Exploratory Data Analysis (EDA) Whitepaper

Exploratory Data Analysis (EDA) aka visualization toolbox plays a pivotal role in the fields of machine learning and data analysis for several compelling reasons. Firstly, EDA serves as the foundational step in understanding and familiarizing oneself with the dataset at hand. Before diving into complex machine learning models or advanced analytical techniques, it's crucial to have a comprehensive grasp of the data's characteristics. EDA helps practitioners identify data quality issues, such as missing values, outliers, or inconsistencies, enabling them to take necessary preprocessing steps to clean and prepare the data. This initial exploration also aids in feature selection or engineering, as it unveils insights about which variables might be relevant or redundant for the task at hand.

Secondly, EDA is instrumental in pattern discovery and hypothesis generation. Through techniques like data visualization, summary statistics, and correlation analysis, EDA allows practitioners to identify meaningful relationships, trends, or anomalies within the data. This not only helps in understanding the underlying data structure but also guides the selection of appropriate machine learning algorithms and modeling strategies. For instance, EDA might reveal that certain features exhibit strong multicollinearity, prompting the use of dimensionality reduction techniques or regularization methods to improve model performance. In essence, EDA empowers data scientists and analysts to make informed decisions and develop a more profound understanding of their data, which in turn enhances the effectiveness and efficiency of subsequent machine learning or data analysis tasks.

When deploying EDA with Streamlit, its value becomes even more evident. Streamlit is a user-friendly web application framework for Python that allows data scientists to create interactive and shareable data applications quickly. Integrating EDA into a Streamlit app provides numerous advantages. First, it enhances collaboration by enabling team members, stakeholders, or clients to interact with and explore the dataset in a user-friendly manner, even if they lack technical expertise. This fosters better communication and alignment on project objectives and insights. Moreover, a Streamlit-based EDA app makes it easy to document and reproduce data exploration steps, ensuring transparency and reproducibility in data analysis workflows. Additionally, Streamlit's real-time interactivity allows for on-the-fly adjustments, making it effortless to adapt the exploration process based on immediate feedback or evolving research questions. Overall, deploying EDA with Streamlit makes data exploration more accessible, engaging, and impactful, enhancing the overall effectiveness of machine learning and data analysis projects.

5- Technical explanation of EDA

5-1- Usability

In this section, we will cover the various components of the website.

Sidebar Menu: Users can choose from various preloaded datasets.

Exploratory Data Analysis (EDA) Whitepaper

In version 2.0, the covered datasets include Iris, PIMA, Wine, and Health Insurance.

Users also have the option to select an external dataset. This provides them with two options: they can either upload a numeric CSV file or simply paste the URL for a raw CSV file from the internet.

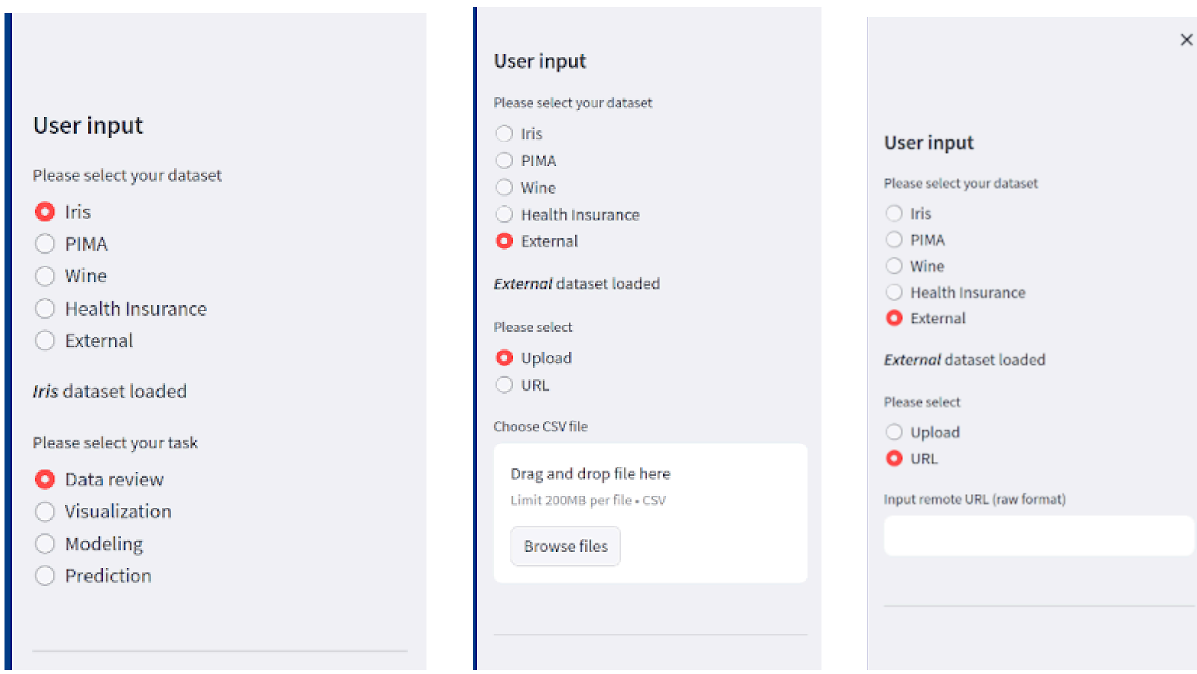


Figure 2: EDA sidebar: selecting or uploading the data

After selecting the dataset, there are four task available. The options are data review, visualization, modeling and prediction.

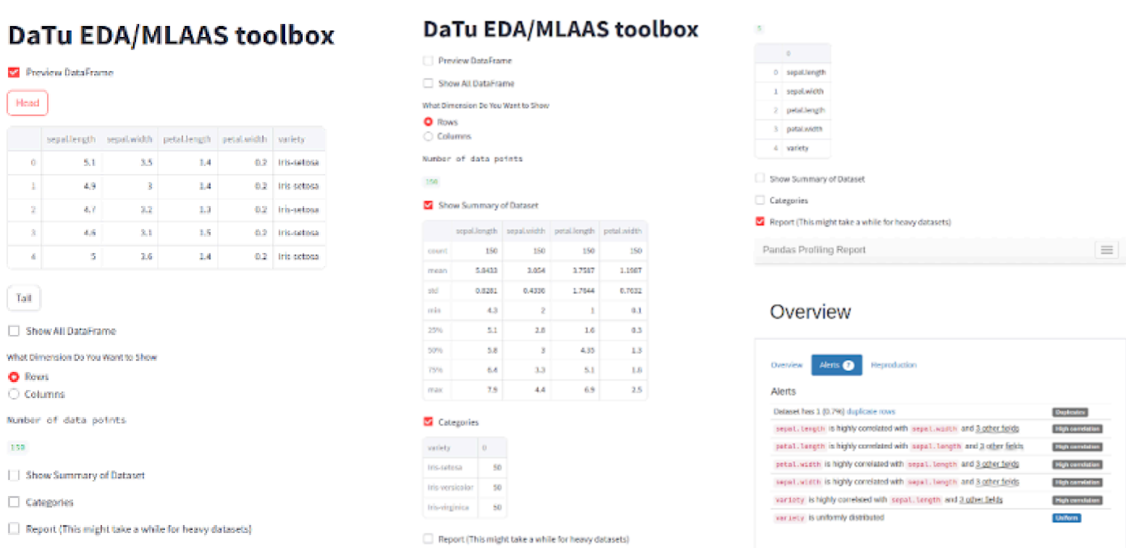


Figure 3: EDA data review: explanatory and panda report

As seen in Figure 3, the Data Review provides a preliminary overview of the data. This includes displaying the head and tail of the dataframe, generating a summary, and utilizing Pandas profiling to gain insights into the dataset, such as correlations, missing values, distributions, duplicates, interactions, and more.

In the second task, Visualization, users will find an additional option in the left sidebar to choose from. They can select a scatter plot for specific classes, which will display histograms and scatter plots for the selected features. Additionally, they have the option to choose group plots, which encompass Matplotlib for individual points (note that this may be time-consuming for large datasets), box plots, correlation plots, and bar plots. Some of these plots are exemplified using the IRIS dataset.

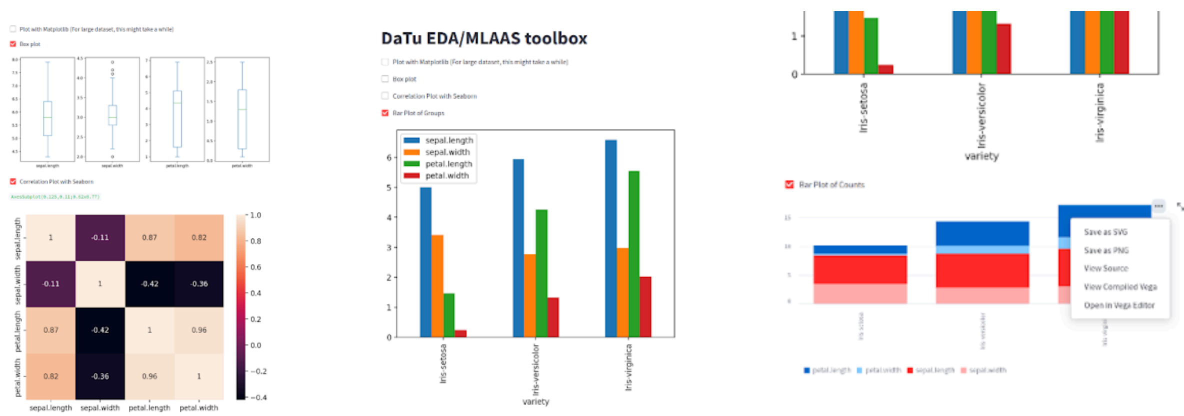


Figure 3: EDA visualization: confusion matrix, bar, and box plot

For the plots, users have the option to save the figure, resize it, or interact with it (zoom, label, highlight, etc.). In the modeling section, there is on-the-fly learning available for different algorithms, including logistic regression, random forest, decision tree, support vector machine, naive Bayes, K-nearest neighbors, and linear discriminant analysis. When users select any of these methods, the app will generate metrics (Accuracy, AUC score, Precision, Recall, F1 score), the confusion matrix, and the ROC curve.

While users can change the model, they are also provided with additional tuning parameters in the sidebar. They can adjust the train-test ratio (set as 0.7 by default) and select a sampling method (None, class weight, Random under sampler, Random over sampler, and SMOTE) for datasets with unbalanced data. This allows users to determine which model works better and if any changes in the hyperparameters can lead to improved performance in important metrics.

Exploratory Data Analysis (EDA) Whitepaper

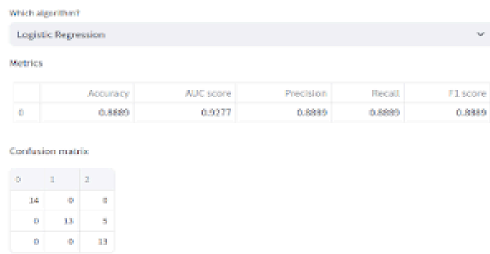


Figure 4: EDA Modeling: confusion matrix and metrics for classification

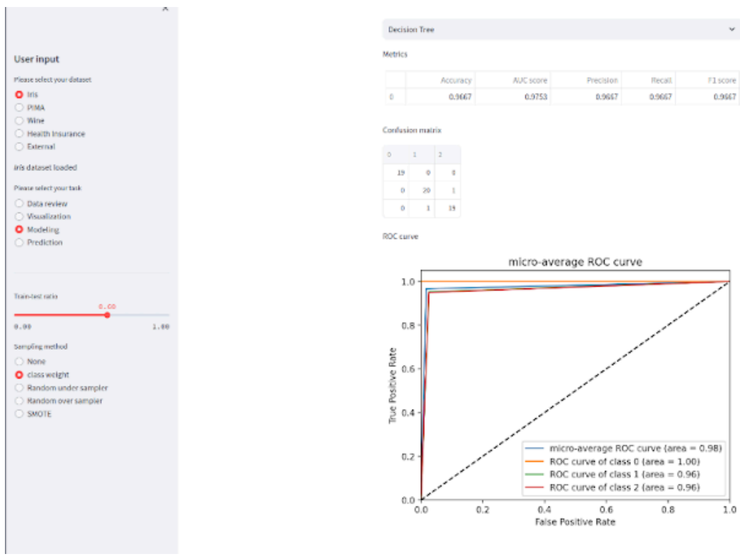


Figure 5: EDA modeling: changing hyper-parameters and view ROC curve

If a dataset has missing values, in both modeling and prediction, the app will automatically detect them. Then it will add a new option to the sidebar, asking the user to select from the available options to handle them (impute with the mean, drop the entry, or replace with zero).

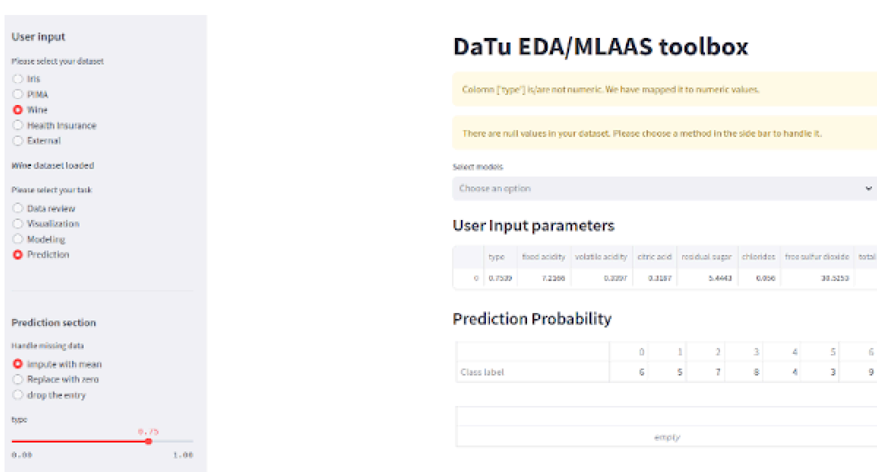
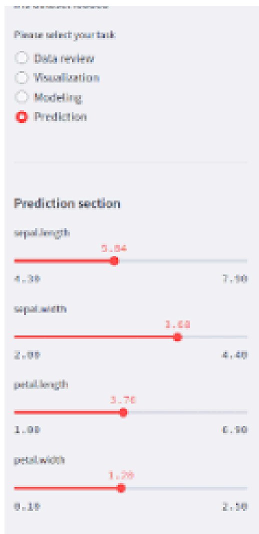
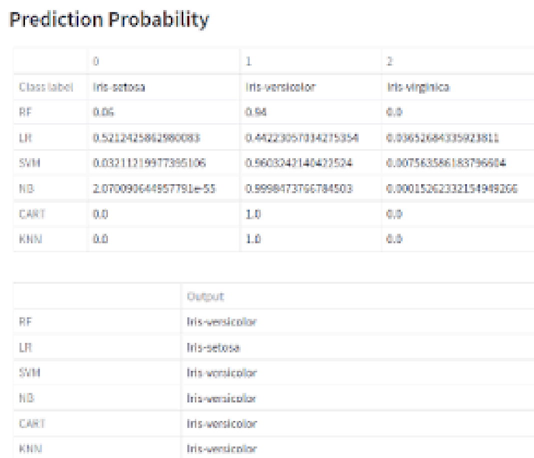


Figure 6: EDA prediction: choosing the method to handle missing data

Finally, the prediction will allow the user to choose from the models and compare the prediction results for any feature values within the range of the inputs. This will give the user the ability to use ensemble techniques, such as voting, to decide the class of the given features.





Prediction Probability

	0	1	2
Class label	Iris-setosa	Iris-versicolor	Iris-virginica
RF	0.06	0.94	0.0
LR	0.5212425862980083	0.44223057034275354	0.03652684335923811
SVM	0.03211219977395106	0.9603242140422524	0.007563586183796604
NB	2.070090644957791e-55	0.9998473766784503	0.00015262332154949266
CART	0.0	1.0	0.0
KNN	0.0	1.0	0.0

	Output
RF	Iris-versicolor
LR	Iris-setosa
SVM	Iris-versicolor
NB	Iris-versicolor
CART	Iris-versicolor
KNN	Iris-versicolor

Figure 7: EDA prediction: selecting the features value and compare different methods

5-2- architecture

In this section, we will provide some technical implementation for the EDA.

We are using Streamlit as the backbone in Python, along with components.v1 and pandas_profiling for additional features. To enhance overall performance and reduce queries to the server, we use caching. To do so, we decorate functions with @st.cache. The list of functions includes load_image, show_html (for Pandas Profiling), and read_csv.

For the side menu, we used the st.sidebar.slider function to add an extra layer to the main app. The user_input_features function is responsible for handling user inputs and models. It returns features and classifier objects. The table for different functions is shown below:

Table 1: List of functions in the EDA with the I/O and the descriptions

Function name	Inputs	Output	Description
show_html	HtmlFile	Read HTML	Cached read HTML

Exploratory Data Analysis (EDA) Whitepaper

read_csv	DATA_URL	Read CSV	Cached Read CSV
user_input_features	features	features, clfs	Create range of features and create clfs as model
compute_predition	clfs, features, labels, df_test	predictions	Append classifier models for prediction
show_scatter_plot	selected_species_df	void	Create scatter plot of the dataframe
select_species	source_df	selected_species_df	A sub dataframe having data for the selected species
show_histogram_plot	selected_species_df	void	Create histogram plot of selected features
_handle_missing	features, labels	new_features, new_labels	Handle missing values by imputing with mean, drop entry or fill with 0
handle_io	source_df	features, labels	Check if the data is numeric, otherwise map category, check null
show_machine_learning_model	source_df	void	show the performance of a trained ML Algorithm, check balance (apply sampling), compute accuracy

We use the compute_prediction function for the prediction part, which essentially utilizes scikit-learn models from a list. This allows us to fit the training set and test the user-selected features.

For different parts of the data analysis, we've developed separate functions. These include `show_scatter_plot` (for scatter plots), `select_species` (to display the number of distinct elements in the output column), and `show_histogram_plot` (for plotting histograms).

The data from the existing dataset or loaded dataset can contain missing values. To prevent errors, the training/testing datasets are first passed through the `_handle_missing` function, which ensures there are no missing values. We've implemented three approaches for handling missing values: impute with mean, drop the entry, or replace with zero. This option is displayed only if the dataset has missing values.

To detect missing values or non-numeric values in the dataset, we've designed a function called `handle_io`. Any data before being delivered to the model must pass through this function. It will detect non-numeric values, attempt to create a mapping based on the number of unique categories, and replace them with mapped numbers. Note that this is only for categorical features and does not work for continuous strings.

The `show_machine_learning_model` function is responsible for training the model. It begins by setting the train-test ratio, calling `handle_io`, and presenting additional options such as the sampling method, which can be used for imbalanced datasets. It then computes the confusion matrix and metrics (ROC AUC score, precision score, recall score, F1 score). For visualization, it also calls `roc_curve` to plot the ROC curve.

For datasets, we have already provided several classification datasets on public GitHub repository: https://github.com/pagand/MLAAS/tree/main/Colab The list of datasets includes Iris, PIMA, Wine, and Health Insurance. Users have the option to upload a new dataset or simply paste a URL for a raw CSV file.

Each section is categorized by its task. These tasks are Data Review for data exploration, Visualization for plotting the results, Classification Modeling for training a classifier, and Classification Prediction for predicting with the trained model.

Table 2: List of task in the EDA with the utility and the descriptions

Task	**Utility**	**Description**
''	Data loader	Capture data from predefined dataset, upload csv file, or URL
'Data review'	ProfileReport, EDA	Review data, produce report
'Visualization'	Histogram, Pyplot, Scatter	Visualize data
'Classification modeling'	Sklearn, imblearn	Create the classification model, handle imbalance, show measurements
'Classification prediction'	Sklearn, imblearn	Create the new datapoint from same range, learn model, predict