**Page 1: Project Title – *Algonix: Custom Machine Learning Library***

**Vaishnav:**  
Good [morning/afternoon], everyone. We are excited to present our project **“Algonix – A Custom Machine Learning Library.”**  
This project is developed by both of us — **Bhushan Zade** and **Vaishnav** — as a hands-on implementation of classic machine learning algorithms and evaluation metrics, all bundled into a single, reusable Python library.

**🔹 Page 2: Introduction**

**Vaishnav:**  
The goal behind Algonix was to understand machine learning from the ground up by not just using pre-built functions from libraries like scikit-learn, but by implementing core algorithms ourselves.  
Through this project, we explored the math, logic, and Python programming behind various machine learning models. The result is a modular library that can be easily installed and used with the command pip install algonix.  
It’s simple, well-documented, and can be a great learning resource for beginners or a customizable tool for developers.

**🔹 Page 3: Performance Metrics (Bhushan)**

**Bhushan:**  
Let’s begin with evaluation metrics. These are crucial for assessing how well our machine learning models perform.  
In **Algonix**, we’ve implemented the following:

* **MAE** (Mean Absolute Error): Gives the average absolute difference between actual and predicted values.
* **MSE** (Mean Squared Error): Squares the errors to penalize larger mistakes.
* **RMSE** (Root Mean Squared Error): Takes the square root of MSE to bring it back to the original unit.
* **R² Score**: Indicates how well the model explains variance in the data.
* **Adjusted R²**: Especially useful in multiple linear regression, as it adjusts the R² based on the number of predictors used.

**🔹 Page 4: Simple Linear Regression**

**Bhushan:**  
We implemented **Simple Linear Regression** from scratch — which models the relationship between a single input variable and the target.  
It calculates the slope and intercept using the least squares method. The model can predict values and evaluate performance using our custom metrics.

**🔹 Page 5: Multiple Linear Regression**

**Bhushan:**  
Building on that, **Multiple Linear Regression** allows us to use more than one feature.  
Here, we’ve implemented it using the normal equation and gradient descent. We also allow users to evaluate the model with Adjusted R², which is more informative in this case.

**🔹 Page 6: Logistic Regression**

**Bhushan:**  
For classification tasks, we implemented **Logistic Regression**, where the output is between 0 and 1.  
We use the sigmoid activation function and optimize weights using gradient descent. This is useful in binary classification problems such as spam detection or fraud classification.

**🔹 Page 7: Decision Tree (Vaishnav)**

**Vaishnav:**  
Next, we implemented a **Decision Tree classifier**.  
Our implementation includes recursive splitting using the Gini Index as the splitting criterion. The tree structure is created manually and can classify categorical and numerical data effectively. It’s one of the most interpretable ML models.

**🔹 Page 8: K-Nearest Neighbors (KNN)**

**Vaishnav:**  
In **KNN**, the prediction is based on the majority class among the K closest data points.  
We used Euclidean distance to calculate the closeness and implemented the model for both classification and regression scenarios. It’s simple yet powerful for pattern-based tasks.

**🔹 Page 9: Gradient Boosting (Regression & Classification)**

**Vaishnav:**  
Gradient Boosting is an advanced ensemble method.  
We implemented a lightweight version of **Gradient Boosting** for both regression and classification.  
The core idea is to combine weak learners, typically decision stumps, into a strong learner through iterative improvement.  
We use the residuals to guide the learning process in each stage.

**🔹 Page 10: K-Means Clustering**

**Vaishnav:**  
In the unsupervised learning category, we implemented **K-Means Clustering**.  
The algorithm starts with random centroids and iteratively refines them based on the distance from cluster members until convergence.  
We visualize the clusters to validate results and also allow users to set the number of clusters manually.

**🔹 Page 11: pip install algonix + PyPI Info (Bhushan)**

**Bhushan:**  
Once our library was stable, we packaged and published it on **PyPI** — Python’s package index.  
You can now install it easily using:

pip install algonix

We have a proper setup file, documentation, and versioning in place. You can find our project here:  
👉 <https://pypi.org/project/algonix/>

We also included examples and usage instructions in the official documentation to help others get started quickly.

**🔹 Page 12: Conclusion**

**Bhushan:**  
To conclude, this project gave us deep insights into the working of machine learning algorithms — from math to implementation and packaging.  
It wasn’t just about building models, but also about writing reusable, clean code and sharing it with the open-source community.  
We believe Algonix can serve as a stepping stone for learners who want to go beyond just using existing libraries.

**🔹 Page 13: Thank You**

**Vaishnav:**  
Thank you all for listening to our presentation.  
We’re proud of how far we’ve come with Algonix, and we hope it inspires others to build their own tools and libraries.  
If you have any questions or suggestions, we’d be happy to hear them!