	0.691587	Low
5	908123	MDAMB468_BREAST	1598	LGK974	4.647848	High
6	908122	MDAMB453_BREAST	1059	AZD8055	1.692313	Low
7	910852	CAL851_BREAST	1997	WEHI-539	3.127102	Low
8	925338	MDAMB157_BREAST	1012	Vorinostat	4.659746	High

Figure 157: Content of Downloaded "LN_IC50.csv"

Based on outputs given and observed, we can conclude that all the cases for " Upload and Export Dataset Testing" have **PASSED**.

Section 6.1.3: Table Sorting and Filtering Testing

(a) Test Description (What is being tested)

To enhance user experience in navigating large datasets, the presence of sort and filter buttons facilitates data arrangement and extraction, aiding users in finding relevant information efficiently. The test aims to **ensure that data is sorted based on specific orders and that targeted data is extracted accurately**. Event-triggered actions via buttons facilitate sorting and filtering processes, with four filter buttons designated for COSMIC_ID, CCLE_Name, DRUG_ID, and DRUG_NAME columns, and six sort buttons for COSMIC_ID, CCLE_Name, DRUG_ID, DRUG_NAME, LN_IC50, and Resistance_Cut-Off columns. Testing will validate data arrangement in ascending and descending orders and verify the functionality of dropdown menus activated upon button clicks. All buttons will be tested to ensure expected outputs are achieved.

(b) Test Set-Up/ Methodology (How it is being tested)

Assuming initial dataset alignment, the **first checkbox values** will be utilised for testing. User interactions with buttons trigger sorting and filtering actions, affecting data organisation and extraction. To validate code functionality, three methodologies will be employed: ***physical code observation***, ***intermediate output printing***, and ***software output observation***. Testing will encompass various button scenarios to observe corresponding outputs. **Figure 158** illustrates the initial table data display.

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low
909907	ZR7530_BREAST	1008	Methotrexate	4.140704	High
909907	ZR7530_BREAST	1010	Gefitinib	5.263526	High
909907	ZR7530_BREAST	1011	Navitoclax	5.557596	High

Figure 158: Part of Default Dataset being Displayed

Cases to be tested are summarised in Table 2 below:

Case	Description
1	Data is sorted in ascending order by COSMIC_ID upon the first click of the COSMIC_ID column sorting button.
2	Data is sorted in descending order by COSMIC_ID upon the second click of the COSMIC_ID column sorting button.
3	Data is sorted in ascending order by CCLE_Name upon the first click of the CCLE_Name column sorting button.
4	Data is sorted in descending order by CCLE_Name upon the second click of the CCLE_Name column sorting button.
5	Data is sorted in ascending order by DRUG_ID upon the first click of the DRUG_ID column sorting button.
6	Data is sorted in descending order by DRUG_ID upon the second click of the DRUG_ID column sorting button.
7	Data is sorted in ascending order by DRUG_NAME upon the first click of the DRUG_NAME column sorting button.
8	Data is sorted in descending order by DRUG_NAME upon the second click of the DRUG_NAME column sorting button.
9	Data is sorted in ascending order by LN_IC50 upon the first click of the LN_IC50 column sorting button.
10	Data is sorted in descending order by LN_IC50 upon the second click of the LN_IC50 column sorting button.
11	Data is sorted in ascending order by Resistance_Cut-Off upon the first click of the Resistance_Cut-Off column sorting button.
12	Data is sorted in descending order by Resistance_Cut-Off upon the second click of the Resistance_Cut-Off column sorting button.
13	Data relevant to the selected COSMIC_ID in the Cosmic ID filter checkbox is extracted.
14	Data relevant to the selected CCLE_Name in the Cell Line Name filter checkbox is extracted.
15	Data relevant to the selected DRUG_ID in the Drug ID filter checkbox is extracted.
16	Data relevant to the selected DRUG_NAME in the Drug Name filter checkbox is extracted.
17	When multiple checkboxes in the dropdown for different filter buttons are selected, data matching all the filter conditions will be extracted.

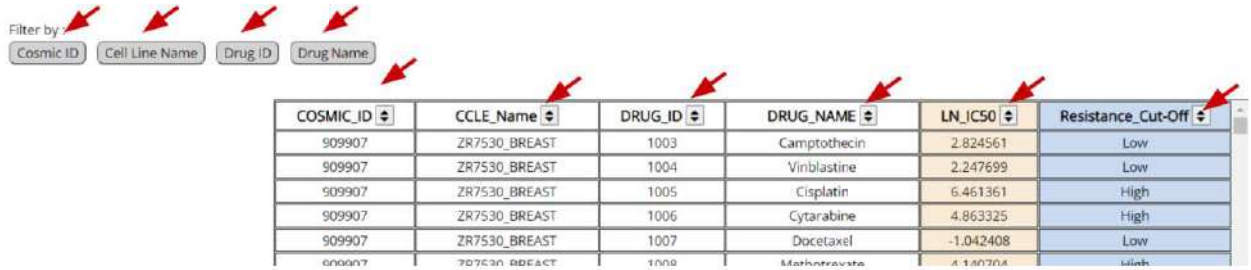
Table 2: Table of Summarisation of different cases for Table Sorting and Filtering Testing

All the cases will be tested with the identical steps.

Step 1: Conduct a physical observation of the code in web.js to ensure that the sorting and filtering functions are implemented correctly.

Step 2: Add intermediate printing commands to print specific messages under the relevant function headers to confirm that the functions are triggered when the relevant buttons are clicked for different cases. For the filter button, print the list of selected checkbox values and the extracted data for easy comparison. Part (d) will show where the commands to print these messages are located. For the sorting buttons, print messages that indicate comparisons in the if-else statements, such as "data1 is smaller than data2", replacing "data1" and "data2" with the actual data values from the specific column. Part (d) will provide detailed locations for these print statements.

Step 3: Click on the buttons relevant to each case shown in **Figure 159**.



COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low

Figure 159: Sorting and Filter buttons available in the Web Application

Step 4: Observe the printed output in the terminal.

Step 5: For filter test, observe how the table updates after clicking the button. Check if the number of rows in the table is reduced after using the filter button, and check if only the data relative to the selected values are displayed. Additionally, for the sorting test, verify that in ascending order, the upper block of values is smaller than the lower block, and in descending order, the upper block of values is greater than the lower block.

Throughout the testing process, document any unexpected outputs resulting from the sorting or filtering actions not working properly. This documentation will facilitate future maintenance and debugging. Ensuring these interactive features work correctly is crucial for enhancing the user experience of the web application.

(c) Inputs to the code or part of code being tested

For **Case 1 and Case 2**, the sort button beside the header of the COSMIC_ID column will be clicked to sort the data based on the value in **COSMIC_ID**. Upon executing the sort action for the COSMIC_ID column, the function "**queue**" in the file "**web.js**" shown in **Figure 160** below will be executed.

```
634  var isAscending = 1;
635  //sort function for column1
636  function queue(){
637    isAscending+=1;
638    var table = document.getElementById("drug-table");
639    if (!table) {
640        console.error("Table not found.");
641        return;
642    }
643
```

Figure 160: Code Segment of "queue" function in "web.js"

For **Case 3 and Case 4**, the sort button beside the header of the CCLE_Name column will be clicked to sort the data based on the value in **CCLE_Name**. Upon executing the sort action for the CCLE_Name column, the function "**queue1**" in the file "**web.js**" shown in **Figure 161** below will be executed.

```
674  var isAscending1 = 1;
675  //sort function for column2
676  function queue1(){
677    isAscending1+=1;
678    var table = document.getElementById("drug-table");
679    if (!table) {
680        console.error("Table not found.");
681        return;
682    }
683
```

Figure 161: Code Segment of "queue1" function in "web.js"

For **Case 5 and Case 6**, the sort button beside the header of the DRUG_ID column will be clicked to sort the data based on the value in **DRUG_ID**. Upon executing the sort action for the DRUG_ID column, the function "**queue2**" in the file "**web.js**" shown in **Figure 162** below will be executed.

```
714  var isAscending2 = 1;
715  //sort function for column3
716  function queue2(){
717    isAscending2+=1;
718    var table = document.getElementById("drug-table");
719    if (!table) {
720        console.error("Table not found.");
721        return;
722    }
723
```

Figure 162: Code Segment of "queue2" function in "web.js"

For **Case 7 and Case 8**, the sort button beside the header of the DRUG_NAME column will be clicked to sort the data based on the value in **DRUG_NAME**. Upon executing the sort action for the DRUG_NAME column, the function "**queue3**" in the file "**web.js**" shown in **Figure 163** below will be executed.

```
753  var isAscending3 = 1;
754  //sort function for column4
755  function queue3(){
756    isAscending3+=1;
757    var table = document.getElementById("drug-table");
758    if (!table) {
759        console.error("Table not found.");
760        return;
761    }
762
```

Figure 163: Code Segment of "queue3" function in "web.js"

For **Case 9** and **Case 10**, the sort button beside the header of the LN_IC50 column will be clicked to sort the data based on the value in **LN_IC50**. Upon executing the sort action for the LN_IC50 column, the function "**queue4**" in the file "**web.js**" shown in **Figure 164** below will be executed.

```
792  var isAscending4 = 1;
793
794  function queue4(){
795    isAscending4+=1;
796    var table = document.getElementById("drug-table");
797    if (!table) {
798        console.error("Table not found.");
799        return;
800    }
801
```

Figure 164: Code Segment of "queue4" function in "web.js"

For **Case 11 and Case 12**, the sort button beside the header of the Resistance_Cut-Off column will be clicked to sort the data based on the value in **Resistance_Cut-Off**. Upon executing the sort action for the Resistance_Cut-Off column, the function "**queue5**" in the file "**web.js**" shown in **Figure 165** below will be executed.

```
831  //sort function for column5
832  var isAscending5 = 1;
833
834  function queue5(){
835    isAscending5+=1;
836    var table = document.getElementById("drug-table");
837    if (!table) {
838        console.error("Table not found.");
839        return;
840    }
841
```

Figure 165: Code Segment of "queue5" function in "web.js"

For **Case 13**, when the application is launched, the "Cosmic ID" filter button will be clicked to filter the data based on the **COSMIC_ID** column. Upon executing the filter action, the function "**filter**" in the file "**web.js**" shown in **Figure 166** below will be executed.

```
875  function filter(index, element) {
908    //Create Select All Button
909    var label = document.createElement("label");
910    var checkbox = document.createElement("input");
911    checkbox.type = "checkbox";
912    checkbox.value = "Select All";
913    checkbox.id="checkAll"+index;
914    label.appendChild(checkbox);
```

Figure 166: Code Segment of "filter" function in "web.js"

For **Case 14 to Case 17**, the "Cell Line Name" filter for **CCLE_Name** column,"Drug ID" filter for **DRUG_ID** column and "Drug Name" filter for **DRUG_NAME** column will also be clicked and selected accordingly. All the filter action will also be executed via the function "**filter**" in the file "**web.js**".

To serve as an evidence of the testing, the commands below are added (**intermediate output printing** methodology) in different functions in "**web.js**" for inspection purpose:

(Case 1): Line 658: console.log(value1+" is smaller than "+value2) in "queue" function (shown in **Figure 167**)
(Case 2): Line 661: console.log(value2+" is bigger than "+value1) in "queue" function (shown in **Figure 167**)

```
652    rows.sort(function(row1, row2) {
653        var value1 = row1.cells[index].textContent.trim();
654        var value2 = row2.cells[index].textContent.trim();
655        if(isAscending%2 ===0){
656          console.log(value1+" is smaller than "+value2);
657          return value1.localeCompare(value2);
658        }
659        else{
660          console.log(value2+" is smaller than "+value2);
661          return value2.localeCompare(value1);
662        }
```

Figure 167: Commands added for Inspection in"queue" function for Case 1 and Case 2

(Case 3): Line 697: console.log(value1+" is smaller than "+value2) in "queue1" function (shown in **Figure 168**)
(Case 4): Line 701: console.log(value2+" is bigger than "+value1) in "queue1" function (shown in **Figure 168**)

```
692    rows.sort(function(row1, row2) {
693        var value1 = row1.cells[index].textContent.trim();
694        var value2 = row2.cells[index].textContent.trim();
695        if(isAscending1%2 ===0){
696          console.log(value1+" is smaller than "+value2);
697          return value1.localeCompare(value2);
698        }
699        else{
700          console.log(value2+" is smaller than "+value1);
701          return value2.localeCompare(value1);
702        }
```

Figure 168: Commands added for Inspection in"queue1" function for Case 3 and Case 4

(Case 5): Line 776: console.log(value1+" is smaller than "+value2) in "queue2" function (shown in **Figure 169**)
(Case 6): Line 780: console.log(value2+" is bigger than "+value1) in "queue2" function (shown in **Figure 169**)

```
732    rows.sort(function(row1, row2) {
733        var value1 = row1.cells[index].textContent.trim();
734        var value2 = row2.cells[index].textContent.trim();
735        if(isAscending2%2 ===0){
736
737          return value1.localeCompare(value2);
738        }
739        else{
740          return value2.localeCompare(value1);
741        }
```

Figure 169: Commands added for Inspection in"queue2" function for Case 5 and Case 6

(Case 7): Line 815: console.log(value1+" is smaller than "+value2) in "queu3" function (shown in **Figure 170**)
(Case 8): Line 819: console.log(value2+" is bigger than "+value1) in "queue3" function (shown in **Figure 170**)

```
771    rows.sort(function(row1, row2) {
772        var value1 = row1.cells[index].textContent.trim();
773        var value2 = row2.cells[index].textContent.trim();
774        if(isAscending3%2 ===0){
775          console.log(value1+" is smaller than "+value2);
776          return value1.localeCompare(value2);
777        }
778        else{
779          console.log(value2+" is smaller than "+value1);
780          return value2.localeCompare(value1);
781        }
```

Figure 170: Commands added for Inspection in"queue3" function for Case 7 and Case 8

(Case 9): Line 855: console.log(value1+" is smaller than "+value2) in "queue4" function (shown in **Figure 171**)
(Case 10): Line 859: console.log(value2+" is bigger than "+value1) in "queue4" function (shown in **Figure 171**)

```
rows.sort(function(row1, row2) {
    var value1 = row1.cells[index].textContent.trim();
    var value2 = row2.cells[index].textContent.trim();
    if(isAscending4%2 ===0){
      console.log(value1+" is smaller than "+value2);
      return value1.localeCompare(value2);
    }
    else{
      console.log(value2+" is smaller than "+value1);
      return value2.localeCompare(value1);
    }
```

Figure 171: Commands added for Inspection in"queue4" function for Case 9 and Case 10

(Case 11): Line 855: console.log(value1+" is smaller than "+value2) in "queue5" function (shown in **Figure 172**)
(Case 12): Line 859: console.log(value2+" is bigger than "+value1) in "queue5" function (shown in **Figure 172**)

```
rows.sort(function(row1, row2) {
    var value1 = row1.cells[index].textContent.trim();
    var value2 = row2.cells[index].textContent.trim();
    if(isAscending5%2 ===0){
      console.log(value1+" is smaller than "+value2);
      return value1.localeCompare(value2);
    }
    else{
      console.log(value2+" is smaller than "+value1);
      return value2.localeCompare(value1);
    }
```

Figure 172: Commands added for Inspection in"queue5" function for Case 11 and Case 12

(Case 13-17): Line 973: console.log(checkedValues); in "filter" function (shown in **Figure 173**)
(Case 13, 17): Line 985: console.log(row.cells[0].textContent.trim()) in "filter" function (shown in **Figure 173**)
(Case 14): Line 991: console.log(row.cells[1].textContent.trim()) in "filter" function (shown in **Figure 173**)
(Case 15, 17): Line 996: console.log(row.cells[2].textContent.trim()) in "filter" function (shown in **Figure 173**)
(Case 16): Line 1001: console.log(row.cells[3].textContent.trim()) in "filter" function (shown in **Figure 173**)

```
if(checkedValues[0].length>0 || checkedValues[1].length>0 ||checkedValues[2].length>0 ||checkedValues[3].length>0){

  console.log(checkedValues)
  for(var k = 0; k<rows.length; k++){


    var row = rows[k];
    var bool = false;
    var bool1 = false;
    var bool2=false;
    var bool3=false;

    if (checkedValues[0].length===0 || checkedValues[0].includes(row.cells[0].textContent.trim())){
      bool = true
      console.log(row.cells[0].textContent.trim())

    }
    if (checkedValues[1].length===0 || checkedValues[1].includes(row.cells[1].textContent.trim())){
      bool1 = true

      console.log(row.cells[1].textContent.trim())
    }
    if (checkedValues[2].length===0 || checkedValues[2].includes(row.cells[2].textContent.trim())){
      bool2 = true

      console.log(row.cells[2].textContent.trim())
    }
    if (checkedValues[3].length===0 || checkedValues[3].includes(row.cells[3].textContent.trim())){
      bool3 = true

      console.log(row.cells[3].textContent.trim())
```

Figure 173: Commands added for Inspection in"filter" function for Case 13 to Case 17

Upon **physical observation**, for **Case 1 and Case 2**, within the queue function, a code segment shown in **Figure 174** is identified spanning **lines 634 to 662** which executes sorting actions for the column "COSMIC_ID", toggling between ascending and descending order based on the "isAscending" variable. When "isAscending" is an even number, sorting is performed in ascending order, while an odd number triggers sorting in descending order. Similar code segment is also found in the other queue functions (queue 1 to queue 5), which performs sorting operations for **Case 3 to Case 12**.

```
var isAscending = 1;
//sort function for column1
function queue(){
  isAscending+=1;
  var table = document.getElementById("drug-table");
  if (!table) {
      console.error("Table not found.");
      return;
  }
```

```
var index = 0;


var rows = Array.from(table.rows);
rows.shift();


rows.sort(function(row1, row2) {
    var value1 = row1.cells[index].textContent.trim();
    var value2 = row2.cells[index].textContent.trim();
    if(isAscending%2 ===0){
      console.log(value1+" is smaller than "+value2);
      return value1.localeCompare(value2);
    }
    else{
      console.log(value2+" is smaller than "+value2);
      return value2.localeCompare(value1);
    }
```

Figure 174: Code Segment responsible for Sorting Actions for different queue functions

For **Case 13 to Case 17**, the filter function is responsible for executing the filter action. Due to size limitation, the code snippet is not provided here, but it is accessible in our GitHub Repository in "web.js". The concept is that initially, it retrieves a list of unique values for each filter button based on the column to which the filter is applied. Then, it adds an event listener to all the checkboxes and adjusts the visibility and display styles accordingly. Specifically, it sets the visibility style and display style of extracted values to "visible" and "table-row", respectively. Conversely, it sets the visibility style and display style of filtered-out values to "hidden" and "none," respectively. This ensures that only the extracted rows are visible in the table. Hence, it is apparent that our team has devised methods for executing sorting and filtering actions through manual code observations.

(d) Expected Outputs

During the sorting button tests, the expected outcome includes the display of the statement "**value1 is smaller than value2**" for **ascending** cases when value1 is indeed smaller than value2. Conversely, for **descending** cases, the expectation is for the statement "**value2 is bigger than value1**" to be displayed when value2 is indeed larger than value1.

In the filter button tests, the following selections are made: **Cosmic ID = 1290798** for **Case 13**, **Cell Line Name = UACC893_BREAST** for **Case 14**, **Drug ID = 1168** for **Case 15**, and **Drug Name = Daporinad** for **Case 16**. In **Case 17**, **both Cosmic ID = 1290798 and Drug ID = 1168** are selected. The expected outcome is that only the rows containing the selected values will be printed and displayed in the output table.

In the inspection console, the following outputs are expected.
Command: console.log(value1+" is smaller than "+value2) **(case 1, case 3, case 5, case 7, case 9, case 11)**
>> Expected Output: For example: if value1 is 1 and value2 is 2, the output should be "**1 is smaller than 2**"
Command: console.log(value2+" is bigger than "+value1) **(case 2, case 4, case 6, case 8, case 10, case 12)**
>> Expected Output: For example: if value1 is 1 and value2 is 2, the output should be "**2 is bigger than 1**"
Command: console.log(checkedValues);
>> Expected Output: **[[1290798], [], [], []] (case 13)**
>> Expected Output: **[[], [UACC893_BREAST], [], []] (case 14)**
>> Expected Output: **[[], [], [1168], []] (case 15)**
>> Expected Output: **[[], [], [], [Daporinad]](case 16)**
>> Expected Output: **[[1290798],[], [1168], []](case 17)**
Command: console.log(row.cells[0].textContent.trim()) **(case 13)**
>> Expected Output: **1290798**
Command: console.log(row.cells[1].textContent.trim()) **(case 14)**
>> Expected Output: **UACC893_BREAST**
Command: console.log(row.cells[2].textContent.trim()) **(case 15)**
>> Expected Output: **1168**
Command: console.log(row.cells[3].textContent.trim()) **(case 16)**
> Expected Output: **Daporinad**
Command: console.log(row.cells[0].textContent.trim()) **(case 17)**

>> Expected Output: **1290798**

Command: console.log(row.cells[2].textContent.trim()) **(case 17)**

>> Expected Output: **1168**

(e) Actual Outputs and Discussions

Case 1

Upon reviewing the actual output for **Case 1**, as illustrated in **Figure 175**, it is evident that it deviates from the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. However, in this output, it shows that value1 is not smaller than value2 as expected, for example, 908123 is not smaller than 749712. This discrepancy indicates that the COSMIC_ID values are not sorted in ascending order upon the first click of the sort button next to the "COSMIC_ID" header. Additionally, as depicted in **Figure 176**, the larger value of the upper block compared to the lower block suggests that the COSMIC_ID values are not sorted in ascending order, for example 1303900 is not smaller than 749709. Based on the observation above, we can conclude the test for **Case 1** has **FAILED.**

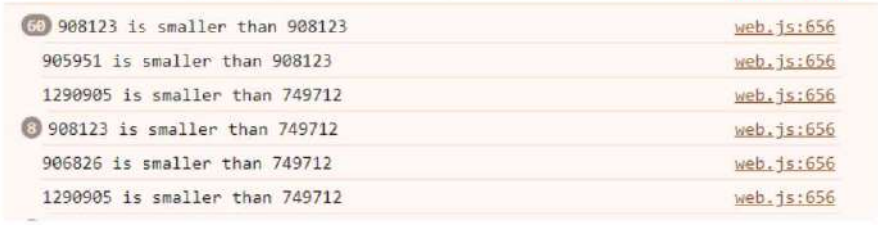


Figure 175: Inspection Console Output for Case 1

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
1240172	MDAMB436_BREAST	1003	Camptothecin	-1.257174	Low
1290798	EFM192A_BREAST	1003	Camptothecin	1.555769	Low
1290905	HCC1428_BREAST	1003	Camptothecin	1.566777	Low
1290922	HDQP1_BREAST	1003	Camptothecin	-1.384751	Low
1298157	JIMT1_BREAST	1003	Camptothecin	-1.127505	Low
1303900	HCC1500_BREAST	1003	Camptothecin	-0.468555	Low
749709	HCC1954_BREAST	1003	Camptothecin	0.317741	Low
749710	HCC1143_BREAST	1003	Camptothecin	0.636184	Low

Figure 176: Table Output for Case 1

Case 2

Upon reviewing the actual output for **Case 2**, as illustrated in **Figure 177**, it is evident that it deviates from the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. However, in this output, it shows that value2 is not bigger than value1 as expected, for example, 1240172 is not bigger than 1290798. This discrepancy indicates that the COSMIC_ID values are not sorted in descending order upon the second click of the sort button next to the "COSMIC_ID" header. Additionally, as depicted in **Figure 178**, the smaller value of the upper block compared to the lower block suggests that the COSMIC_ID values are not sorted in descending order, for example 749709 is not bigger than 1303900. Based on the observation above, we can conclude the test for **Case 2** has **FAILED.**

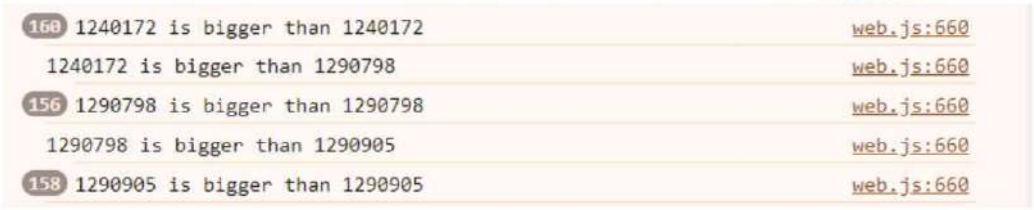


Figure 177: Inspection Console Output for Case 2

749712	HCC1395_BREAST	1003	Camptothecin	-2.255899	Low
749711	HCC1187_BREAST	1003	Camptothecin	1.235544	Low
749710	HCC1143_BREAST	1003	Camptothecin	0.636184	Low
749709	HCC1954_BREAST	1003	Camptothecin	0.317741	Low
1303900	HCC1500_BREAST	1003	Camptothecin	-0.468555	Low
1298157	JIMT1_BREAST	1003	Camptothecin	-1.127505	Low
1290922	HDQP1_BREAST	1003	Camptothecin	-1.384751	Low
1290905	HCC1428_BREAST	1003	Camptothecin	1.566777	Low

Figure 178: Table Output for Case 2

Case 3

Upon reviewing the actual output for **Case 3**, as illustrated in **Figure 179**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. In this output, it shows that value2 is smaller than value1 as expected, for example, CAL120_BREAST is smaller than HCC1395_BREAST. This consistency indicates that the CCLE_Name values are sorted in ascending order upon the first click of the sort button next to the "CCLE_Name " header. Additionally, as depicted in **Figure 180**, the smaller value of the upper block compared to the lower block suggests that the CCLE_Name values are sorted in ascending order, for example AU565_BREAST is smaller than BT20_BREAST. Based on the observation above, we can conclude the test for **Case 3** has **PASSED.**

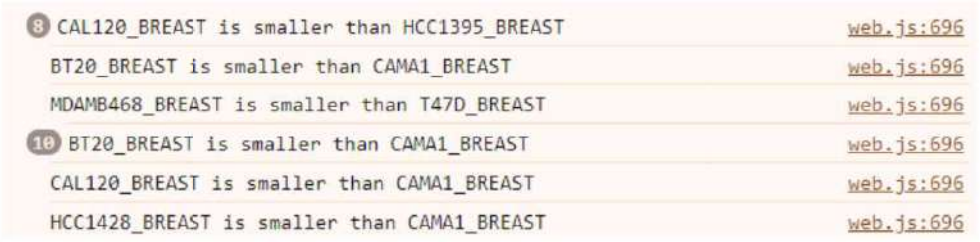


Figure 179: Inspection Console Output for Case 3

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
910704	AU565_BREAST	1003	Camptothecin	-2.834766	Low
906801	BT20_BREAST	1003	Camptothecin	-2.468058	Low
946359	BT474_BREAST	1003	Camptothecin	3.498183	Low
949093	BT483_BREAST	1003	Camptothecin	2.159467	Low
905951	BT549_BREAST	1003	Camptothecin	-0.84884	Low
906826	CAL120_BREAST	1003	Camptothecin	-0.400993	Low
924106	CAL148_BREAST	1003	Camptothecin	-4.452581	Low
910927	CAL51_BREAST	1003	Camptothecin	-3.290208	Low
910852	CAL851_BREAST	1003	Camptothecin	-1.153588	Low

Figure 180: Table Output for Case 3

Case 4

Upon reviewing the actual output for **Case 4**, as illustrated in **Figure 181**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. In this output, it shows that value2 is bigger than value1 as expected, for example, BT20_BREAST is bigger than AU565_BREAST. This consistency indicates that the CCLE_Name values are sorted in descending order upon the second click of the sort button next to the "CCLE_Name " header. Additionally, as depicted in **Figure 182**, the bigger value of the upper block compared to the lower block suggests that the CCLE_Name values are sorted in descending order, for example ZR7530_BREAST is bigger than UACC893_BREAST. Based on the observation above, we can conclude the test for **Case 4** has **PASSED.**

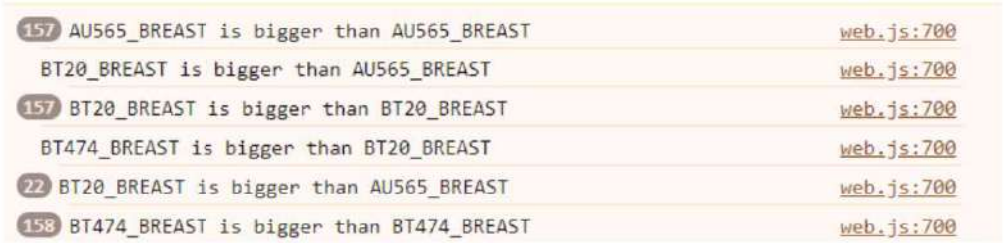


Figure 181: Inspection Console Output for Case 4

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909778	UACC893_BREAST	1003	Camptothecin	-0.382536	Low
910910	UACC812_BREAST	1003	Camptothecin	-2.131015	Low
905945	T47D_BREAST	1003	Camptothecin	-1.334657	Low
908123	MDAMB468_BREAST	1003	Camptothecin	-2.512764	Low
908122	MDAMB453_BREAST	1003	Camptothecin	-1.875994	Low
1240172	MDAMB436_BREAST	1003	Camptothecin	-1.257174	Low
924240	MDAMB415_BREAST	1003	Camptothecin	0.048436	Low

Figure 182: Table Output for Case 4

Case 5

Upon reviewing the actual output for **Case 5**, as illustrated in **Figure 183**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. In this output, it shows that value2 is smaller than value1 as expected, for example, 1003 is smaller than 1004. This consistency indicates that the DRUG_ID values are sorted in ascending order upon the first click of the sort button next to the "DRUG_ID " header. Additionally, as depicted in **Figure 184**, the smaller value of the upper block compared to the lower block suggests that the DRUG_ID values are sorted in ascending order, for example 1003 is smaller than 1004. Based on the observation above, we can conclude the test for **Case 5** has **PASSED.**

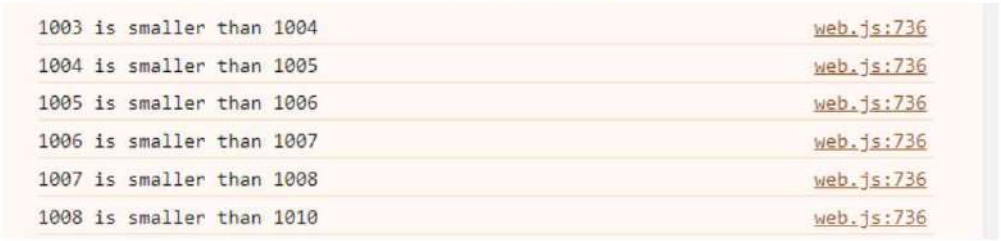


Figure 183: Inspection Console Output for Case 5

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1003	Camptothecin	2.824561	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low
909907	ZR7530_BREAST	1005	Cisplatin	6.461361	High
909907	ZR7530_BREAST	1006	Cytarabine	4.863325	High
909907	ZR7530_BREAST	1007	Docetaxel	-1.042408	Low
909907	ZR7530_BREAST	1008	Methotrexate	4.140704	High
909907	ZR7530_BREAST	1010	Gefitinib	5.263526	High
909907	ZR7530_BREAST	1011	Navitoclax	5.557596	High
909907	ZR7530_BREAST	1012	Vorinostat	3.636889	Low
909907	ZR7530_BREAST	1013	Nilotinib	2.086747	Low

Figure 184: Table Output for Case 5

Case 6

Upon reviewing the actual output for **Case 6**, as illustrated in **Figure 185**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. In this output, it shows that value2 is bigger than value1 as expected, for example, 1004 is bigger than 1003. This consistency indicates that the DRUG_ID values are sorted in descending order upon the second click of the sort button next to the "DRUG_ID" header. Additionally, as depicted in **Figure 186**, the bigger value of the upper block compared to the lower block suggests that the DRUG_ID values are sorted in descending order, for example 2359 is bigger than 2177. Based on the observation above, we can conclude the test for **Case 6** has **PASSED.**

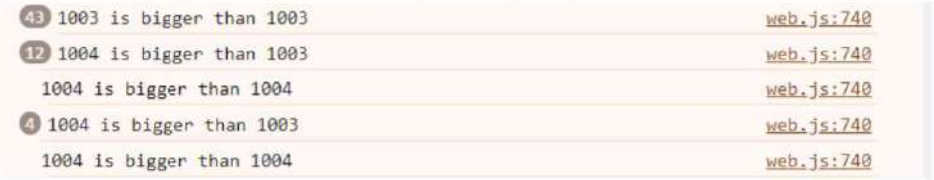


Figure 185: Inspection Console Output for Case 6

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	2359	GSK2830371	6.967681	High
909907	ZR7530_BREAST	2177	SGC0946	3.904823	High
909907	ZR7530_BREAST	2173	PFI-1	6.40038	High
909907	ZR7530_BREAST	2172	JQ1	4.139253	High
909907	ZR7530_BREAST	2169	AZD6482	5.825014	High
909907	ZR7530_BREAST	2111	VE821	7.074833	High
909907	ZR7530_BREAST	2110	GSK591	5.891059	High
909907	ZR7530_BREAST	2107	LJI308	6.73636	High
909907	ZR7530_BREAST	2106	Uprosertib	2.978849	Low

Figure 186: Table Output for Case 6

Case 7

Upon reviewing the actual output for **Case 7**, as illustrated in **Figure 187**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. In this output, it shows that value2 is smaller than value1 as expected, for example, Fulvestrant is smaller than GDC0810. This consistency indicates that the DRUG_NAME values are sorted in ascending order upon the first click of the sort button next to the "DRUG_NAME" header. Additionally, as depicted in **Figure 188**, the smaller value of the upper block compared to the lower block suggests that the DRUG_NAME values are sorted in ascending order, for example ABT737 is smaller than Afatinib. Based on the observation above, we can conclude the test for **Case 7** has **PASSED.**

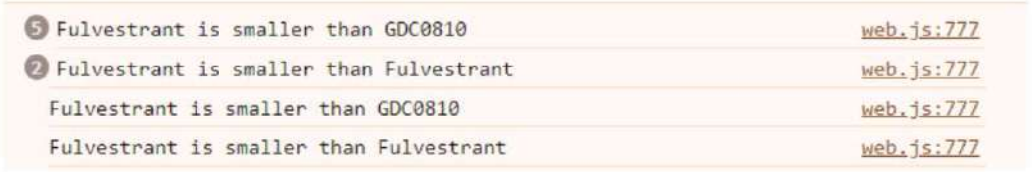


Figure 187: Inspection Console Output for Case 7

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1073	5-Fluorouracil	7.18055	High
909907	ZR7530_BREAST	1910	ABT737	3.740726	Low
909907	ZR7530_BREAST	1032	Afatinib	-1.759182	Low
909907	ZR7530_BREAST	1912	Afuresertib	3.316931	Low
909907	ZR7530_BREAST	1913	AGI-5198	6.404763	High
909907	ZR7530_BREAST	1634	AGI-6780	5.380703	High
909907	ZR7530_BREAST	1051	Alisertib	6.389333	High
909907	ZR7530_BREAST	1560	Alpelisib	2.925281	Low
909907	ZR7530_BREAST	2045	AMG-319	6.504184	High

Figure 188: Table Output for Case 7

Case 8

Upon reviewing the actual output for **Case 8**, as illustrated in **Figure 189**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. In this output, it shows that value2 is bigger than value1 as expected, for example, ABT737 is bigger than 5-Fluorouracil. This consistency indicates that the DRUG_NAME values are sorted in descending order upon the second click of the sort button next to the "DRUG_NAME" header. Additionally, as depicted in **Figure 190**, the bigger value of the upper block compared to the lower block suggests that the DRUG_NAME values are sorted in descending order, for example ZM447439 is bigger than YK-4-279. Based on the observation above, we can conclude the test for **Case 8** has **PASSED.**

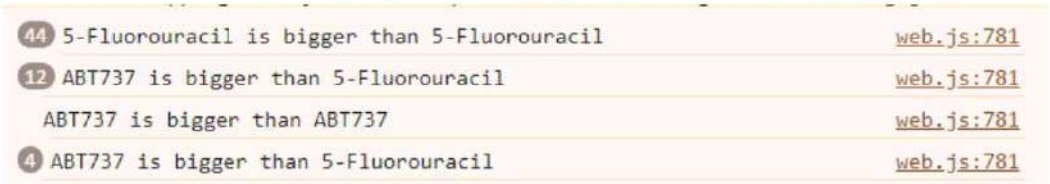


Figure 189: Inspection Console Output for Case 8

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1050	ZM447439	4.513331	High
909907	ZR7530_BREAST	1239	YK-4-279	6.490983	High
909907	ZR7530_BREAST	1614	WZ4003	5.870738	High
909907	ZR7530_BREAST	1622	Wnt-C59	6.460248	High
909907	ZR7530_BREAST	1940	WIKI4	6.836985	High
909907	ZR7530_BREAST	1997	WEHI-539	5.763054	High
909907	ZR7530_BREAST	1046	Wee1 Inhibitor	5.950289	High
909907	ZR7530_BREAST	2096	VX-11e	4.11817	High
909907	ZR7530_BREAST	1012	Vorinostat	3.636889	Low

Figure 190: Table Output for Case 8

Case 9

Upon reviewing the actual output for **Case 9**, as illustrated in **Figure 191**, it is evident that it deviates from the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. However, in this output, there are two different results. One of them shows that value1 is not smaller than value2 as expected, for example, 6.461361 is not

smaller than 2.247699 while one another shows that value1 is smaller than value2 as expected, for example, 2.247699 is smaller than 2.824561. This discrepancy indicates that the LN_IC50 values are not sorted in ascending order upon the first click of the sort button next to the "LN_IC50" header. Additionally, as depicted in **Figure 192**, the larger value of the upper block compared to the lower block suggests that the LN_IC50 values are not sorted in ascending order, for example -0.012102 is not smaller than -0.017448. However, as depicted in **Figure 193,** the smaller value of the upper block compared to the lower block suggests that the LN_IC50 values are sorted in ascending order, for example 3.99445 is smaller than 3.995819. Based on the observation above, we can conclude the test for **Case 9** has **FAILED.**

```
2.247699 is smaller than 2.824561                    web.js:816
6.461361 is smaller than 2.247699                    web.js:816
6.461361 is smaller than 2.824561                    web.js:816
4.863325 is smaller than 2.824561                    web.js:816
4.863325 is smaller than 6.461361                    web.js:816
-1.042408 is smaller than 4.863325                   web.js:816
-1.042408 is smaller than 2.824561                   web.js:816
-1.042408 is smaller than 2.247699                   web.js:816
```

Figure 191: Inspection Console Output for Case 9

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
749710	HCC1143_BREAST	1613	VE-822	-0.012102	Low
905951	BT549_BREAST	1024	Lestaurtinib	-0.017448	Low
906851	EFM19_BREAST	1511	Epirubicin	-0.017664	Low
749716	HCC2218_BREAST	1849	Sabutoclax	-0.022768	Low
908121	MDAMB361_BREAST	1560	Alpelisib	-0.035977	Low
906801	BT20_BREAST	1862	MG-132	-0.041433	Low
906844	DU4475_BREAST	1012	Vorinostat	-0.044249	Low
906844	DU4475_BREAST	1096	Tozasertib	-0.04641	Low

Figure 192: Table Output for Case 9 (FAILED)

749716	HCC2218_BREAST	1177	Niraparib	3.994216	High
946359	BT474_BREAST	1372	Trametinib	3.99445	High
946359	BT474_BREAST	1629	ML323	3.995819	High
749709	HCC1954_BREAST	1133	Serdemetan	3.996208	High
1290798	EFM192A_BREAST	1079	Dasatinib	3.996727	High
910910	UACC812_BREAST	1932	NVP-ADW742	3.996965	High
749709	HCC1954_BREAST	1624	I-BET-762	3.998124	High
749714	HCC1937_BREAST	1632	Ribociclib	3.998447	High
909907	ZR7530_BREAST	1069	EHT-1864	3.998486	High

Figure 193: Table Output for Case 9 (PASSED)

Case 10

Upon reviewing the actual output for **Case 10**, as illustrated in **Figure 194**, it is evident that it deviates from the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. However, in this output, there are two different results. One of them shows that value2 is not bigger than value1 as expected, for example, 2.021195 is not bigger than 5.743307 while another one shows that value 2 is bigger than value 1 as expected, for example, 2.021195 is bigger than 1.944143 . This discrepancy indicates that the LN_IC50 values are not sorted in descending order upon the second click of the sort button next to the "LN_IC50" header. Additionally, as depicted in **Figure 195**, the smaller value of the upper block compared to the lower block suggests that the LN_IC50 values are not sorted in descending order, for example -0.069707 is not bigger than -0.068557. However, as depicted in **Figure 196,** the larger value of the upper block compared to the lower block suggests that the LN_IC50 values are sorted in descending order, for example 9.643361 is bigger than 9.559726. Based on the observation above, we can conclude the test for **Case 10** has **FAILED.**

```
2.021195 is bigger than 5.743307                     web.js:820
2.021195 is bigger than 1.944143                     web.js:820
2.0787 is bigger than 1.174176                       web.js:820
2.0787 is bigger than 5.743307                       web.js:820
2.0787 is bigger than 1.930275                       web.js:820
2.0787 is bigger than 2.021195                       web.js:820
```

Figure 194: Inspection Console Output for Case 10

908120	MDAMB175VII_BREAST	1912	Afuresertib	-0.069707	Low
905960	MDAMB231_BREAST	1060	PD0325901	-0.068557	Low
749717	HCC38_BREAST	1008	Methotrexate	-0.061889	Low
906851	EFM19_BREAST	1930	Telomerase Inhibitor IX	-0.059618	Low
910910	UACC812_BREAST	1919	Osimertinib	-0.059372	Low
908122	MDAMB453_BREAST	2106	Uprosertib	-0.059214	Low
909907	ZR7530_BREAST	1494	SN-38	-0.057708	Low
905945	T47D_BREAST	1561	Taselisib	-0.054402	Low

Figure 195: Table Output for Case 10 (FAILED)

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
949093	BT483_BREAST	1375	Temozolomide	9.643361	High
906826	CAL120_BREAST	1375	Temozolomide	9.559726	High
908120	MDAMB175VII_BREAST	1375	Temozolomide	9.326389	High
1290905	HCC1428_BREAST	1375	Temozolomide	9.025851	High
949093	BT483_BREAST	1025	SB216763	8.910063	High
949093	BT483_BREAST	1239	YK-4-279	8.768315	High
949093	BT483_BREAST	1054	Palbociclib	8.56677	High

Figure 196: Table Output for Case 10 (PASSED)

Case 11

Upon reviewing the actual output for **Case 11**, as illustrated in **Figure 197**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value1 is smaller than value2" was anticipated. In this output, it shows that value1 is smaller than value2 in alphabetical order as expected, for example, High is smaller than Low. This consistency indicates that the Resistance_Cut-Off values are sorted in ascending order upon the first click of the sort button next to the "Resistance_Cut-Off " header. Additionally, as depicted in **Figure 198**, the smaller value of the upper block compared to the lower block suggests that the Resistance_Cut-Off values are sorted in ascending order, for example High is smaller than Low. Based on the observation above, we can conclude the test for **Case 11** has **PASSED.**

```
Low is smaller than Low                    web.js:856
4 High is smaller than Low                 web.js:856
  High is smaller than High                web.js:856
2 Low is smaller than Low                  web.js:856
  High is smaller than Low                 web.js:856
```

Figure 197: Inspection Console Output for Case 11

909907	ZR7530_BREAST	1051	Alisertib	6.389333	High
909907	ZR7530_BREAST	1634	AGI-6780	5.380703	High
909907	ZR7530_BREAST	1913	AGI-5198	6.404763	High
909907	ZR7530_BREAST	1073	5-Fluorouracil	7.18055	High
909907	ZR7530_BREAST	1012	Vorinostat	3.636889	Low
909907	ZR7530_BREAST	2048	Vinorelbine	1.944143	Low
909907	ZR7530_BREAST	1004	Vinblastine	2.247699	Low

Figure 198: Table Output for Case 11

Case 12

Upon reviewing the actual output for **Case 12**, as illustrated in **Figure 199**, it is evident that it shows consistencies with the expected results outlined in part (d), where "value2 is bigger than value1" was anticipated. In this output, it shows that value2 is bigger than value1 as expected in alphabetical order, for example, Low is bigger than High. This consistency indicates that the Resistance_Cut-Off values are sorted in descending order upon the second click of the sort button next to the "Resistance_Cut-Off" header. Additionally, as depicted in **Figure 200**, the bigger value of the upper block compared to the lower block suggests that the Resistance_Cut-Off values are sorted in descending order, for example Low is bigger than High. Based on the observation above, we can conclude the test for **Case 12** has **PASSED.**

```
3130 High is bigger than High              web.js:860
  Low is bigger than High                  web.js:860
3715 Low is bigger than Low                web.js:860
  32 Low is bigger than High               web.js:860
>
```

Figure 199: Inspection Console Output for Case 12

909907	ZR7530_BREAST	1617	AZD5582	2.32519	Low
909907	ZR7530_BREAST	1560	Alpelisib	2.925281	Low
909907	ZR7530_BREAST	1912	Afuresertib	3.316931	Low
909907	ZR7530_BREAST	1032	Afatinib	-1.759182	Low
909907	ZR7530_BREAST	1910	ABT737	3.740726	Low
909907	ZR7530_BREAST	1050	ZM447439	4.513331	High
909907	ZR7530_BREAST	1239	YK-4-279	6.490983	High

Figure 200: Table Output for Case 12

Case 13

Upon examination of the actual output for **Case 13**, depicted in **Figure 201**, it becomes evident that it aligns with the expected result outlined in part (d) which is [[1290798], [], [], []] for the checked values and 1290798 for the content trimmed. This suggests that only the rows with COSMIC_ID values present in the checkedValues list are printed out. Besides, as shown in **Figure 202**, the number of rows of table displayed is reduced, and this can be observed by looking at the scroll element becoming longer. It is shown that the table only displays the data COSMIC_ID is 1290798. Based on the observation above, we can conclude that the test for **Case 13** has **PASSED.**

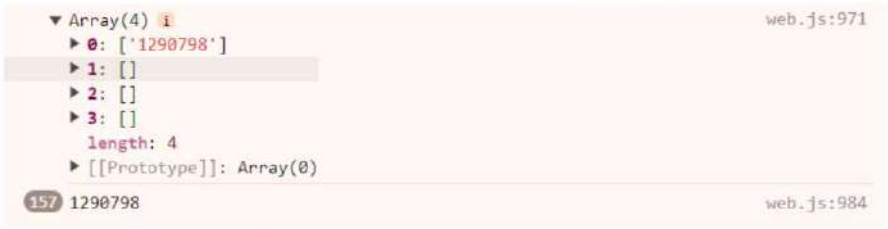


Figure 201: Inspection Console Output for Case 13

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
1290798	EFM192A_BREAST	1003	Camptothecin	1.555769	Low
1290798	EFM192A_BREAST	1004	Vinblastine	1.203306	Low
1290798	EFM192A_BREAST	1005	Cisplatin	5.344505	High
1290798	EFM192A_BREAST	1006	Cytarabine	4.022987	High
1290798	EFM192A_BREAST	1007	Docetaxel	-2.317943	Low
1290798	EFM192A_BREAST	1008	Methotrexate	1.05595	Low
1290798	EFM192A_BREAST	1010	Gefitinib	3.939502	High
1290798	EFM192A_BREAST	1011	Navitoclax	4.198878	High
1290798	EFM192A_BREAST	1012	Vorinostat	1.69382	Low
1290798	EFM192A_BREAST	1013	Nilotinib	3.302964	Low
1290798	EFM192A_BREAST	1014	Refametinib	3.481255	Low
1290798	EFM192A_BREAST	1016	Temsirolimus	0.451474	Low
1290798	EFM192A_BREAST	1017	Olaparib	4.812859	High
1290798	EFM192A_BREAST	1018	Veliparib	5.688972	High
1290798	EFM192A_BREAST	1019	Bosutinib	4.059881	High
1290798	EFM192A_BREAST	1020	Lenalidomide	5.877458	High
1290798	EFM192A_BREAST	1021	Axitinib	5.311053	High
1290798	EFM192A_BREAST	1022	AZD7762	4.102205	High

Figure 202: Table Output for Case 13

Case 14

Upon examination of the actual output for **Case 14**, depicted in **Figure 203**, it becomes evident that it aligns with the expected results outlined in part (d) which is [[], [UACC893_BREAST], [], []] for the checked values and UACC893_BREAST for the content trimmed. This suggests that only the rows with CCLE_Name values present in the checkedValues list are printed out. Besides, as shown in **Figure 204**, the number of rows of table displayed is reduced, and this can be observed by looking at the scroll element becoming longer. It is shown that the table only displays the data of CCLE_Name = UACC893_BREAST. Based on the observation above, we can conclude that the test for **Case 14** has **PASSED.**

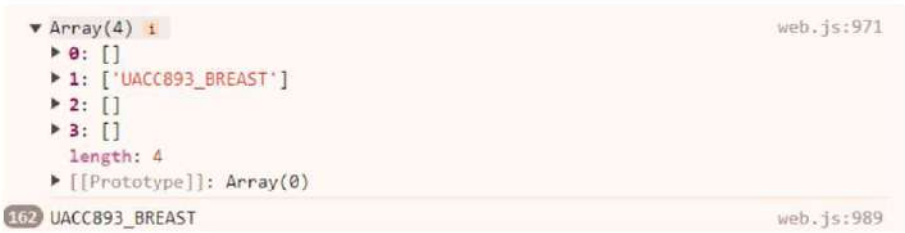


Figure 203: Inspection Console Output for Case 14

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909778	UACC893_BREAST	1003	Camptothecin	-0.382536	Low
909778	UACC893_BREAST	1005	Cisplatin	3.317903	Low
909778	UACC893_BREAST	1007	Docetaxel	-3.855187	Low
909778	UACC893_BREAST	1008	Methotrexate	1.853864	Low
909778	UACC893_BREAST	1009	Tretinoin	4.536733	High
909778	UACC893_BREAST	1010	Gefitinib	2.767898	Low
909778	UACC893_BREAST	1011	Navitoclax	2.411072	Low
909778	UACC893_BREAST	1012	Vorinostat	1.301703	Low
909778	UACC893_BREAST	1013	Nilotinib	2.281483	Low
909778	UACC893_BREAST	1014	Refametinib	0.978738	Low
909778	UACC893_BREAST	1016	Temsirolimus	-0.518789	Low

Figure 204: Table Output for Case 14

Case 15

Upon examination of the actual output for **Case 15**, depicted in **Figure 205**, it becomes evident that it aligns with the expected result outlined in part (d) which is [[], [], [1168], []] for the checked values and 1168 for the content trimmed. This suggests that only the rows with DRUG_ID values present in the checkedValues list are printed out. Besides, as shown in **Figure 206**, the number of rows of table displayed is reduced, and this can be observed by looking at the scroll element becoming longer. It is shown that the table only displays the data DRUG_ID is 1168. Based on the observation above, we can conclude that the test for **Case 15** has **PASSED.**

Figure 205: Inspection Console Output for Case 15

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1168	Erlotinib	4.924159	High
1290798	EFM192A_BREAST	1168	Erlotinib	2.200655	Low
909778	UACC893_BREAST	1168	Erlotinib	1.955808	Low
749717	HCC38_BREAST	1168	Erlotinib	4.330462	High
949093	BT483_BREAST	1168	Erlotinib	5.476614	High
910927	CAL51_BREAST	1168	Erlotinib	2.612853	Low
749714	HCC1937_BREAST	1168	Erlotinib	2.294308	Low
905946	MCF7_BREAST	1168	Erlotinib	4.058467	High
906851	EFM19_BREAST	1168	Erlotinib	3.942961	High
1290922	HDQP1_BREAST	1168	Erlotinib	0.35372	Low
1298157	JIMT1_BREAST	1168	Erlotinib	3.045408	Low
910704	AU565_BREAST	1168	Erlotinib	1.852	Low
910852	CAL851_BREAST	1168	Erlotinib	3.627856	Low

Figure 206: Table Output for Case 15

Case 16

Upon examination of the actual output for **Case 16**, depicted in **Figure 207**, it becomes evident that it aligns with the expected result outlined in part (d) which is [[], [], [], [Daporinad]] for the checked values and Daporinad for the content trimmed. This suggests that only the rows with DRUG_NAME values present in the checkedValues list are printed out. Besides, as shown in **Figure 208**, the number of rows of table displayed is reduced, and this can be observed by looking at the scroll element becoming longer. It is shown that the table only displays the data DRUG_NAME is Daporinad. Based on the observation above, we can conclude that the test for **Case 16** has **PASSED.**

Figure 207: Inspection Console Output for Case 16

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
909907	ZR7530_BREAST	1248	Daporinad	0.894435	Low
909778	UACC893_BREAST	1248	Daporinad	3.272298	Low
749717	HCC38_BREAST	1248	Daporinad	-2.764125	Low
949093	BT483_BREAST	1248	Daporinad	2.268997	Low
906851	EFM19_BREAST	1248	Daporinad	-4.417529	Low
910852	CAL851_BREAST	1248	Daporinad	-4.587479	Low
749713	HCC1599_BREAST	1248	Daporinad	-4.55806	Low
905957	HS578T_BREAST	1248	Daporinad	-4.692851	Low
924240	MDAMB415_BREAST	1248	Daporinad	-3.09046	Low
907048	HCC70_BREAST	1248	Daporinad	-3.758924	Low
910910	UACC812_BREAST	1248	Daporinad	-0.436381	Low
925338	MDAMB157_BREAST	1248	Daporinad	-4.599352	Low
749716	HCC2218_BREAST	1248	Daporinad	-1.683716	Low
924106	CAL148_BREAST	1248	Daporinad	0.587062	Low
905960	MDAMB231_BREAST	1248	Daporinad	-4.508599	Low
1240172	MDAMB436_BREAST	1248	Daporinad	1.091169	Low
946382	CAMA1_BREAST	1248	Daporinad	-3.397561	Low
908120	MDAMB175VII_BREAST	1248	Daporinad	0.909327	Low

Figure 208: Table Output for Case 16

Case 17

Upon examination of the actual output for **Case 17**, depicted in **Figure 209**, it becomes evident that it aligns with the expected result outlined in part (d) which is [[1290798],[], [1168], []], []] for the checked values,1290798 and 1168 for the content trimmed. This suggests that only the rows with COSMIC_ID values and DRUG_ID values present in the checkedValues list are printed out. Besides, as shown in **Figure 210**, the number of rows of table displayed is reduced, and this can be observed by looking at the scroll element becoming longer. It is shown that the table only displays the data **CSSED.**

Figure 209: Inspection Console Output for Case 17

COSMIC_ID	CCLE_Name	DRUG_ID	DRUG_NAME	LN_IC50	Resistance_Cut-Off
1290798	EFM192A_BREAST	1168	Erlotinib	2.200655	Low

Figure 210: Table Output for Case 17

(f) Bugs

A bug has been identified in the sorting function, causing it to fail in several cases, specifically cases 1, 2, 9, and 10. The current sorting function is unable to correctly sort negative values for LN_IC50. Additionally, the sorting function for COSMIC_ID is flawed. It sorts based on the number of digits rather than the numerical value, which leads to incorrect ordering. For example, in case 1, the COSMIC_IDs are sorted as 1240172 < 1303900 < 749709 < 749710, which shows that it sorts ascendingly within groups of identical digit lengths rather than treating the IDs as whole numbers. This results in a misleading sorting order. The sorting function has been left over as a software "bug" because it is just a subcomponent of our application, and our main focus is on the prediction functionality. Our software provides other interactive features, such as search and filter functions, which can also help retrieve drugs or cells of interest with similar capabilities to the sorting function. The primary function of our software is to make predictions. Currently, we are concentrating on refining the model and ensuring critical features, such as CSV file upload and table display, are functioning correctly. We will continue to improve the entire application, including fixing this bug, before the end of the semester.

Section 6.1.4: Visualisation Interactive Functionalities Testing

(a) Test Description (What is being tested)

Given the default dataset with over 6000 rows and potentially larger user-uploaded datasets, displaying all data in a single visualisation can be overwhelming and non-comprehensive. To ensure a user-friendly experience, this test focuses on evaluating the visualisation's interactive functionalities, including filter, tooltips and legends. By implementing a filtering function, the visualisation can reduce clutter by displaying data for each cancer cell type individually, rather than all data at once. This makes the visualisation clearer and more focused. The colour scheme used to represent drug resistance levels is also assessed to ensure it enhances user understanding. The primary objectives of this testing are to **verify that the filtering function correctly filters data based on the selected Cosmic ID**, **confirm the presence and accuracy of legends representing different elements of the visualisation**, and **test the tooltip functionality to ensure it displays correct values relative to each bar**. By achieving these objectives, we aim to ensure that the visualisation is both functional and user-friendly, allowing users to easily interpret and understand the displayed data.

(b) Test Set-Up/ Methodology (How it is being tested)

The test is conducted by uploading a sample dataset, "sample_user_upload_usability_vis_test.csv," to the web application and making predictions. The subsequent tests focus on manually experimenting with and observing the visualisation features of the output produced from this prediction.

Test 1: Testing on Filtering Functionality

We begin by testing the filtering functionality via selecting specific cancer cell types (Cosmic ID) different from the default displayed one, and observe how the visualisation updates to display only the relevant data. This involves checking that the filter correctly applies the selected Cosmic ID. Next, we will experiment by modifying the code responsible for the filter functionality. We will test the logic of the filter by commenting out the relevant parts of the code and observing if any rows are filtered, as well as noting the size or bars of the resulting visualisation.

Test 2: Testing on Interactive Tooltip

Assuming there is no filter function, we examine the tooltip functionality by hovering over different elements of the visualisation. We verify that the tooltips display accurate and relevant information for each data point, such as values of COSMIC_ID or DRUG_ID of specific bars. This step ensures that the tooltips provide users with precise and useful information that enhances their understanding of the data.

Test 3: Evaluating Bar Color Scheme and Sorting Conditions

Additionally, we will test the legends associated with the visualisation to confirm they accurately represent the drug resistance levels. This involves verifying the colour scheme and other visual indicators for clarity and consistency. Lastly, we will check that the bars are sorted correctly, ensuring they are arranged in ascending order from low to high resistance.

(c) Inputs to the code or part of code being tested

Figure 211 illustrates the portion of the "createTable" function code responsible for identifying the header of the input dataset based on its path.

```
394  function createTable(path) {
412      function toTable(text) {
424          headers.forEach(function(header, index) {
425              var trimmedHeader = header.trim();
426              if (trimmedHeader) {
427                  if (index === 0) {
428                      tableHTML += '<th>' + trimmedHeader + '<button id="sortBtn" onclick="queue()"> <i class="fa fa-sort"></i></button></th>';
429                  } else if(index === 1){
430                      tableHTML += '<th>' + trimmedHeader + '<button id="sortBtn1" onclick="queue1()"> <i class="fa fa-sort"></i></button></th>';
431                      cellname = trimmedHeader;
432                  }
433                  else if(index === 2){
434                      tableHTML += '<th>' + trimmedHeader + '<button id="sortBtn2" onclick="queue2()"> <i class="fa fa-sort"></i></button></th>';
435                  }
436                  else if(index === 3){
437                      tableHTML += '<th>' + trimmedHeader + '<button id="sortBtn3" onclick="queue3()"> <i class="fa fa-sort"></i></button></th>';
438                  }
439                  else if(index === 4){
440                      tableHTML += '<th>' + trimmedHeader + '<button id="sortBtn4" onclick="queue4()"> <i class="fa fa-sort"></i></button></th>';
441                      Ic50 = trimmedHeader;
442                  }
```

Figure 211: Code Segment of "createTable" function responsible for retrieving header information

Figure 212 outlines our methodology for retrieving unique COSMIC_ID values to serve as filter options. Given that the dataset may contain duplicate COSMIC_IDs due to different cell-drug pairings (e.g., Cell A paired with Drug A, Drug B, or Drug C in different rows), it is essential to extract only the unique COSMIC_IDs to avoid redundant filter options.

```
449         tableHTML += '</tr></thead><tbody>';
450
451         rows.forEach(function(row) {
452             var trimmedRow = row.trim();
453             if (trimmedRow) {
454
455                 var cols = trimmedRow.split(DELIMITER);
456                 tableHTML += '<tr>';
457                 cols.forEach(function(col, index) {
458                     var trimmedCol = col.trim();
459
460                     if(index ==0){
461
462                         uniqueValues1.add(trimmedCol);
463                     }
464                     tableHTML += '<td>' + trimmedCol + '</td>';
465                 });
466                 tableHTML += '</tr>';
467             }
468         });
```

Figure 212: Code Segment of "createTable" function responsible for identifying unique values of COSMIC_ID

Figure 213, 214 and 215 depicts the code segment handling the filter action by converting the unique values of COSMIC_ID into an array and passing it as options for the filter. The default filter option is set to the first value of COSMIC_ID in the dataset. This setup includes a mechanism to transform the visualisation to display only the data corresponding to the selected COSMIC_ID via the filter.

```
473             var arr1 = Array.from(uniqueValues1);
```

Figure 213: Code Segment of "createTable" function responsible for converting unique values of COSMIC_ID into array

```
530             "params": [
531                 {
532                     "name": "COSMIC_ID",
533                     "value": arr1[0],
534                     "bind": {
535                         "input": "select",
536                         "options": arr1,
537                         "name": "Select Cosmic ID: "
538                     }
539                 }
540             ],
```

Figure 214: Code Segment of "createTable" function responsible for retrieving options for filter

```
488             "transform": [
489                 {"filter": "datum.COSMIC_ID == COSMIC_ID"}
490             ],
```

Figure 215: Code Segment of "createTable" function responsible for defining filter logic

In addition to testing the functionality of the filter, for Test 1, we will comment out the lines of code depicted in **Figure 214** and **Figure 215**. Subsequently, we will observe the changes applied to the visualisation to confirm the correctness of the filter logic.

Figure 216 illustrates the code segment dedicated to constructing the tooltip for each bar in the bar chart. Our objective is to verify whether the value displayed in the tooltip corresponds accurately with the value presented in the bar.

```
521             "tooltip": [
522                 {"field": "DRUG_NAME", "type": "nominal"},
523                 {"field": "DRUG_ID", "type": "nominal"},
524                 {"field": cellname, "type": "nominal"},
525                 {"field": "COSMIC_ID", "type": "nominal"},
526                 {"field": Ic50, "type": "quantitative", "format": ".2f"},
527                 {"field": "Resistance_Cut-Off", "type": "nominal", "title": "Resistance"}
528             ]
```

Figure 216: Code Segment of "createTable" function responsible for retrieving values for Tooltips

Figure 217 depicts the implementation of the colour scheme applied to the bars and the sorting of the bars in ascending order. Our goal is to confirm that the colour scheme accurately represents the drug resistance values and that the bars are appropriately sorted according to the drug resistance value.

```
"encoding": {
  "x": {
    "field": Ic50,
    "type": "quantitative",
    "title": "LN_IC50",
    "axis": {
      "titleFontSize": 18,
      "labelFontSize": 14}
  },
  "y": {
    "field": "DRUG_ID",
    "type": "nominal",
    "title": "Drug ID",
    "sort": {
      "op": "mean",
      "field": Ic50,
      "order": "ascending"
    },
    "axis": {"titleFontSize": 18, "labelFontSize": 14}
  },
  "color": {
    "field": Ic50,
    "type": "quantitative",
    "scale": {
      "domain": [null, 3.77],
      "range": ["darkorange", "gray"],
      "clamp": true // Clamp values outside the domain to the range
    },
    "title": "LN_IC50"
  },
```

Figure 217: Code Segment of "createTable" function responsible for defining colour scheme and sorting for bars

(d) Expected Outputs

Test 1:

Upon selecting a specific COSMIC_ID in the filter, the expected outcome is that **only the bars relevant to the selected COSMIC_ID will be displayed** in the visualisation. Conversely, after the filter parts of the code are removed, it is anticipated that the **visualisation's size will increase**, accommodating more bars representing data from all COSMIC_IDs simultaneously. However, if the COSMIC_IDs have similar drugs being paired, there might be **duplication in the bars, such that one bar is represented by multiple values.**

Test 2:

When hovering over the bars, it is expected that the **tooltip will accurately display the values corresponding to each bar**. The displayed values should align precisely with the representation of the bars and the prediction results, ensuring clarity and coherence in the visualisation.

Test 3:

The anticipated outcome is that the bars will be sorted in **ascending** order, starting from the smallest negative value to the largest positive value. Bars with LN_IC50 **< 3.77 are expected to exhibit a more orange colour**, while those with LN_IC50 **>= 3.77** should display a **more grey colour**.

(e) Actual Outputs and Discussions

Test 1:

The prediction results depicted in **Figure 218** reveal that the uploaded dataset contains two distinct COSMIC_IDs: "749716" and "908123", comprising 9 and 10 rows, respectively. Consequently, the default visualisation, as illustrated in **Figure 219**, will feature the bar chart specific to COSMIC_ID = 749716. Upon applying a filter to the bar chart and selecting the COSMIC_ID = 908123 option, as demonstrated in **Figure 220**, the output visualisation dynamically adjusts to display only the bar chart relevant to COSMIC_ID = 908123, as depicted in **Figure 221**. However, when the filter code segments are commented out, as shown in **Figure 222**, the visualisation expands to include 10 bars, despite COSMIC_ID = 749716 being expected to have only 9 bars. Additionally, some bars exhibit duplicate values, such as DRUG_ID = 1060. **Figures 223** and **224** highlight the inconsistency where the same bar represents different values for different COSMIC_IDs, which contradicts the intended representation of each prediction output value by a distinct bar. Consequently, the test successfully **PASSED** the expected outcomes: the filter function correctly displays bars

corresponding to the selected cell, and without the filter, the visualisation's size increases, but the values represented by each bar become inaccurate due to the display of multiple values within the same bar.

	COSMIC_ID	CELL_LINE_NAME	DRUG_ID	DRUG_NAME	PRED_LN_IC50	Resistance_Cut-Off
0	749716	HCC2218_BREAST	1007	Docetaxel	-2.631801	Low
1	749716	HCC2218_BREAST	1010	Gefitinib	1.740488	Low
2	749716	HCC2218_BREAST	1030	KU-55933	1.975521	Low
3	749716	HCC2218_BREAST	1032	Afatinib	0.759863	Low
4	749716	HCC2218_BREAST	1046	Wee1 Inhibitor	1.911579	Low
5	749716	HCC2218_BREAST	1060	PD0325901	1.806497	Low
6	749716	HCC2218_BREAST	1507	Ruxolitinib	2.303889	Low
7	749716	HCC2218_BREAST	1922	Cediranib	1.690433	Low
8	749716	HCC2218_BREAST	1932	NVP-ADW742	1.522244	Low
9	908123	MDAMB468_BREAST	1007	Docetaxel	-3.145956	Low
10	908123	MDAMB468_BREAST	1010	Gefitinib	3.513914	Low
11	908123	MDAMB468_BREAST	1030	KU-55933	4.015914	High
12	908123	MDAMB468_BREAST	1032	Afatinib	2.305163	Low
13	908123	MDAMB468_BREAST	1046	Wee1 Inhibitor	3.859415	High
14	908123	MDAMB468_BREAST	1060	PD0325901	3.376752	Low
15	908123	MDAMB468_BREAST	1507	Ruxolitinib	4.411232	High
16	908123	MDAMB468_BREAST	1922	Cediranib	3.451726	Low
17	908123	MDAMB468_BREAST	1932	NVP-ADW742	3.201281	Low
18	908123	MDAMB468_BREAST	2359	GSK2830371	4.035609	High

Figure 218: Prediction Result of Uploaded Dataset for Test 1

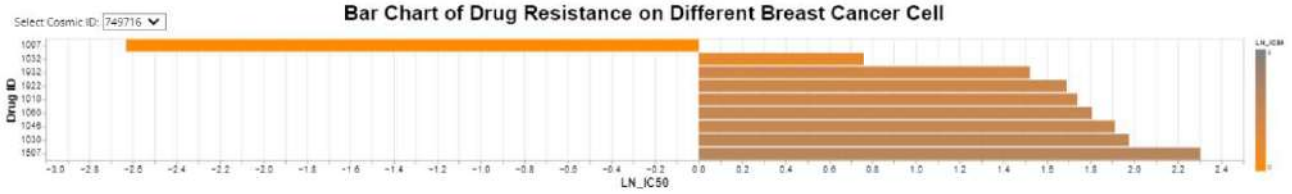


Figure 219: Default Visualisation Displayed for Test 1 of COSMIC_ID = 749716

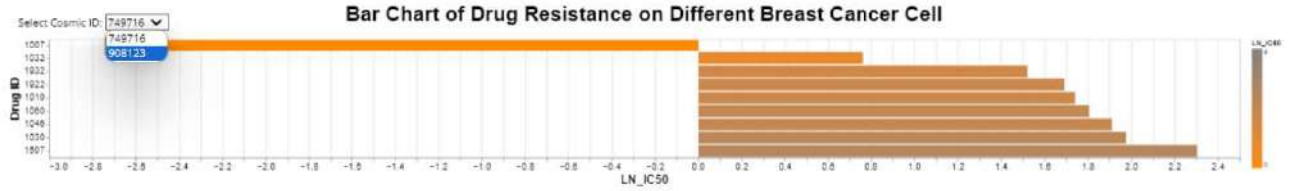


Figure 220: Selection of other Cosmic ID value via filter for Test 1

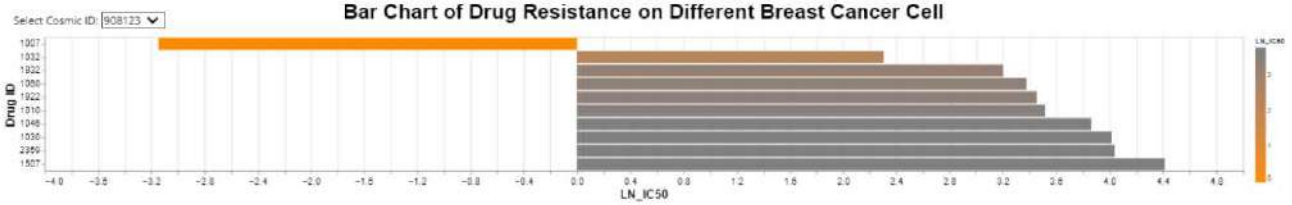


Figure 221: Visualisation Displayed for Test 1 of COSMIC_ID = 908123

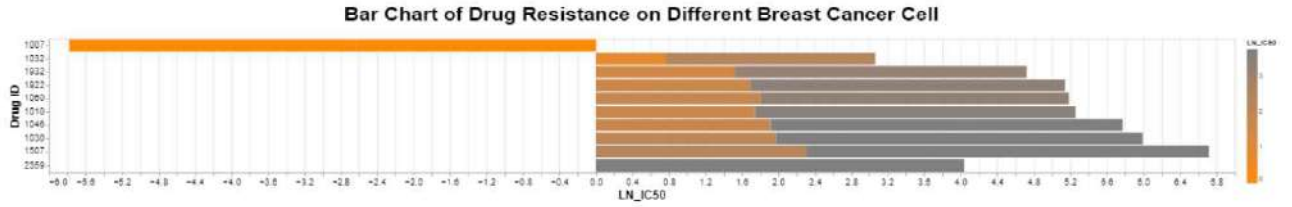


Figure 222: Visualisation Displayed after removing the filter codes

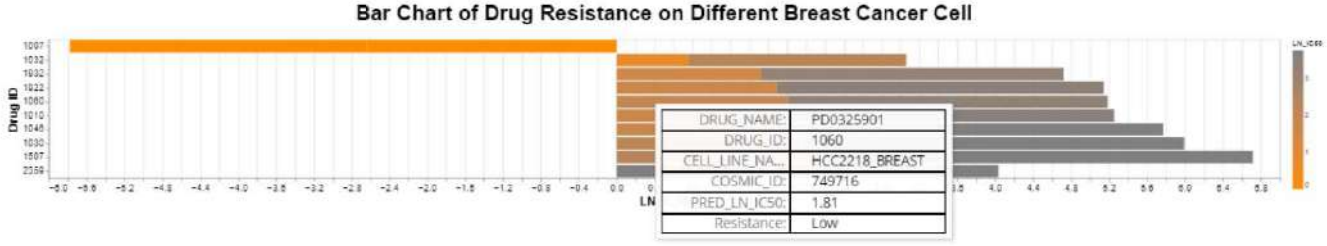


Figure 223: Same Bar, Value for COSMIC_ID = 749168

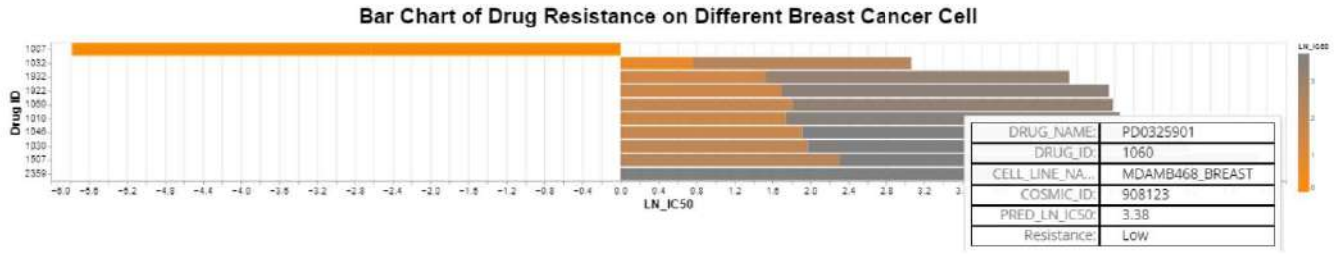


Figure 224: Same Bar, Value for COSMIC_ID = 908123

Test 2 and Test 3:

Given the prediction output for COSMIC_ID = 908123 and DRUG_ID = 1007, as depicted in **Figure 225**, filtering the visualisation to display only the bar chart for COSMIC_ID = 908123 reveals that DRUG_ID = 1007 occupies the first bar. Upon hovering over the first bar, as shown in **Figure 226**, the values displayed in the tooltip precisely align with those in the prediction output table, thereby confirming the successful **PASSED** of Test 2, which verifies the accuracy of the tooltip values. Furthermore, the LN_IC50 value for this cell-drug pair is -3.15, indicating sensitivity to the drug, as it is less than 3.77. In the visualisation, the bar appropriately displays an orange colour, reflecting the sensitivity, and the legend correctly illustrates that bars closer to 0 are more orange. Additionally, scrutiny of the sorting patterns confirms that the bar chart is sorted in ascending order. Consequently, all requirements of Test 3 are satisfactorily **PASSED**.

908123	MDAMB468_BREAST	1007	Docetaxel	-3.145956	Low

Figure 225: Prediction output of COSMIC_ID = 908123 and DRUG_ID = 1007

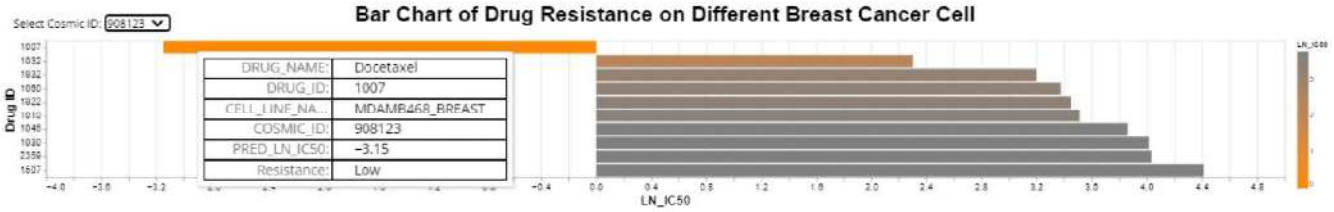


Figure 226: Tooltip Displayed for COSMIC_ID = 908123 and DRUG_ID = 1007

Section 6.2: Third Party Testing

Section 6.2.1: Third Party Testing Set-Up

To ensure the effectiveness and user-friendliness of our web application, we have designed a comprehensive third-party testing protocol. Six participants will be invited to serve as software testers, actively engaging with our application to evaluate its configuration, functionality, and user interface. To facilitate this evaluation, we have meticulously crafted three distinct testing datasets (Set A, Set B, Set C), each containing six CSV files. These files include two valid sample upload CSV files, one file with missing values, and one with incorrect dimensions. Each CSV file comprises two COSMIC_IDs and ten randomly selected drugs, providing diverse testing scenarios. Testers are encouraged to download any set of CSV files from our provided links within the Google Form. Furthermore, a crucial aspect of this testing phase involves assessing the installation, configuration, and operational aspects of our software. Participants will be tasked with ensuring seamless software installation and execution, as errors in these processes may impede user experience. Additionally, we aim to gauge user satisfaction with the prediction performance of our software, emphasising the importance of a smooth and efficient user experience. This testing initiative serves not only to validate the usability of our software but also to identify areas for improvement and enhancement.

Upon the commencement of third-party usability testing, participants will be directed to a designated Google Form, organised into seven sections outlined in **Table 3** below to comprehensively evaluate various aspects of our web application. Each section is tailored to assess different functionalities, including software configuration, user prediction, and button functionalities. Participants will execute a series of tasks outlined in the form and provide feedback on their experiences. The Google Form serves as a centralised platform for collecting valuable insights and suggestions from testers, ensuring thorough evaluation and continuous refinement of our software. Access to the Google Form is provided via this link.

All responses from the participants will be recorded in this spreadsheet. The results recorded will be used for further analysis in Section 6.3.2: Third Party Testing and Review below.

Section	Description
Software Configuration Process	This section provides detailed instructions for users to install all necessary software components, ensuring the successful launch of our web application.
Download Data Samples for User Manual Prediction Testing	Users are directed to download links for test datasets, facilitating manual prediction testing. Clear guidance is provided to ensure users obtain all necessary files for testing purposes.
Software User Guideline (Q1-Q4)	User responses regarding the usability and effectiveness of features outlined in the user guideline are collected, focusing on aspects such as navigation, accessibility, and clarity of instructions.
Software Functionality Test (Q5-Q15)	Users assess the functionality of key buttons and features within the application's main interface. Feedback is gathered on the responsiveness, accuracy, and intuitiveness of these elements.
User Prediction Testing (Q16-Q22)	Participants engage in manual prediction tasks to evaluate the application's predictive capabilities. Feedback is solicited on the ease of use, accuracy, and reliability of the prediction process.
Model Performance Validity (Q23-Q25)	User sentiments regarding the performance and accuracy of the predictive model are documented. This section aims to gauge user confidence in the model's predictions and identify areas for improvement.
Overall Feedback (Q26-Q29)	Users provide comprehensive feedback on their overall experience with the web application, encompassing usability, functionality, and satisfaction. This feedback aids in refining and enhancing the application to better meet user needs and preferences.

Table 3; Table of Summary of Survey Aspects

Section 6.2.2: Third Party Testing Reviews and Responses

Table 4 below summarises participant responses collected from a Google Survey Form regarding the performance of the web application. In total, we gathered responses from six different participants. Here is an overview of the feedback received.

Section	Question	Response	Count of Response
Software Configuration Process	Not applicable	The majority of respondents successfully installed the software, with only one participant experiencing installation difficulties, likely due to issues related to their device system or version.	Not applicable
Download Data Samples for User Manual Prediction Testing	Not applicable	Not applicable	Not applicable
Software User Guideline	Q1. Click on the "Manual User Guideline" button to navigate to the User Guideline Page.	I have done this step. The button can navigate to the User Guideline Page	5
		No	1
	Q2. Please read on the User Guidelines for better understanding of how the software operates.	Yes, I have read and understood	5
		No	1
	Q3. Kindly click the "Download sample template" button to obtain the template for uploading user data for prediction. This ensures alignment with the data format of the user upload dataset.	Yes, I am able to download the sample template	5
		No	1
	Q4. The User Guidelines are comprehensive and easy to understand.	Yes	5
		No	1
Software Functionality Test	Q5. Once the back button is clicked, it will display the default dataset interface.	Yes, I can see the default dataset interface	5
		No	1
	Q6. In the "Default GDSC-CCLE Drug-Cell Line Resistance Dataset" section, feel free to experiment with the filter options. Explore filtering the dataset by various parameters such as COSMIC ID, Cell Line Name, Drug ID, and Drug Name.	Yes, the filter buttons function well	5
		No	1
	Q7. Within the table, you will notice small "up-down" arrows next to the header of each column. These arrows facilitate sorting. Experiment with the sorting function and observe how it alters the displayed data.	Yes, the sorting functionality works perfectly	5
		No	1
	Q8. Within the "**Default GDSC-CCLE Drug-Cell Line Resistance Dataset**" section, select the Export button located at the top-right corner. This action allows you to download the preloaded "GDSC-CCLE Drug-Cell Line Resistance Dataset."	Yes, the export button functions well. I can download the dataset.	5
		NO	1
	Q9. Within the "**Default GDSC-CCLE**	Yes, the dataset is scrollable	5

	Drug-Cell Line Resistance Dataset" section, the dataset displayed is scrollable to ensure all rows of the dataset are visible.	No	1
	Q10. Within the "**Default GDSC-CCLE Drug-Cell Line Resistance Dataset"** section, the dataset is presented in a format that is clear and easily comprehensible. The interactive features are user-friendly and straightforward to utilize.	Yes	5
		No	1
	Q11. Within the "**Default GDSC-CCLE Drug-Cell Resistance Visualisation**" section, the bar chart showing drug resistance on different breast cancer cells is well displayed.	Yes, I can see the barchart	5
		No	1
	Q12. Within the "**Default GDSC-CCLE Drug-Cell Resistance Visualisation**" section, experiment with the filters located at the top left corner. This will enable you to visualise the resistance of different drugs on various types of breast cancer cells.	Yes, the filter functions well for different breast cancer cells	5
		No	1
	Q13. Within the "**Default GDSC-CCLE Drug-Cell Resistance Visualisation**" section, as you hover over the bars in the bar chart, tooltips will appear, providing information for each bar.	Yes, the tooltip is provided and easy to understand	5
		No	1
	Q14. Within the "**Default GDSC-CCLE Drug-Cell Resistance Visualisation**" section, a clear and understandable legend is provided to represent the colour coding of the bar chart.	Yes, I can understand the colour coding from the legend	5
		No	1
	Q15. Overall, the bar chart is comprehensive and understandable. Users can gain insights of drug-cell relationship from the visualisation.	Yes	5
		No	1
User Prediction Testing	Q16. Please click on the "User Manual Prediction" button to direct to the User Manual Prediction Interface.	Yes, the buttons function well, I can see the User Manual Prediction interface	5
		No	1
	Q17. There is a section for uploading data for prediction. Click on the browse file button. Upload the "sample_user_upload_data_with_NA.csv" . The uploaded filename will be displayed at the right side.	Yes, the file can be uploaded	5
		No	1
	Q18. <u>Test 1: Input Data with NA values</u> Click on the "Make Prediction Button". There will be a message showing "**Prediction completed, Rows with NA values have been dropped**". Select all that are applicable:	Yes, the "Make Prediction" button functions well	5
		Yes, the NA value cases have been handled correctly.	5
		Yes, a message is printed upon successful prediction	5
		No	1
	Q19. Upon Successful prediction, the prediction result is displayed in tabular	Yes, the prediction is successfully made. A table of results is displayed,	5

	format at the section "**Prediction Result**".	and is relevant to the input dataset.	
		No	1
	Q20. Upon Successful prediction, the prediction result is displayed in bar chart visualisation format at the section "**Prediction Result**".	Yes, the visualisation is successfully displayed. A bar chart of relationship between drug-cell pairs are shown, and are relevant to the input dataset.	5
		No	1
	Q21. Click on the "Clear Prediction" button, the prediction results are discarded.	Yes, clicking the "Clear Prediction" button clears the interface	5
		No	1
	Q22. Test 2: Wrong Data Dimension/ Incorrect Data Format Testing Upload the "**sample_user_upload_data_wrong_dim.csv**" .Click on the "Make Prediction Button". There will be an error message showing "Error during prediction: Wrong Dimension, Please Input Correct Dimensional Data". There will be no prediction result table and visualisation displayed in this case.	Yes, the error message is displayed. The data with the wrong dimension case is handled in a well manner.	5
		No	1
	Q23. Test 3: Performance Evaluation 1 Upload the "**sample_user_upload_1.csv**" and click on the make prediction button. Based on the prediction result, by observing the "Resistance Cut-Off", compare the predicted "Resistance Cut-Off" with the actual "Resistance Cut-Off" provided in "**y_test_1.csv**" Number of correctly classified Resistance Cut-Off: ____ Number of misclassification: ____ ** Your answer should be separated by comma ','	17,3	2
		16,4	0
		15,5	3
		No	1
	Q24. Test 4: Performance Evaluation 2 Repeat the process of step X on "**sample_user_upload_2.csv**" and compare the prediction result on "**y_test_2.csv**" Number of correctly classified Resistance Cut-Off: ____ Number of misclassification: ____ ** Your answer should be separated by comma ','	17,3	1
		16,4	1
		15,5	3
		No	1
	Q25. Are you satisfied with the prediction result?	Yes	3
		decent results but can be further improved	1
		Cannot understand the question too well. I just answer it based on what I understand. Overall, uploading file for prediction is easy	1
		No	1

Overall Feedback	26. Overall Feedback/ Suggestions on Improvements	loading time can be improved, filter can remove unselect all	1
		no because failure at step 4	1
		Use a python virtual environment so that the libraries won't have conflict/override with existing one.	1
		1. Could use a virtual environment for users to install the dependencies. 2. Could use requirements.txt file instead of manually installing it one by one 3. File path can be shorter to avoid OS error to certain users. 4. Instructions can be clearer to tell the users how to check the prediction result, as the users have no prior knowledge on the project. 5. The UI is very good, very pretty and the data display is in a good visualisation. Overall, good job!	1
		The installation is very user friendly, and the instruction is clear enough to guide me complete the installation and test.	1
		very good, the guideline is really clear, and the web looks gorgeous	1
	Q27. Rate on the installation of "DRUGRES" based on your experience from the tasks you have performed previously.	1	1
		2	0
		3	1
		4	1
		5	3
	Q28. Rate on the UI of "DRUGRES" based on your experience from the tasks you have performed previously.	1	1
		2	0
		3	0
		4	0
		5	5
	Q29. Rate on the UX of "DRUGRES" based on your experience from the tasks you have performed previously.	1	1
		2	0
		3	1
		4	0
		5	4

Table 4: Table of Summary of Responses for Third Party Testing

After reviewing feedback from third-party participants, the majority found our software configuration process straightforward and easy to follow. Most reported smooth laptop setup without errors. However, some noted that certain package installations were time-consuming, impacting their user experience. One participant encountered an error while

installing TensorFlow, indicating potential compatibility issues with certain laptops. This case highlights that our configuration process may not be as flexible as desired, as compatibility issues with certain packages can arise. Unfortunately, the unsuccessful installation prevented this participant from providing structured feedback on software performance. Additionally, participants generally found the process of downloading data samples for manual prediction straightforward, thanks to their familiarity with GitHub. Clear instructions facilitated successful sample downloads for most participants. Suggestions for using virtual environments were made to address potential version compatibility issues, offering valuable insights for improving our software configuration process.

Excluding responses that failed due to software configuration issues, the majority of respondents praised the performance and functionality of our user guideline page. They reported seamless navigation to the page upon clicking the "Manual User Guideline" button and successful downloading of the sample dataset template. Participants expressed understanding of the user guideline page and found it instrumental in properly utilising our web application. Furthermore, third-party participants expressed satisfaction with the prediction results and overall performance of our software functionalities. They lauded the interactive features, such as the filter, sort, and export functions, which operated as expected. Additionally, participants commended the accurate display of elements such as bar charts and tables, noting their simplicity and ease of analysis. Some participants specifically appreciated the presence of tooltips in the visualisation, enhancing their understanding of the displayed data. Overall, respondents positively evaluated the UI and layout of our software application.

Upon reviewing the prediction functionalities and performance, all third-party participants reported successful predictions using our web application. They noted smooth upload of CSV files through both methods—clicking the "Browse File" button and dragging and dropping files. After clicking the "Make Prediction" button, they observed successful printing of expected messages based on the uploaded sample data, with proper handling of missing values or other error cases. While most participants expressed satisfaction with the prediction results, one participant suggested room for improvement, citing a desire for higher model accuracy. Nonetheless, overall feedback indicated ease of file uploading for prediction. Thus, while our prediction results are deemed valid, there is potential for further improvement, as indicated by participant feedback.

Upon reviewing the overall performance of our web application, several third-party participants offered suggestions for improvement. These included enhancing loading times, implementing a virtual environment for easier dependency installation, reducing file paths to prevent operational errors, and providing clearer instructions for checking prediction results. However, the majority of the participants commended our user-friendly interface, indicating success in its design. Regarding satisfaction with the installation of "DRUGRES," responses varied. While some rated the process positively based on prior configuration tasks, indicating the difficulties of software configuration was below average, others expressed dissatisfaction, particularly due to the one participant who was unable to install certain packages. Additionally, assessing participant satisfaction with the UI of "DRUGRES" based on their prior tasks, we conclude that the design aligns with public aesthetics, as all participants except those experiencing setup issues rated satisfaction with the UI. Similarly, evaluating participant satisfaction with the UX of "DRUGRES" based on their prior tasks, we determine that the majority found the user experience satisfactory. However, one participant expressed dissatisfaction, suggesting a need for refinement in both the model and web application.

In summary, our web application demonstrates strong overall performance, with all functionalities operating as intended and boasting a user-friendly interface. Despite identifying weaknesses, we are committed to refining our application before the semester concludes.

Section 7: Recommendations and Improvements

Section 7.1: Recommendation for Improvementes

1. Recommendation on Improving Data Format Acceptance and Columns Matching Checks

The current situation of the application is that it accepts only CSV files for data uploads, as shown in Section 4.1. To make the user's experience and flexibility even better, it is advised to introduce new file formats like Excel (. xlsx) and text file. This new feature will give users more alternatives for uploading their data. Besides, the column validation should be carried out in a way that not only the uploaded files have the right number of columns but also they contain the correct columns. This is because the users might have uploaded the same number of dimensions but different content of columns. For example, the input dataset can be having same number of columns, can be passed in for prediction, but definitely not accurate because for instance, starting from column 5 of the user input data should be Gene A | Gene B | Gene C | Gene D | …. , following the sequence of columns of the training dataset and the format template provided in User Guideline. However, user might upload the data in the format of Gene B | Gene A | Gene D | Gene C | …. The misalignment of sequence of training data will result in result bias, because the content of different columns are misidentified.

2. Recommendation on Improving Data Format Validation

Currently, the application lacks validation checks to ensure that all gene expression values are numeric, leading to model failure when non-numeric values are present, as demonstrated in Scenario 3 of Section 4.1. To address this issue, it is recommended to implement pre-processing checks to verify the numeric nature of all gene expression values before model execution. While the model currently rejects non-numeric input data, resulting in failed predictions, this approach negatively impacts user experience. Therefore, incorporating explicit checks for non-numeric values and implementing a mechanism to drop rows containing such values, while notifying users of the action, is suggested. This approach allows predictions to proceed smoothly by removing non-compliant rows, rather than outright rejection of input data. This enhancement ensures a more seamless user experience and robust model functionality.

3. Recommendation for Enhanced Error Message Handling in Data Format Cases

Currently, the application provides generic error messages for issues related to non-numeric values, missing values, and incorrect dimensions, as outlined in Section 4.1. To improve user awareness and facilitate troubleshooting, it is recommended to enhance these error messages for greater specificity. Specifically, for dimension-related issues, such as mismatches between expected and actual column numbers, the error message should highlight the specific columns that differ from the expected configuration, thereby aiding users in identifying discrepancies. Similarly, for cases involving non-numeric or missing values, the error message should include details about the affected rows and columns, or specify that certain rows have been dropped, to inform users of the data preprocessing actions taken. This approach allows users to pinpoint problematic areas in their data uploads and take corrective measures accordingly. Offering comprehensive error messages enables users to efficiently address data issues, particularly considering the substantial size of input data. Without precise error messages, troubleshooting becomes challenging, hindering users' ability to rectify issues and compromising both data quality and predictive model performance.

4. Recommendation for Usability Improvements

The current application demonstrates basic sorting and filtering capabilities, despite there are bugs identified for the sorting buttons, as highlighted in Section 6.1.3. Furthermore, not all filtering functions have been automated, as observed in usability testing. To improve user-friendliness, several enhancements are proposed. Firstly, the sorting function should be refined to effectively handle negative values and other relevant bugs, addressing the current issue with sorting. Additionally, the filtering feature should be augmented to provide greater automation. Specifically, implementing a mechanism where selecting "Select All" automatically selects all available options, and deselects them upon clicking again, akin to Microsoft Excel's filter function, would significantly streamline the user experience. Moreover, to expedite user interactions, it is recommended to automatically close the dropdown box after applying filters. This adjustment would enhance efficiency by eliminating the need for manual closure, facilitating quicker navigation through the interface.These usability enhancements aim to reduce the cognitive load on users by minimising the number of clicks required to perform actions. By improving efficiency and ease of use, the application will become more intuitive and user-friendly, as validated through manual testing.

5. Recommendation for Continuous Model Improvement

While the current model performance meets the required standards, as detailed in Section 4.4, there remains potential to further decrease the RMSE and enhance prediction accuracy. It is recommended to continuously refine and retrain the model using updated and diversified datasets to achieve this objective. The suggestion is supported by the findings of performance validation tests, which indicate room for improvement. Additionally, exploring advanced feature engineering techniques can contribute to model enhancement, leading to more accurate predictions. Continuous improvement of the model is crucial for practical applications and user confidence. Setting a target RMSE below 0.8 is advisable, as even though the current RMSE is below 1, some predictions lack precision.

6. Visualisation Enhancements

To address the current limitations in visualisations due to the great dimensions of gene expressions (>19200 genes), as outlined in the software limitations (Section 1.7), it is essential to introduce more advanced visualisation features. These enhancements should aim to effectively illustrate the relationships between gene expressions and drug responses. For instance, implementing tools to identify and visualise the top 10 genes that significantly influence the model's predictions can be highly beneficial. By incorporating interactive visualisation capabilities, users will have the opportunity to explore data connections in real-time, facilitating a deeper understanding of the information at hand. These improvements will not only make the application more informative but also visually appealing. Users will be empowered to discern both the drug response across different cell lines and the underlying factors contributing to drug sensitivity based on gene expressions in cancer cell lines.

Section 8: Test Limitations

Section 8.1: Limitations of Blackbox Testing Process

1. Data Format Testing Limitation - Absence of Checking for Mismatching Columns

Limitation:

The current testing process lacks validation for mismatching columns in the data format. As explained in Section 7.1, the application does not check whether the columns match the expected structure. For example, while the input data may have the correct number of columns, the order or naming of columns starting from column 5 (e.g., Gene A | Gene B | Gene C | Gene D | ...) may not be verified. Users might upload data with the correct number of columns but in the wrong order (e.g., Gene B | Gene A | Gene D | Gene C | ...).

Impact:

If the columns do not fit the expected structure, it may lead to imprecise predictions and falsification of data by the model, compromising the reliability of the results. Misalignment between gene expression data and the expected format can cause the model to interpret data incorrectly, leading to inaccurate predictions. This, in turn, affects downstream analysis and decision-making, potentially resulting in erroneous conclusions and actions based on flawed predictions.

Unexecuted Tests:

Tests were not conducted to verify whether the columns in the uploaded data match the expected structure. Specifically, the testing process did not confirm if columns beyond a certain point align with the anticipated gene names. As a result, users could upload data with the correct number of columns but incorrect order or naming conventions. Without these tests, potential errors in the data format may go undetected, increasing the risk of inaccurate predictions and unreliable results.

Mitigation:

To address this limitation, additional validation measures should be implemented to compare the uploaded file's column headers against a predefined set of allowable headers before processing the data. Test cases can be designed to intentionally alter the column order and naming conventions to ensure the software detects and reports these discrepancies accurately. This approach will help enhance the robustness of the testing process and ensure the accuracy of predictions. By validating the alignment of columns with expected gene names, the software can provide users with more meaningful insights into their data and improve the overall reliability of predictions.

No test on specific error message – now error message for dimension wrong and non-numeric data are the same, should test for different case, specific error message. Now no new testing is made on specific error message printing yet because the terminal output already show evidence of error specific to two cases, but after this, we can further improve it by printing specific messages, and test on it – not now because terminal shows difference is correct)

2. Lack of Testing on Specific Error Messages

Limitation:

The current application uses a generic error message for different types of data format issues, such as dimension mismatches and non-numeric data errors. This limitation means that the specific nature of the error is not communicated to the user, making it difficult to identify and correct the issue.

Impact:

The absence of precise error messages can lead to user frustration as they may not understand the specific cause of the error. This can result in delays in obtaining predictions, as users will need to spend additional time troubleshooting. The lack of informative error messages reduces the efficiency of the application and can negatively impact the user experience.

Unexecuted Tests:

Tests to verify the display of specific error messages for different data format issues were not conducted. Currently, the application displays the same error message for both dimension mismatches and non-numeric data errors. While the terminal output does provide evidence of the specific nature of the errors, no tests have been performed to ensure that distinct and informative error messages are printed for each case. The primary focus was on ensuring that the terminal output correctly identifies and distinguishes between different types of errors. As the terminal output currently provides sufficient detail for developers to understand the nature of the errors, no additional tests were performed to differentiate the error messages displayed to users.

Mitigation:

To address this limitation, specific error messages should be developed and tested for various data format issues. This can be achieved by deliberately causing dimension mismatches and non-numeric data errors and verifying that the application displays distinct error messages for each scenario. For example, datasets with incorrect dimensions and non-numeric values should be uploaded separately to confirm that the software produces different error messages for each type of error. Implementing these tests will ensure that users receive more informative feedback, making it easier to identify and correct data issues, thereby enhancing the overall user experience and efficiency of the application.

3. Limitation in Simulated Delayed Testing

Limitation:

The current testing process only evaluated the application with a single large file upload. It did not test the scenario where a second large file is uploaded immediately after the first prediction, without clearing the previous prediction. This omission could lead to issues where the software incorrectly processes the input from the first file instead of waiting for the second file to fully upload.

Impact:

If a user uploads a second large file without clearing the previous prediction, the application might still use the data from the first file for predictions. This can result in incorrect predictions being made based on outdated data. If users are not notified that the new file has not been fully uploaded, they might unknowingly rely on inaccurate results, leading to potential misinformed decisions.

Unexecuted Tests:

Tests to simulate delayed uploads and ensure that the application correctly waits for the second file to upload before making a prediction were not conducted. Specifically, the scenario where a user uploads a second large file immediately after the first prediction without clearing the previous prediction was not tested. The reason this test is not undertaken yet is because the application includes a “clear prediction” button intended to mitigate this issue by encouraging users to clear previous predictions before uploading new files. However, this relies on users consistently following this practice, which may not always happen. Also, the test is not conducted because currently, there are still some small bugs on the simulated delay aspects for uploading the dataset, but if the user waits for a few seconds to ensure that the file is fully uploaded, this situation can be mitigated.

Mitigation:

To address this limitation, additional tests should be performed to simulate the upload of a second large file immediately after the first prediction. These tests should ensure that the application waits until the second file is fully uploaded before enabling the "make prediction" button. This will ensure that predictions are always based on the most recently uploaded data. Improving the handling of delayed uploads, as specified in Section 4.2.1, and conducting these additional tests will help prevent inaccurate predictions and enhance the reliability of the application.

Section 8.2: Limitations of Integration Testing Process

1. Absence of System Compatibility Testing

Limitation:

The integration testing was only conducted on a Windows browser system, without verifying the display and integration of components on other systems such as iOS or Linux. This limitation means that the application's functionality might be inconsistent across different operating systems and browsers.

Impact:

The lack of system compatibility testing can potentially lead to poor user experiences on unsupported platforms. Users on iOS, Linux, or even different browsers on Windows may encounter issues that were not identified during testing, such as visual glitches, functionality errors, or slow performance. This inconsistency can undermine user trust and reduce the overall usability of the web application. The greatest concern is that the most important features, such as the prediction output, predictive model, or visualisation output, may not be integrated successfully into different servers.

Unexecuted Tests:

Cross-platform and cross-browser testing were not performed due to the limitation that all three team members' laptops are Windows systems. This constraint prevented the team from ensuring that the application performs consistently and reliably across a diverse range of environments.

Mitigation:

To address this limitation, implementing cross-browser and cross-platform testing using tools like BrowserStack or Selenium is essential. These tools allow for automated testing across various operating systems and browser configurations, ensuring broader compatibility. Additionally, setting up a virtual testing environment or using online services that offer different OS and browser combinations for testing could have provided more comprehensive coverage. Alternatively, soliciting help from friends or colleagues with different hardware systems to test the web application would ensure that all features are integrated and functioning well across various platforms. This proactive approach would help identify and resolve compatibility issues early, ensuring a smoother and more consistent user experience across different systems. Currently, we can only guarantee that our software integrates well into the Windows system, but we are uncertain whether the predictive model, output, and visualisations can be smoothly integrated into the web application on other systems. By extending our testing efforts, we can enhance the reliability and user satisfaction of our application across a broader audience.

2. Limitation of Data Preprocessing Integration

Limitation:

The testing process included integration testing for only Morgan fingerprint and normalisation on the user input upload data. Other crucial preprocessing steps, such as dropping rows with non-numeric values for gene expression columns, were not included. This means that our current system does not handle data irregularities beyond the scope of Morgan fingerprinting and normalisation.

Impact:

If the user data contains non-numeric values, the current system directly rejects it, preventing any prediction from being made. This limitation can significantly affect user experience, particularly with large datasets where users may not be aware of the presence of non-numeric values. Users might find it frustrating if their data is rejected without clear feedback or automated correction, especially when they are unsure how to clean their data properly.

Unexecuted Tests:

Testing for additional preprocessing steps like removing rows with non-numeric values or handling NA values was not conducted. However, the issue arises particularly in handling data with non-numeric values because as mentioned in Section 4.1 User Data Format Testing, rows with NA values are dropped, but there is absence of checking on non-numeric values, therefore, also absence in preprocessing the user data to drop the rows with non-numeric values. This means that while the software performs well with perfectly formatted data, it may not be user-friendly or robust when handling real-world, messy datasets. The absence of these preprocessing tests does not affect the software's performance in a controlled environment but might lead to user frustration and lower user satisfaction due to the lack of feedback on data issues.

Mitigation:

To improve user experience and ensure the robustness of the software, additional preprocessing steps should be integrated. This includes checking for non-numeric values, handling NA values, and providing clear feedback to users about data issues. Integration tests should be conducted to verify the correct handling of these cases, ensuring that the

preprocessing step is comprehensive and reliable. These tests could have been carried out by modifying the preprocessing script to include steps for checking and removing rows with non-numeric values, then testing with various datasets that include such anomalies. Furthermore, implementing detailed error messages or suggestions for data correction would enhance user guidance and satisfaction.

3. Limited Scalability Testing

Limitation:

The current testing processes primarily focus on validating the integration of prediction output and visualisation for specific scenarios and datasets. However, they may not adequately address scalability concerns as the user database expands and the volume of data increases with the number of predictions made. As more files are stored in the user upload data folder and the results folder, scalability becomes critical.

Impact:

As the application's user base grows and more predictions are made, the lack of scalability testing could lead to performance issues, such as slow processing speeds and delays in visualisations. These issues can result in suboptimal user experience, affecting user satisfaction and potentially leading to user abandonment of the application. Furthermore, incorrect file path readings due to scalability issues could result in displaying incorrect prediction output, undermining the application's reliability and integrity.

Unexecuted Tests:

Comprehensive scalability testing to evaluate the application's performance under various load conditions, including handling large datasets, has not been conducted. Additionally, specific tests to validate the accuracy of file path readings and the integration of prediction output corresponding to the input files have not been performed extensively. While some testing has been conducted on a small scale, it may not fully capture the potential scalability challenges and their impact on the application.

Mitigation:

To address the limitation of limited scalability testing, it is essential to incorporate comprehensive scalability testing into the testing process. This includes testing the application's performance under varying load conditions, and using large dataset sizes. Additionally, implementing automated testing procedures that iteratively make predictions and monitor the application's performance can help identify scalability issues early in the development process. Furthermore, ensuring robust error handling mechanisms to handle incorrect file path readings and other scalability-related issues is crucial for maintaining the application's reliability and user experience. Regular monitoring and optimization of the application's performance can also help mitigate scalability challenges as the user database continues to grow.

Section 8.3: Limitations of Usability Testing Process

1. Limited Number and Diversity of Response Collected

Limitation:

The current testing process only collects responses from 6 participants. This small sample size may not adequately capture the diverse perspectives needed to thoroughly evaluate the web application. Consequently, the limited feedback may not identify all potential bugs and issues, and it does not effectively represent the broader public opinion.

Impact:

Due to the varying perspectives of different individuals, the small number of responses may fail to uncover all the weaknesses of the web application. This limitation can hinder the refining and maintenance processes, as undiscovered bugs may persist, adversely affecting the user experience even after updates. Additionally, the limited feedback could increase the time and cost associated with ongoing refinement and maintenance. Notably, relevant opinions from specific groups, such as Bachelor of Bioscience students, were not included.

Unexecuted Tests:

The limited number of responses is primarily due to participants' busy schedules and the time-consuming setup required to run the application, which includes pre-installing tools like VS Code and Python. This setup requirement makes it impractical to engage random students on campus as software testers.

Mitigation:

To address this limitation, efforts should be made to collect more responses both within and outside the university to ensure a diverse range of feedback. The goal is to gather additional responses before the end of the semester to refine the application effectively. Moreover, keeping the Google form open for ongoing feedback will allow continuous improvement of the application as it evolves.

2. Absence of Feedback on System Usage

Limitation:

In the third-party testing process, there was no inquiry about the users' operating systems (iOS, Windows, etc.). This omission limits our understanding of how the software performs across different systems, which is crucial for effective integration testing because this is the most effective way to understand our web application's performance on different systems.

Impact:

Without feedback on the operating systems used by testers, we cannot fully assess the software's compatibility and performance on various platforms. This gap may result in undiscovered issues related to system-specific functionality, potentially affecting user experience across different devices. Consequently, the software might encounter unexpected bugs or performance issues when used on unsupported or less-tested systems.

Unexecuted Tests:

The lack of system usage feedback means that no specific tests were conducted to evaluate the software's performance on different operating systems. This oversight leaves a critical aspect of integration testing unaddressed, as we did not systematically verify cross-platform compatibility.

Mitigation:

To mitigate this limitation, future testing phases should include questions about the testers' operating systems, or the browser they used. This information will help identify and address any system-specific issues. Additionally, targeted testing should be conducted on various operating systems to ensure comprehensive integration and compatibility. Keeping an open channel for continuous feedback, including system usage data, will further enhance the software's adaptability and user experience across different platforms.

Section 9: Appendix

Section 9.1: CCLE Gene Expression Dataset Analysis

[Appendix of Data Source 2: Cancer Cell Line Encyclopedia (CCLE)]

Data Source	Cancer Cell Line Encyclopedia (CCLE): ➢ DepMap Public 22Q2
Data Origin	https://depmap.org/portal/download/all/?releasename=DepMap+Public+22Q2&filename=CCLE_expression.csv
Data Authorisation	Broad Institute, Merkin Building, 415 Main St., Cambridge, MA 02142
Data Format	CSV file
Data Shape	1406 rows 19222 columns
Number of Genes (columns)	19221
Number of Cancer Cell Lines (rows)	1406
Primary Diseases	33
Lineages	30

Table 5: Table of Summarisation of CCLE Gene Expression Dataset (22Q2)

Section 9.2: CCLE Sample Info Dataset Analysis

[Appendix of Data Source 2: Cancer Cell Line Encyclopedia (CCLE)]

Data Source	Cancer Cell Line Encyclopedia (CCLE): ➢ Sample_info ➢ It is a data collected set of every cancer model/cell line of DepMap. It gives a complete description of each column found in the DepMap Release README file.	
Data Origin	https://depmap.org/portal/download/all/?releasename=DepMap+Public+22Q2&filename=sample_info.csv	
Data Authorisation	Broad Institute, Merkin Building, 415 Main St., Cambridge, MA 02142	
Data Format	csv file	
Data Shape	1840 rows 29 columns	
Number of Cell Lines	1840	
Primary Diseases	33	
Lineages	30	
Columns Representation and Description	DepMap_ID	DepMap assigns a static primary key to each cell line.
	cell_line_name	Original name of the cell line, containing punctuation.
	stripped_cell_line_name	AL nomenclature of a cell line.
	CCLE_Name	Old naming convention of having the naked cell line name followed by the lineage, not to be given to new cell lines.
	alias	Other cell line identifiers (not an exhaustive list)
	COSMICID	Cosmic cancer database cell line ID.
	sex	Gender of tissue donor if available

	source	Supplier of cell line vail used by DepMap.
	RRID	Cellosaurus research resource identifier
	WTSI_Master_Cell_ID	Sanger Drug dataset record ID.
	sample_collection_site	Tissue collection site
	primary_or_metasis	Determines if tissue specimen is from the primary or the metastatic site.
	primary_disease	General cancer lineage category
	Subtype	name of the disease.
	age	If available, the age of the tissue donor at the time of sampling.
	Sanger_Model_ID	Sanger Institute CM Passport ID
	depmap_public_comments	Additional information about the cell line.
	lineage	Cancer type classifications in a uniform manner.
	lineage_subtype	Cancer type classifications in a uniform manner.
	lineage_sub_subtype	Cancer type classifications in a uniform manner.
	lineage_molecular_subtype	Cancer type classifications in a uniform manner.
	default_growth_pattern	Normal growth outline of the cell line
	model_manipulation	Cell line alterations including drug resistance and gene knockout.
	model_manipulation_details	More information about the model manipulation
	patient_id	Cell lines sharing the same patient
	patient_depmap_id	If known, the DepMap ID of the parental cell line.
	Cellosaurus_NCIt_disease	Disease term from Cellosaurus, NCI thesaurus
	Cellosaurus_NCIt_id	Cellosaurus, even NCI thesaurus code
	Cellosaurus_issues	Cellosaurus, documentation of cell line problems.

Table 6: Table of Summarisation of CCLE Sample Info (22Q2)

Section 9.3: GDSC Drug Response Dataset Analysis

[Appendix of Data Source 1: Genomics of Drug Sensitivity in Cancer (GDSC)]

Data Source	Genomics of Drug Sensitivity in Cancer (GDSC): ➢ Screening Data: Drug Screening - IC50s: GDSC2-dataset	
Data Origin	https://www.cancerrxgene.org/downloads/bulk_download	
Data Authorisation	Wellcome Sanger Institute, Massachusetts General Hospital Cancer Center	
Data Format	xlsx file	
Data Shape	242036 rows 19 columns	
Number of Cell Lines	969	
Number of Drugs	295	
Columns Representation and Description	DATASET	The name of the dataset that the data belongs to.
	NLME_RESULT_ID	The specific id for nonlinear mixed effect model
	NLME_CURVE	The unique id for nonlinear mixed effect model
	COSMIC_ID	A unique id for the cell line in the COSMIC database. The same cell line can have different cosmic id.
	CELL_LINE_NAME	The name of the cell line used in the project
	SANGER_MODEL	The unique id for a specific drug
	TCGA_DESC	The description of the Cancer Genome Atlas
	DRUG_ID	The unique id for a specific drug
	DRUG_NAME	The name of the drug used in the experiment
	PUTATIVE_TARGET	A potential target of a substance or drug
	PATHWAY_NAME	A sequence of molecular interactions that are involved in the chemical process
	COMPANY	The unique id for a company.
	WEBRELEASE	To identify if the data is available to public
	MIN_CONC	The minimum concentration of drug used on the cell line
	MAX_CONC	The maximum concentration of drug used on the cell line
	LN_IC50	The logarithm of IC50, representing sensitivity of every cancer cell line to the drugs(Li et al., 2021). The lower the LN_IC50, the higher the sensitivity of the cell line to the drug, the lower the drug resistance to that particular cancer cell line. (Horikawa et al., 2015) (Yuan et al., 2019).
	AUC	The area under the curve of dose-response.This curve is used to represent the relationship between the drug concentration and its effect, and the AUC can be used to quantify the overall efficacy of a drug. The higher the AUC value, the higher the resistance to the drug (Zhu&Dupuy, 2022).
	RMSE	The root mean square error, representing the measurement of the mean difference between the predicted value and the actual value of the model (Frost, 2023).The RMSE is varied as the different drugs used on the same cell line.
	Z_SCORE	The inverse correlation between gene expression and the LN_IC50 for the cell lines and measures the distance between the data point and the mean of the dataset.

Table 7: Table of Summarisation of GDSC-2 Dataset

Section 9.4: GDSC Drug Info Dataset Analysis

[Appendix of Data Source 1: Genomics of Drug Sensitivity in Cancer (GDSC)]

Data Source	Genomics of Drug Sensitivity in Cancer (GDSC): ➢ Compounds: Drug_list	
Data Origin	https://www.cancerrxgene.org/compounds	
Data Authorisation	Wellcome Sanger Institute, Massachusetts General Hospital Cancer Center	
Data Format	csv file	
Data Shape	700 rows 9 columns	
Number of Drugs	624	
Columns Representation and Description	Drug Id	The unique id for a specific drug
	Name	The name of the drug
	Synonyms	The alias of drug name
	Targets	A potential target of a substance or drug
	Target pathway	A sequence of molecular interactions that are involved in the chemical process
	PubCHEM	The unique id for a specific drug in PubCHEM database
	Datasets	The name of the dataset that the data belongs to.
	number of cell lines	The number of cell lines are experimented with the specific drug
	Screening site	The screening assays of a specific drug

Table 8: Table of Summarisation of GDSC Drug Info

Section 9.5: PubChem Data Source Analysis

[Appendix of Data Source 3: PubCHEM]

Data Source	PubChem
Data Origin	https://pubchem.ncbi.nlm.nih.gov/
Data Authorisation	National Library of Medicine
Information Provided	118 Million Compounds, 318 Million Substances, 295 Million Bioactivities, 41 Million Literature, 51 Million Patents

Table 9: Table of Summarisation of Potential Data to be Retrieved from PubChem

---

Section 9.6: Predictive Model Training Data Analysis

Data Source	CCLE, GDSC and PubChem
Data	https://drive.google.com/file/d/11r7CFRXcy19SK_8qvyn3aq8EwQFp4qRU/view?usp=sharing
Data Format	csv file
Data Shape	6847 rows, 19482 columns
Focused Cancer Type	Breast Cancer
Number of Cancer Cell Lines	45
Focused Cancer Cell Lines [COSMIC_ID]	909907, 1290798, 909778, 749717, 949093, 910927, 749714, 905946, 906851, 1290906, 1290922, 1298157, 910704, 910852, 749713, 906844, 749715, 749710, 905957, 924240, 907048, 910910, 749711, 925338, 749716, 907045, 924106, 749709, 946359, 905960, 1303900, 1240172, 946382, 908120, 908122, 905945, 907046, 907047, 749712, 1290905, 906801, 906826, 908123, 905951, 908121
Number of Drugs	170
Focused Drugs	1003, 1004, 1005, 1006, 1007, 1008, 1010, 1011, 1012, 1013, 1014, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1029, 1030, 1032, 1033, 1036, 1038, 1042, 1043, 1046, 1047, 1049, 1050, 1051, 1053, 1054, 1057, 1058, 1059, 1060, 1062, 1067, 1068, 1069, 1072, 1073, 1079, 1080, 1083, 1085, 1086, 1088, 1089, 1093, 1096, 1129, 1131, 1133, 1149, 1168, 1175, 1177, 1179, 1180, 1190, 1191, 1192, 1199, 1200, 1237, 1239, 1248, 1249, 1250, 1372, 1373, 1375, 1378, 1494, 1507, 1510, 1511, 1512, 1529, 1549, 1553, 1557, 1558, 1560, 1561, 1563, 1564, 1576, 1578, 1593, 1598, 1613, 1614, 1615, 1617, 1618, 1620, 1621, 1622, 1624, 1626, 1627, 1629, 1630, 1631, 1632, 1634, 1635, 1786, 1799, 1849, 1852, 1853, 1854, 1855, 1862, 1909, 1910, 1912, 1913, 1915, 1917, 1918, 1919, 1922, 1924, 1925, 1926, 1927, 1928, 1930, 1931, 1932, 1933, 1936, 1939, 1940, 1941, 1997, 2040, 2043, 2044, 2045, 2046, 2048, 2096, 2106, 2107, 2110, 2111, 2169, 2172, 2173, 2177, 2359, 1009, 1031, 1037, 1039, 1194, 1243, 1502, 2174, 2175

Table 10: Table of Summarisation of DRUGRES Predictive Model Training Data

---

Section 10: Reference

Section 10.1: References

1. Barretina, J., Caponigro, G., Stransky, N., Venkatesan, K., Margolin, A. A., Kim, S., Wilson, C. J., Lehár, J., Kryukov, G. V., Sonkin, D., Reddy, A., Liu, M., Murray, L., Berger, M. F., Monahan, J. E., Morais, P., Meltzer, J., Korejwa, A., Jané-Valbuena, J., . . . Garraway, L. A. (2012). The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature*, *483*(7391), 603–607. https://doi.org/10.1038/nature11003
2. Li, Y., Umbach, D. M., Krahn, J. M., Shats, I., Li, X., & Li, L. (2021). Predicting tumor response to drugs based on gene-expression biomarkers of sensitivity learned from cancer cell lines. *BMC Genomics*, *22*(1). https://doi.org/10.1186/s12864-021-07581-7
3. Kim, S., Thiessen, P. A., Bolton, E. E., Chen, J., Fu, G., Gindulyte, A., Han, L., He, J., He, S., Shoemaker, B. A., Wang, J., Yu, B., Zhang, J., & Bryant, S. H. (2015). PubChem Substance and Compound databases. Nucleic Acids Research, 44(D1), D1202–D1213. https://doi.org/10.1093/nar/gkv951
4. Shahzad, M., Tahir, M. A., Alhussein, M., Mobin, A., Shams Malick, R. A., & Anwar, M. S. (2023). NeuPD-A Neural Network-Based Approach to Predict Antineoplastic Drug Response. Diagnostics (Basel, Switzerland), 13(12), 2043. https://doi.org/10.3390/diagnostics13122043
5. Sharifi-Noghabi, H., Jahangiri-Tazehkand, S., Smirnov, P., Hon, C., Mammoliti, A., Nair, S. K., Mer, A. S., Ester, M., & Haibe-Kains, B. (2021). Drug sensitivity prediction from cell line-based pharmacogenomics data: guidelines for developing machine learning models. *Briefings in Bioinformatics*, *22*(6). https://doi.org/10.1093/bib/bbab294
6. Tang, Y., Wang, T., Hu, Y., Ji, H., Yan, B., Hu, X., Zeng, Y., Hao, Y., Xue, W., Chen, Z., Lan, J., Wang, Y., Deng, H., Deng, C., Wu, X., & Yan, J. (2023). Cutoff value of IC50 for drug sensitivity in patient-derived tumor organoids in colorectal cancer. *iScience*, *26*(7), 107116. https://doi.org/10.1016/j.isci.2023.107116
7. Yang, W., Soares, J., Greninger, P., Edelman, E. J., Lightfoot, H., Forbes, S., Bindal, N., Beare, D., Smith, J. A., Thompson, I. R., Ramaswamy, S., Futreal, P. A., Haber, D. A., Stratton, M. R., Benes, C., McDermott, U., & Garnett, M. J. (2012). Genomics of Drug Sensitivity in Cancer (GDSC): a resource for therapeutic biomarker discovery in cancer cells. *Nucleic Acids Research*, *41*(D1), D955–D961. https://doi.org/10.1093/nar/gks1111

Section 10.2: Generative AI Declaration

Our group acknowledges the use of ChatGPT (https://chat.openai.com/) to understand different types of software testing and generate ideas on the possible testings that can be conducted on our software. We entered the following prompts from 8th May 2024 to 20th May 2024:

- What is blackbox testing?
- What is integration testing?
- What are the blackbox testing that can be conducted for the software - drug resistance prediction in cancer cell lines?
- What are the integration testing that can be conducted for the software - drug resistance prediction in cancer cell lines?
- Summarisation of paragraphs
- Improvement of academic tone and correction of grammatical errors
- Sentence restructuring
- Elaboration of points given

The output from the chatGPT was referred for study purposes and modified in this report.
