***CAPSTONE PROJECT TITLE: [Yield Prediction Using Machine Learning]***

**Group Members:**

1. **Lielina Akele**
2. **Melat Kebede**

# ****Data Preparation / Feature Engineering****

### ****1. Overview****

Data preparation and feature engineering are critical stages in any machine learning project, as they directly affect the model's ability to learn from data and make accurate predictions. In our crop yield prediction project, we aimed to extract meaningful insights from environmental factors such as precipitation, humidity, temperature, and others. These steps ensured the data was clean, structured, and optimally transformed for training robust predictive models.

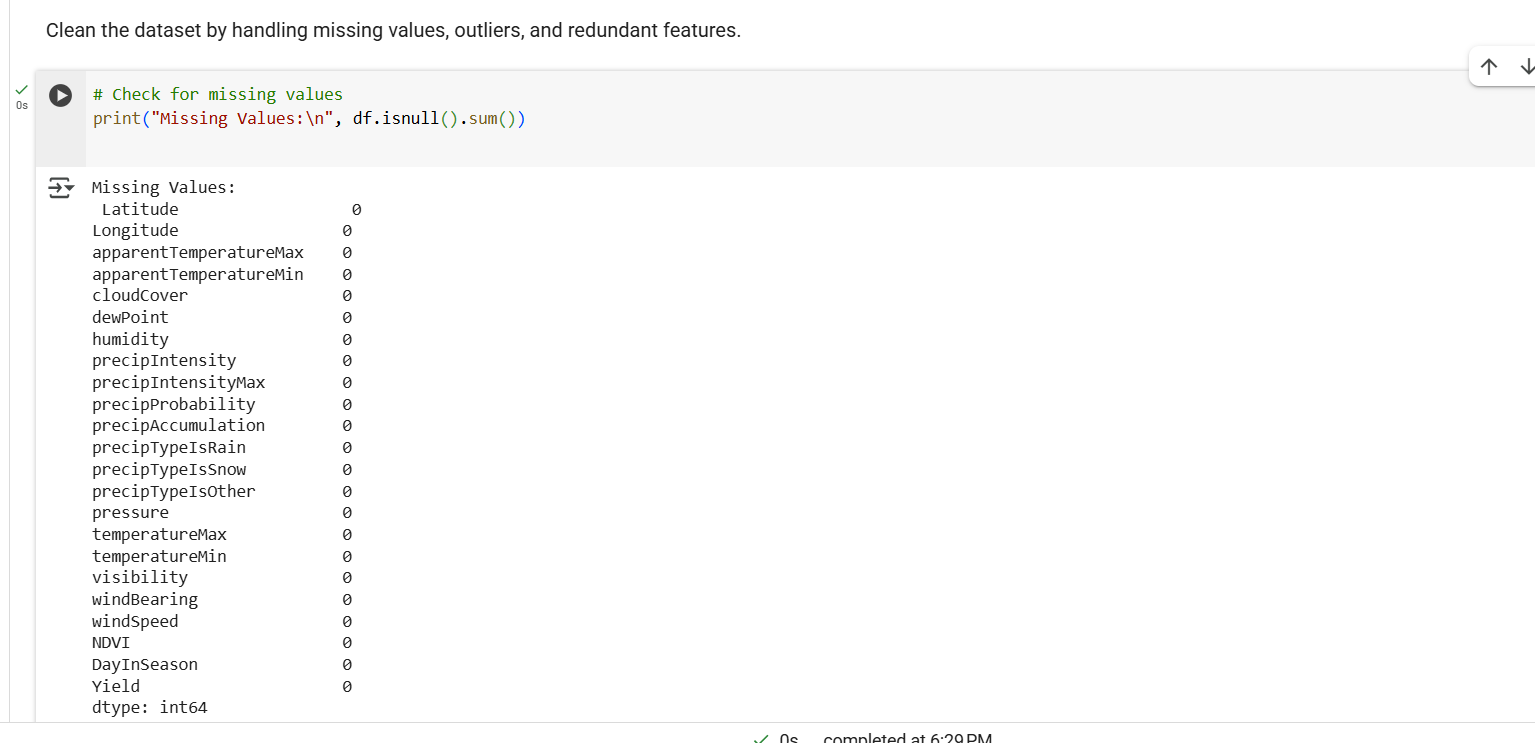
### ****2. Data Collection****

Initially, we attempted to collect data from the Ethiopian Institute of Agricultural Research (EIAR). However, due to the lack of meteorological features and a small sample size, this dataset was deemed insufficient. Although if we are able to get a local data we plan to do a transfer learning to the already trained model. But for now, we turned to a well-structured public dataset from Kaggle, which included over 200,000 rows of weather-related features (humidity, precipitation probability, dew point, cloud cover, etc.) and crop yield values. The Kaggle dataset provided the granularity and feature variety necessary for effective model training.

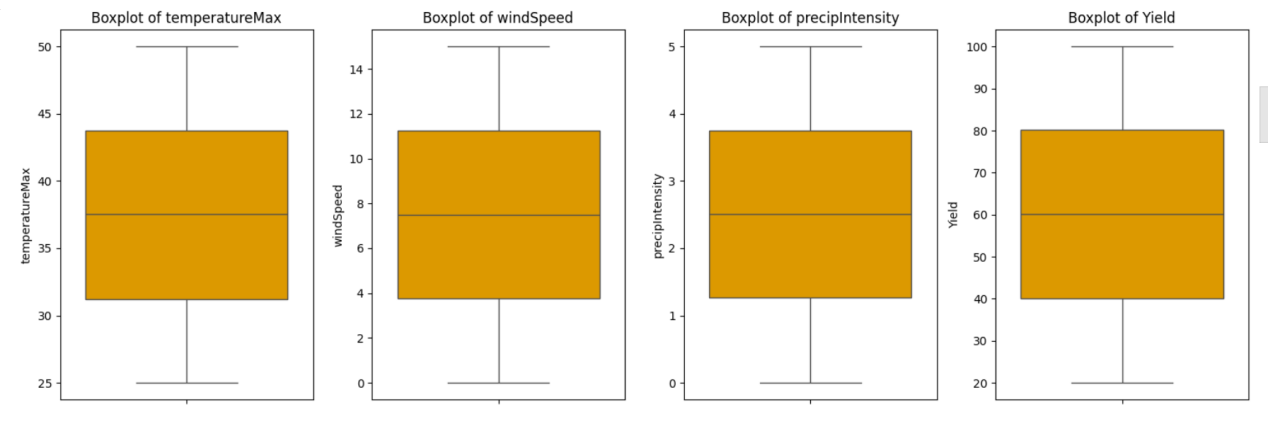
### ****3. Data Cleaning****

The Kaggle dataset required minimal cleaning due to its structured nature. Key cleaning steps included:

* **Missing Values:** No missing values were found.



* **Outlier Detection:** We used the **Interquartile Range (IQR)** method and visualized it with **box plots**. No significant outliers were observed.

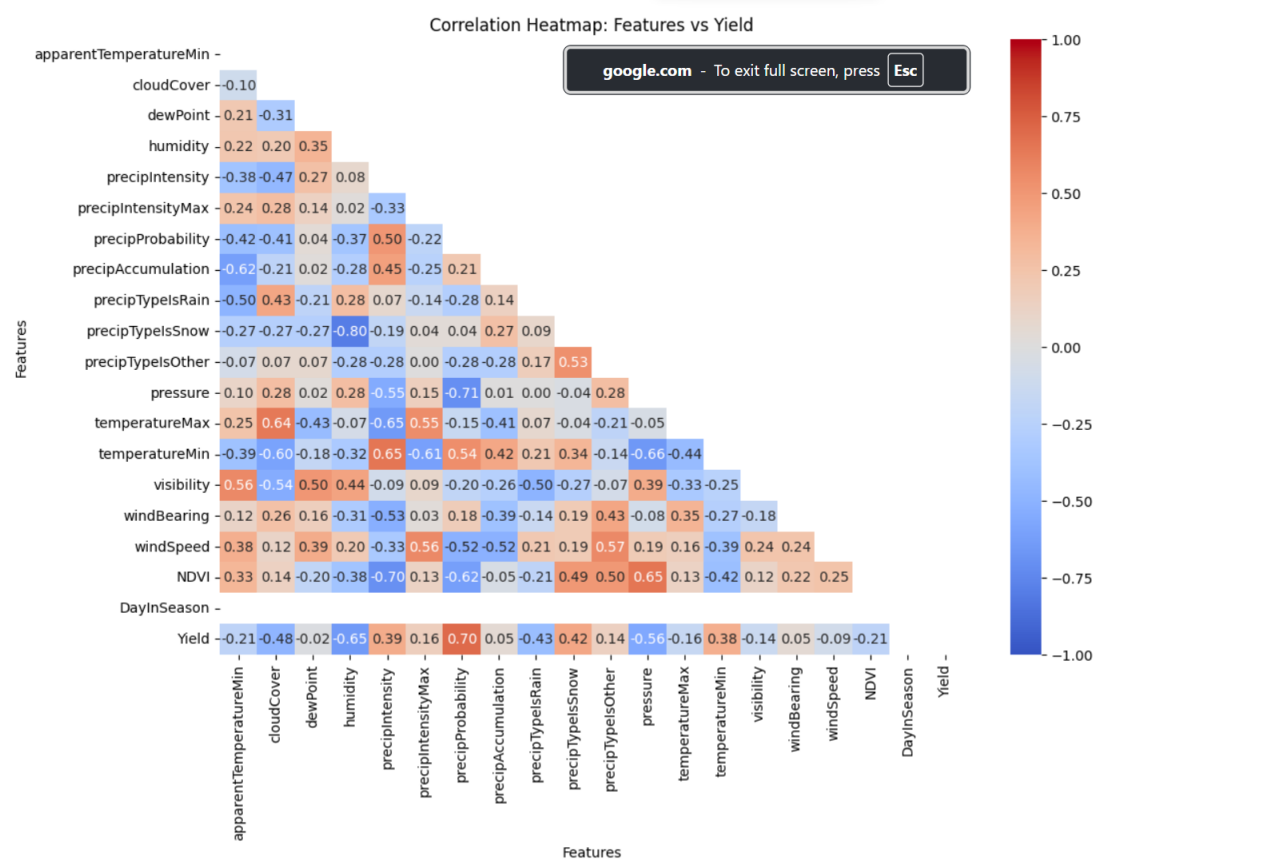


* **Data Consistency:** Features like precipTypeIsSnow, which are irrelevant in the Ethiopian context (since Ethiopia doesn't experience snow), were identified as having no meaningful variation (always 0) and were discarded from further analysis.

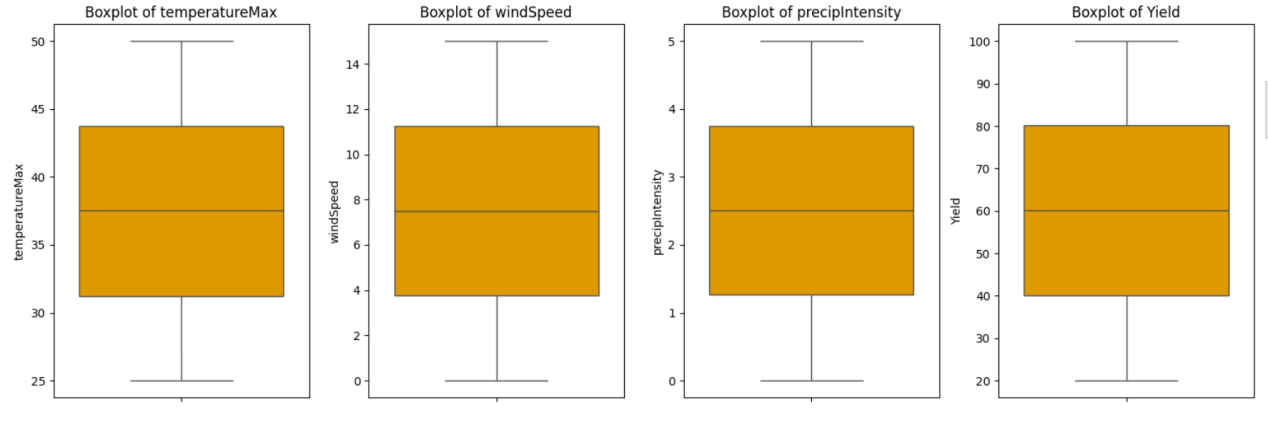
### ****4. Exploratory Data Analysis (EDA)****

EDA was performed to understand feature relationships and the data distribution:

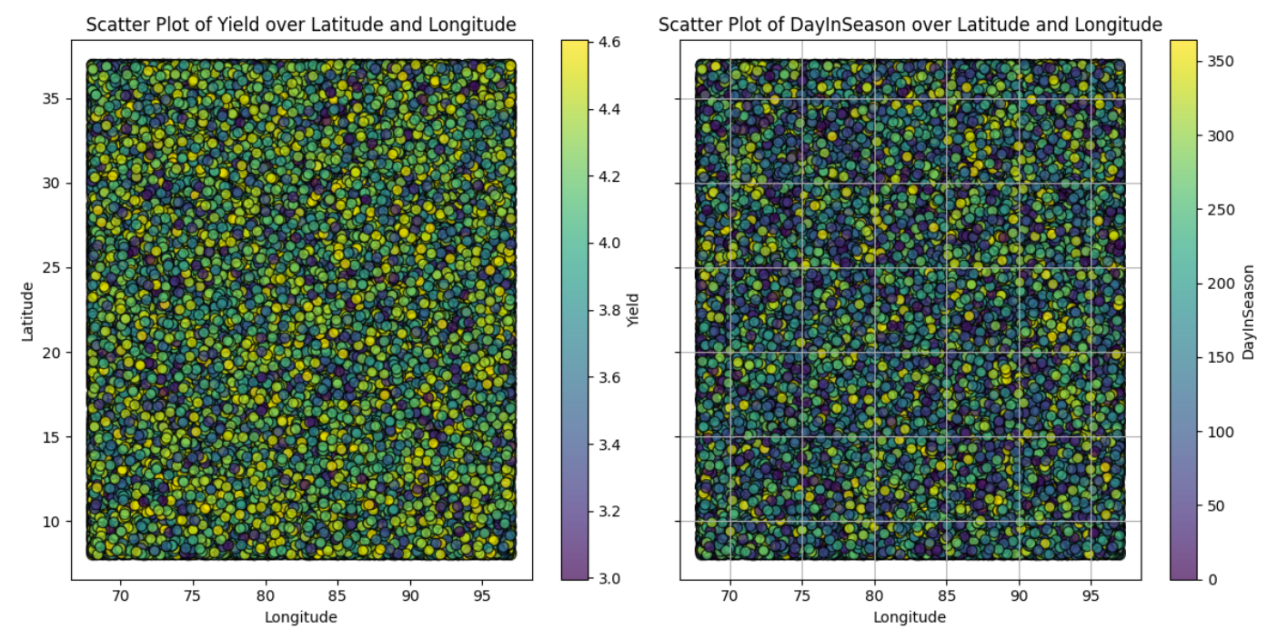
* **Correlation Analysis** helped identify strong predictors of yield, such as precipitation intensity, humidity, and dew point.



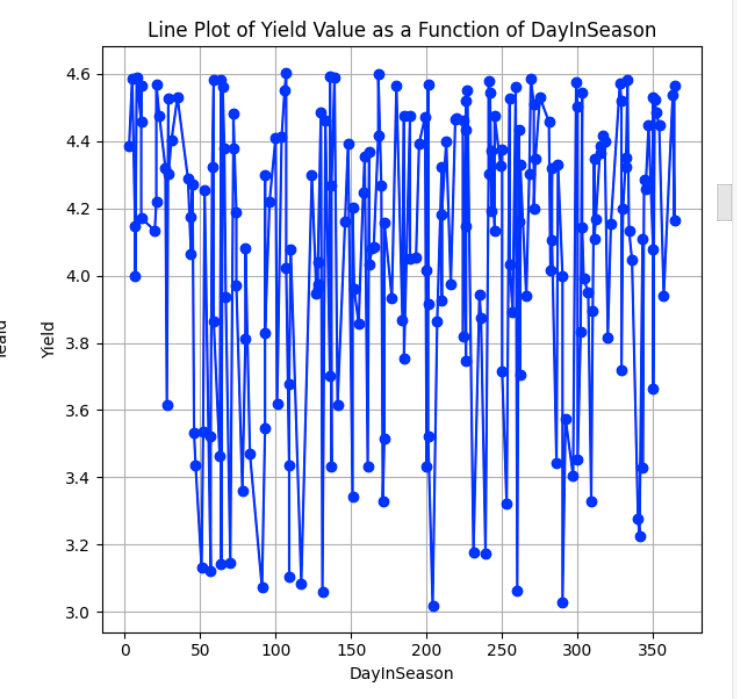
* **Box Plots** were used to visualize the spread of numerical features and detect outliers.

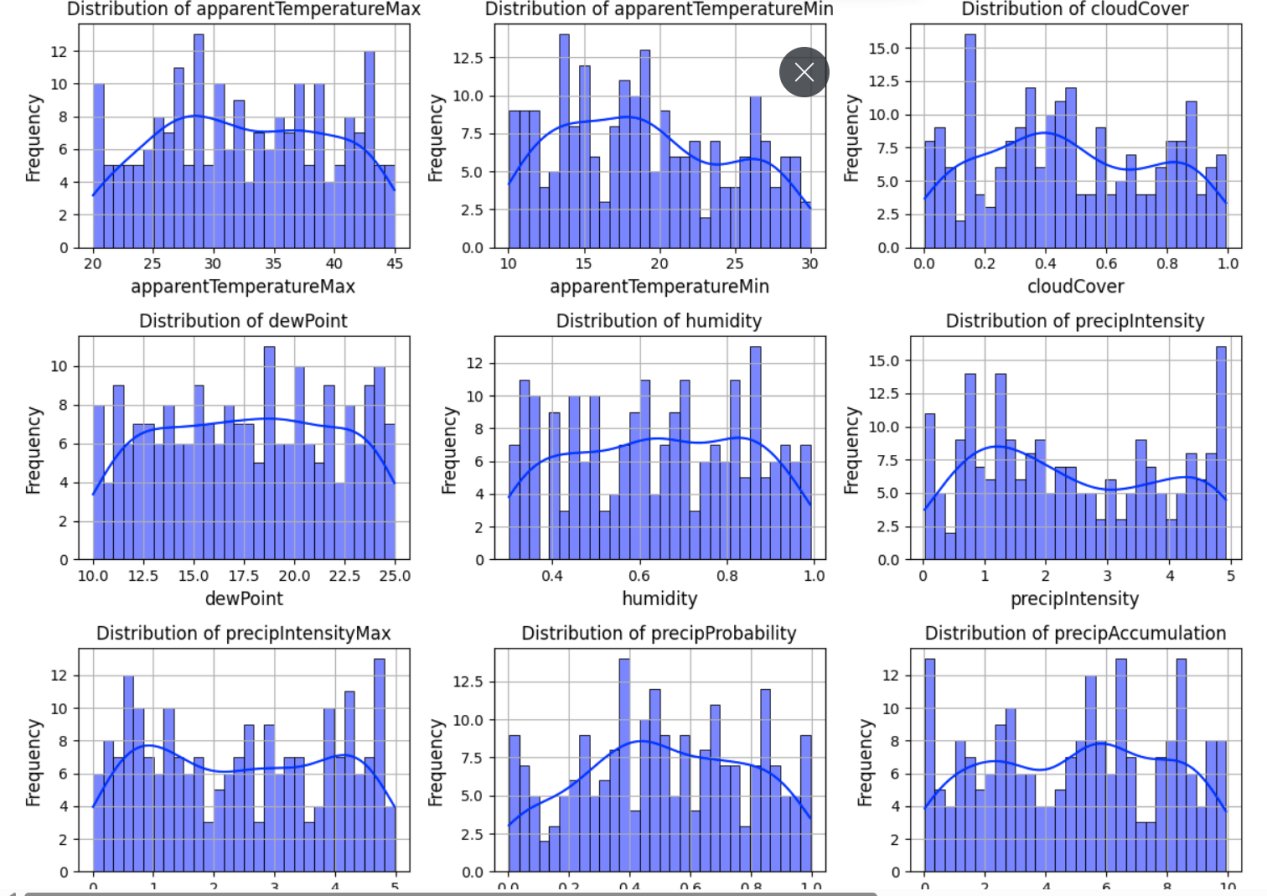


* **Scatter Plot:** to visulaize and better understand the data



* **Line Plot:** to visulaize and better understand the data.





**Key Insights:**

* Precipitation and humidity have the strongest positive correlation with crop yield.
* Features like cloud cover and dew point showed non-linear effects.

### ****5. Feature Engineering & Data Transformation****

* **Log Transformation**: to reduce skewness in positively skewed data we applied logarithmic transformation.
* **No encoding** was required since all features were numerical.
* **Data Splitting:** The dataset was divided into **80% Training set (160,000 samples) and 20% Testing set (40,000 samples)**

# ****Model Exploration****

### ****1. Model Selection****

We explored multiple models and selected:

**Deep Neural Networks (DNN)**: For their strength in modeling complex, non-linear relationships.

**Strengths of DNN:**

Excellent at capturing complex patterns.

Suitable for large datasets.

### ****2. Model Training****

We used Scikit-learn and TensorFlow/Keras libraries in Python:

model = keras.Sequential([

    keras.layers.Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(train\_features.shape[1], 1)),

    keras.layers.MaxPooling1D(pool\_size=2),

    keras.layers.Conv1D(filters=32, kernel\_size=3, activation='relu'),

    keras.layers.MaxPooling1D(pool\_size=2),

    keras.layers.Flatten(),

    keras.layers.Dense(64, activation='relu'),

    keras.layers.Dense(32, activation='relu'),

    keras.layers.Dense(1)  # Regression output layer

])

# Compile model

model.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError(), metrics=[tf.keras.metrics.MeanAbsoluteError()])

# Train model

history = model.fit(train\_features, train\_labels, epochs=50, validation\_split=0.2, batch\_size=32)

# For Random Forest

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

### ****3. Model Evaluation****

We used the following metrics:

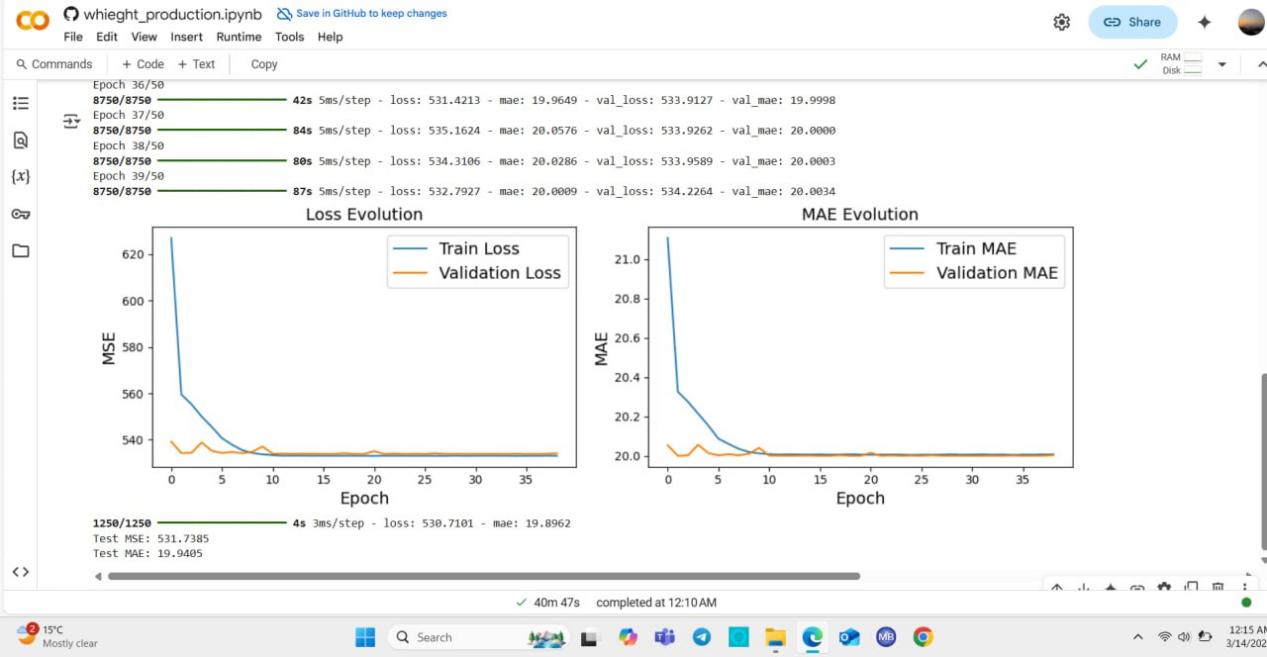
· **R² Score**: Measures the proportion of variance explained by the model. A score closer to 1 is ideal. All models scored poorly, likely due to data limitations.

· **RMSE (Root Mean Square Error)**: Measures the average magnitude of error. LightGBM achieved a reasonably low RMSE (23.06), indicating acceptable performance.

· **Training and Prediction Time**: Important for real-time deployment. LightGBM was efficient in both.

**Evaluation Visualizations:**

**Learning curves** (for DNN) to monitor overfitting or underfitting



### ****4. Code Implementation****

# Data Normalization

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

# Splitting the dataset

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Model Evaluation

from sklearn.metrics import mean\_squared\_error, r2\_score

predictions = model.predict(X\_test)

rmse = mean\_squared\_error(y\_test, predictions, squared=False)

r2 = r2\_score(y\_test, predictions)

print(f"RMSE: {rmse}")

print(f"R² Score: {r2}")