❖ Project Overview

- **Project Title: *CytoAuto Cluster***
- **Description:**

  "CytoAuto Cluster is a machine learning project focused on developing a semi-supervised deep clustering model for analyzing cytometry data. By accurately identifying cell populations in complex biological data, this project aims to improve clustering interpretability and enhance the reliability of automated cytometry analysis. This method holds potential for applications in biomedical research and healthcare."

- **Objective:**

  I. **Improve Clustering Accuracy**: Enhance the precision of identifying distinct cell populations within cytometry data, making it more reliable for biological research and clinical applications.

  II. **Enhance Interpretability**: Focus on producing clusters that are not only accurate but also interpretable, allowing researchers and practitioners to understand the biological significance of each identified group.

  III. **Leverage Semi-Supervised Learning:** Use a semi-supervised learning approach to combine labeled and unlabeled data, improving clustering performance even when labeled data is limited.

❖ Background and Motivation

- **Background on Cytometry Data Analysis:**

  I. **Overview of Cytometry**: Explain that cytometry is a technique commonly used in biological and medical research to measure characteristics of cells, such as size, complexity, and protein expression. This data is vital for applications in immunology, cancer research, and disease diagnostics.

  II. **Importance of Accurate Cell Population Identification:** Cytometry data contains vast amounts of cellular information, where accurately identifying cell types and populations is crucial. This helps in understanding biological processes and can aid in diagnosis, monitoring, and treatment of diseases.

  III. **Current Challenges:** Conventional clustering methods often struggle with the complexity of cytometry data, which has high dimensionality and potential noise. This can lead to less accurate and interpretable results, affecting the quality of insights derived from the data.

- **Motivation for *CytoAuto Cluster*:**

  I. **Need for Improved Methods**: There is a growing need for methods that can provide both accurate and interpretable clustering in cytometry, especially as data sizes and complexity increase.

II. **Benefits of Semi-Supervised Learning:** While labeled cytometry data can be limited, leveraging both labeled and unlabeled data through a semi-supervised approach can improve clustering performance and expand applicability.

III. **Contributions to Biomedical Research:** This project aims to contribute to biomedical research by providing a tool that can aid in discovering new cell types, studying immune responses, and identifying disease markers. The end goal is to make cytometry analysis more accessible, efficient, and insightful.

❖ Dataset Information

- **Dataset Overview:**
  - The dataset contains 265,626 rows and 42 columns, each representing different measurements and features related to cytometry analysis.
- **Key Columns and Their Descriptions:**
  I. **Time**: Captures the time point of the measurement.
  II. **Cell Length:** Refers to the measured length of each cell.
  III. **Feature Columns:** These include biological markers like **CD45RA**, **CD133**, **CD19**, and **CD22**, which measure protein expressions on the cell surface. These markers provide insights into cell characteristics.
  IV. **Viability:** A feature indicating cell viability, potentially used for filtering live cells.
  V. **File Number and Event Number:** Unique identifiers for each file and specific events within the data.
  VI. **Label and Individual:** Provide labeling information and identifiers for different individuals in the dataset.
- **Dataset Purpose and Relevance:**
  - This dataset is essential for clustering cell populations based on their surface markers, enabling the identification of biologically meaningful groups. The availability of both labeled and unlabeled data supports the semi-supervised learning approach for clustering.

❖ Methodology

❖ Implementation Details

❖ Results and Evaluation

❖ Challenges and Solutions

❖ Future Work

❖ Conclusion

❖ References and Acknowledgments

Model:

❖ Spliting of dataset into label and unlabel dataset:

- The purpose of this step is to prepare labeled and unlabeled datasets for a self-supervised learning task. Here's a detailed breakdown of the reasoning behind each part:
- The label_column specifies which column in the dataset contains the labels (e.g., the category or outcome you want to predict). This allows for splitting data into labeled and unlabeled groups.
- **Why this**
  - Self-supervised learning works well when you have a mix of labeled and unlabeled data.
  - label_df contains rows where the label_column is **not null**, meaning these rows have labels.
  - unlabeled_df contains rows where the label_column is **null**, meaning these rows are missing labels but still contain features. These will be used to learn useful feature representations.
- **Overall**
  1. Labeled Data is ready for supervised evaluation (fine-tuning).
  2. **Unlabeled Data** is prepared for self-supervised learning (to extract feature representations using the encoder).
  3. The split clarifies which data is used in what part of the self-supervised pipeline.

❖ Splitting of label dataset into training and testing dataset:

- This code is splitting the labeled dataset into training and testing subsets.
- Purpose of train test split
  - ✓ Training set (x_train, y_train): Used to train the supervised part of the model (e.g., fine-tuning the encoder).
  - ✓ **Testing set (x_test, y_test)**: Used to evaluate the performance of the model after training.
  - ✓ test_size=0.3: Specifies that 30% of the labeled data will be used for testing, while the remaining 70% is for training.
  - ✓ random_state=42: Ensures reproducibility. If you run the code again, the split will be the same because the random seed is fixed.
- Why we split the data
  - ✓ Training Data: The model learns patterns in the data using the features (x_train) and their corresponding labels (y_train).
  - ✓ **Testing Data**: After the model is trained, its performance is tested on unseen data (x_test and y_test). This simulates how well the model generalizes to new, unseen examples.
  - ✓ The training set will be used to fine-tune your encoder or a supervised model.
  - ✓ The testing set will evaluate the performance of the encoder after training.

❖ Training of logistic regression and xgboost on label dataset

- **What is Logistic Regression?**
  - Logistic Regression is a statistical method for binary or multiclass classification problems. It predicts the probability of an outcome (dependent variable) based on one or more input features (independent variables). Unlike linear regression**,** which

predicts continuous values, logistic regression predicts the probability of a class or event by applying the logistic function (also known as the sigmoid function).

- **How does it work?**
  - Sigmoid Function: Logistic regression uses a sigmoid function to convert a linear combination of the features into a probability value between 0 and 1. The formula for this function is:

    $P(y=1|x) = 1 / 1+e-(w0+w1x1+w2x2+\cdots+wnxn)1$
    - $P(y=1|x)$ is the probability that the instance belongs to class 1.
    - $w0,w1,\ldots,wn$ are the weights learned during training.
    - $x1,x2,\ldots,xn$ are the input features.
- **Multiclass Logistic Regression**: For multiclass classification (more than two classes), logistic regression uses the **one-vs-rest (OvR)** approach, where one binary classifier is trained for each class.
- **Why Use Logistic Regression?**
  - ✓ **Interpretability**: The model is easy to interpret, especially when you're dealing with linear relationships between features and the outcome. The coefficients show how each feature contributes to the decision**.**
  - ✓ **Efficiency**: Logistic regression is computationally inexpensive and works well for linearly separable data.
  - ✓ **Probabilistic Outputs**: Logistic regression is often used when you need probabilities (such as in this case where you are using predict_proba), rather than just class predictions.
  - ✓
- **XGBoost (Extreme Gradient Boosting)**
  - XGBoost is a powerful and efficient implementation of Gradient Boosting (GB). Gradient boosting is an ensemble learning technique where multiple weak learners (typically decision trees) are combined to form a stronger predictive model. XGBoost enhances the basic gradient boosting algorithm by adding several optimizations:
    - **Regularization**: It includes L1 (lasso) and L2 (ridge) regularization to prevent overfitting.
    - **Tree Pruning**: XGBoost performs pruning by removing trees that do not contribute much to the prediction.
    - **Parallelization**: The algorithm supports parallelization, making it faster to train on large datasets.
    - **Handling Missing Data**: XGBoost handles missing values during training by learning the optimal way to handle them.
  - How does it work?
    - ✓ XGBoost builds decision trees sequentially, where each new tree corrects the errors made by the previous trees. It minimizes a loss function, which is typically the difference between the predicted and actual labels.
    - ✓ The key concept of gradient boosting is to use the **gradient of the loss function** to determine the direction and magnitude of correction for each subsequent tree. This is why it is called "gradient boosting."
  - Why Use XGBoost?
    - ✓ **Accuracy**: XGBoost often outperforms other models, especially on structured/tabular data. Its ability to learn complex patterns in the data

through boosting makes it more accurate than models like logistic regression.

- ✓ **Handling Non-linearity**: While logistic regression assumes a linear relationship between features and the outcome, XGBoost can handle highly non-linear relationships by building decision trees.
- ✓ **Robustness**: XGBoost can handle various kinds of data (e.g., missing values, categorical data, large feature sets).
- ✓ **Speed**: XGBoost is highly optimized for performance and can train large datasets quickly due to parallelization and efficient memory usage.

➢ **Why Use Logistic Regression and XGBoost Here?**

- Self-Supervised Feature Extraction Encoder
  - ✓ A self-supervised encoder is used to extract meaningful features from the data without requiring labeled examples. The encoder learns to understand the structure of the data, typically by generating representations (features) of the data points. These features can later be used for downstream tasks such as classification, clustering, or anomaly detection.
- **Logistic Regression**
  - ✓ After the encoder has extracted features, **Logistic Regression** can be used as a simple **supervised model** to classify the data based on the features learned by the encoder. Specifically, here's how logistic regression fits into the pipeline:
  - ✓ **Classification Task**: Once you have the feature embeddings (from the encoder), you can use **Logistic Regression** to map these embeddings to class labels.
  - ✓ **Why Logistic Regression?** Since logistic regression is a linear model, it's useful for linearly separable problems where the extracted features can be linearly mapped to the target labels.
  - ✓ In **self-supervised learning**, after the encoder generates features, Logistic Regression can be trained on these features (even with limited labeled data) to perform classification. For example, it can classify images or samples into predefined categories based on the representations learned by the encoder.
- **XGboost**
  - ✓ Similarly, XGBoost can be applied to the features extracted by the encoder for supervised learning tasks. Here's how it complements the encoder:
  - ✓ **Handling Complex Patterns**: Unlike Logistic Regression, XGBoost is a **non-linear** model, meaning it can handle more complex decision boundaries. If the encoder's learned features capture non-linear patterns in the data, XGBoost can better leverage these features to make predictions.
  - ✓ **Why XGBoost?** XGBoost is particularly powerful in extracting complex relationships between features, making it suitable when the encoder's representations involve intricate, non-linear relationships. XGBoost's ability to handle noisy, unstructured, or high-dimensional data makes it an ideal choice for classification tasks after self-supervised feature extraction.
  - ✓ **Ensemble Learning**: XGBoost combines multiple decision trees to form a strong learner. This makes it effective at finding complex patterns in the data that may not be easily captured by simple linear models like logistic regression. In the context of self-supervised learning, the encoder could learn intricate feature representations, and XGBoost can then effectively classify based on these representations.

❖ Calculation of log loss for both the models

- *The log loss (also known as logarithmic loss or binary cross-entropy loss) is a performance metric commonly used for evaluating classification models, particularly when the output is probabilistic. It measures how well the predicted probabilities match the true class labels.*

- Log loss quantifies the accuracy of the probabilistic predictions made by a classification model. Unlike metrics such as accuracy, which simply measures whether the model's prediction was correct, log loss takes into account the **confidence** of the model's predictions.
- For each instance, the log loss is calculated using the following formula:

$$\text{Log Loss} = 1/N \sum_{i=1}^{N} [y_i \cdot \log(p_i) + (1-y_i) \cdot \log(1-p_i)]$$

  - ✓ N is the number of instances.
  - ✓ yi is the true label for the ithi^{th}ith instance (either 0 or 1).
  - ✓ pi is the predicted probability for the positive class (class 1).
  - ✓ The term log⁡(pi)\log(p_i)log(pi) is used when yi=1y_i = 1yi=1 and log⁡(1−pi)\log(1 - p_i)log(1−pi) is used when yi=0y_i = 0yi=0.

- **Lower Log Loss**: A lower log loss indicates that the predicted probabilities are close to the true labels, meaning the model is doing a good job of predicting the likelihood of each class.
- **Higher Log Loss**: A higher log loss indicates that the predicted probabilities are far from the true labels, meaning the model is uncertain or incorrect in its predictions.
- Log loss is particularly useful when dealing with **probabilistic classifiers** (like Logistic Regression and XGBoost with predict_proba) because it takes into account not just the correctness of predictions but also the confidence of the model.

➢ Encoding

- Label Encoding is a method of converting categorical labels into integer values. Each unique class or category in the target column is assigned a unique integer starting from 0
- Why Use Label Encoding?.
- Many machine learning algorithms (e.g., **Logistic Regression**, **XGBoost**) cannot process categorical data directly. They require numerical input for both features and target labels.
- In this case, **Label Encoding** converts the categorical labels in y into numerical values, making them suitable for the models.
- Why Label Encoding Works Here?
- **Classification Task**: The target variable is categorical (classes or labels), and label encoding converts it into numerical values for classification.
- **Suitability for Models**: Both **Logistic Regression** and **XGBoost** can process integer-encoded labels directly for classification tasks.

➢ Standardization

- Standardization transforms the features in a dataset to have:
- A **mean** of 0.
- A **standard deviation** of 1.
- It is done using the formula:

$$z = x - \mu / \sigma$$

- ✓ z: Standardized value.
- ✓ x: Original value of the feature.
- ✓ μ: Mean of the feature.
- ✓ σ: Standard deviation of the feature.

- The resulting transformed data follows a standard normal distribution with μ=0\mu = 0μ=0 and σ=1\sigma = 1σ=1.
- Why Standardization is Important?
- **Consistency Across Features**:
  - ✓ Features in the dataset may have different units (e.g., age in years, income in dollars).
  - ✓ Models that rely on distance or gradient-based optimization (e.g., Logistic Regression, XGBoost) can perform poorly if features have vastly different scales.
- **Faster Convergence**:
  - ✓ Gradient-based algorithms (like those used in Logistic Regression and XGBoost) converge faster when the features are standardized because the scale of the gradients becomes more uniform.
- **Improves Model Performance**:
  - ✓ Standardization helps models interpret all features equally, preventing features with larger values from dominating.
- Why is Standardization Used Here?
  - ✓ Logistic Regression uses gradient descent optimization. Features with larger scales may dominate the gradients, leading to incorrect or slow optimization.
  - ✓ Standardization ensures that all features contribute equally to the model.
  - ✓ XGBoost doesn't require standardized data as strictly as Logistic Regression because it uses decision trees internally, which are scale-invariant.
  - ✓ However, standardizing data can improve numerical stability and performance when using probabilistic outputs (e.g., predict_proba).
- What Happens if You Don't Standardize?
  - ✓ May produce poor results because features with larger scales disproportionately influence the model.
  - ✓ Training might take longer or fail to converge properly.
  - ✓ While the model might still perform reasonably well, thelack of standardization could lead to issues when using additional algorithms that depend on scaled data.

❖ Encoder model (self supervise)

- *The self-supervised encoder model is designed to learn meaningful representations of unlabeled data by using a corruption-and-reconstruction paradigm. This approach uses masking to introduce noise into the input data and then trains the model to estimate both the masked regions and reconstruct the original features. The model* **consists of a neural network**

with two parallel output layers, optimized with two separate loss functions: one for mask estimation and another for feature reconstruction.

- Masking and Corruption
  - ✓ **Masking:** A binary mask is generated using a given masking probability (p_m). Each mask value determines whether the corresponding feature in the input data is replaced with a shuffled value.
  - ✓ **Corruption:** Features are replaced with shuffled values based on the binary mask, introducing structured noise into the data.
- Model Architecture
  - ✓ **Input Layer:** Accepts the unlabeled data with n features as input.
  - ✓ **Hidden Layer:** A dense layer with ReLU activation serves as the encoding layer, creating a compressed representation of the input data.
  - ✓ Output Layers
  - ✓ **Mask Estimation Output:** Predicts the binary mask that was applied to the input data (sigmoid activation).
  - ✓ **Feature Estimation Output:** Reconstructs the original unmasked data (sigmoid activation).
- Loss Functions
  - ✓ **Binary Cross-Entropy:** Used for the mask estimation task, ensuring accurate prediction of the binary mask.
  - ✓ **Mean Squared Error (MSE):** Used for the feature reconstruction task, ensuring the model can accurately restore the original unmasked data.
- Training
  - ✓ The model is trained using two weighted loss functions with the feature estimation loss weighted by a tunable parameter, alpha.
  - ✓ Input data is corrupted, and the model is trained to predict the mask and reconstruct the uncorrupted data simultaneously.
- Encoder Extraction
  - ✓ The hidden layer of the model is extracted as a standalone encoder. This encoder provides a meaningful representation of the input data, useful for downstream tasks.
- Applications
  - ✓ Dimensionality reduction for unlabeled data.
  - ✓ Pretraining representations for downstream supervised learning tasks.
  - ✓ Anomaly detection by analyzing feature reconstructions.
  - ✓ Data imputation for handling missing or corrupted values.
- Advantages
  - ✓ Efficient learning from unlabeled data.
  - ✓ Flexible architecture that can handle various data types.
  - ✓ Improves generalization by focusing on meaningful feature representations.

❖ Semi supervise function

- *This semi-supervised learning framework combines labeled and unlabeled data to train a classification model. It leverages a pre-trained encoder for feature representation and uses both supervised and unsupervised objectives during training. The model employs an iterative optimization process where the supervised loss (on labeled data) and an unsupervised loss (on unlabeled data) are minimized.*
- **Custom Model Architecture**
  - ✓ A fully connected neural network built with TensorFlow/Keras**.**
  - ✓ Inputs: Encoded features from the pre-trained encoder.
  - ✓ **Hidden Layers:** Two dense layers with customizable dimensions and activation functions.
  - ✓ Outputs:
  - ✓ **Logit Layer:** Produces unscaled logits for classification.
  - ✓ **Softmax Layer:** Converts logits into probabilities for class predictions

- **Supervised and Unsupervised Learning**
  - ✓ Supervised Loss: Sparse categorical cross-entropy, used for labeled data.
  - ✓ **Unsupervised Loss:** Measures the variance in the logits of unlabeled data to encourage class separation.
- **Batch Processing**
  - ✓ Combines randomly sampled labeled and unlabeled batches during training.
  - ✓ Unlabeled batches are corrupted with a binary mask to improve generalization.
- **Training Loop**
  - ✓ Trains the model over multiple epochs.
  - ✓ Periodically evaluates on a validation set to monitor supervised loss.
- **Pre-trained Encoder**
  - ✓ A separate encoder model is pre-trained (e.g., using a self-supervised method) to encode both labeled and unlabeled data.
  - ✓ The encoder reduces the dimensionality of input features and improves the quality of learned representations.
- **Apllication**
  - ✓ Leveraging large quantities of unlabeled data in the presence of limited labeled data.
  - ✓ Building robust classifiers in domains with high labeling costs (e.g., medical imaging, NLP).
  - ✓ Improving generalization by incorporating unsupervised objectives.
- **Advantages**
  - ✓ Combines supervised and unsupervised learning for better performance in low-labeled data scenarios.
  - ✓ Flexible design allows for integration with any pre-trained encoder.
  - ✓ Regularization using unsupervised loss prevents overfitting and improves robustness.
  - ✓

❖