Crop Yield Prediction Project Report

Data Preparation/Feature Engineering

1. Overview:

Data preparation and feature engineering are critical stages in the machine learning pipeline, significantly impacting the model's performance. These phases ensure that the raw data is cleaned, properly structured, and enriched with meaningful features. This enhances the model's ability to learn patterns and relationships, which ultimately improves the accuracy of predictions.

2. Data Collection:

The dataset for this project was collected from various government agencies and research organizations, primarily in CSV format. The data includes historical weather information such as temperature, precipitation, and humidity, along with soil metrics such as pH levels and nutrient content.

Preprocessing during data collection:

- All data was converted into a standard format (CSV) for consistency.
- Some columns had inconsistent names or extra spaces, which were corrected by stripping whitespaces.

3. Data Cleaning:

To ensure data quality, the following cleaning steps were applied:

- **Handling missing values**: Numerical columns with missing values were filled with the column mean, while categorical columns with missing values were dropped, as they represented small fractions of the data.
- **Removing unnecessary columns**: Columns like 'Unnamed: 0' were dropped as they were not relevant to the analysis.
- **Renaming columns**: The target variable, originally labeled as 'hg/ha_yield', was renamed to 'yield' for clarity and consistency.
- **Removing duplicates**: Duplicate records were identified and removed to avoid redundancy in model training.

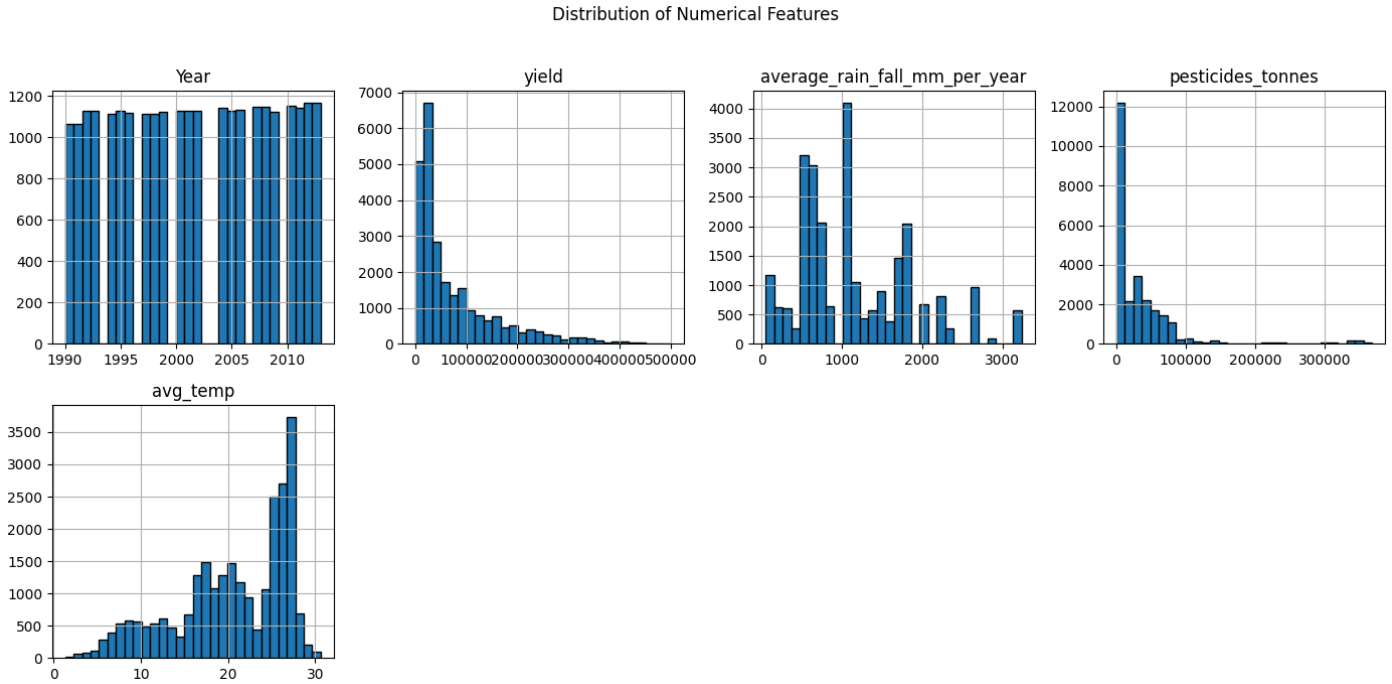
```
# Handling missing values and cleaning
numeric_cols = df.select_dtypes(include=['number']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
df.dropna(subset=['Area', 'Item'], inplace=True)

# Renaming target column
if 'hg/ha_yield' in df.columns:
    df.rename(columns={'hg/ha_yield': 'yield'}, inplace=True)

# Dropping unnecessary columns
if 'Unnamed: 0' in df.columns:
    df.drop(columns=['Unnamed: 0'], inplace=True)
```

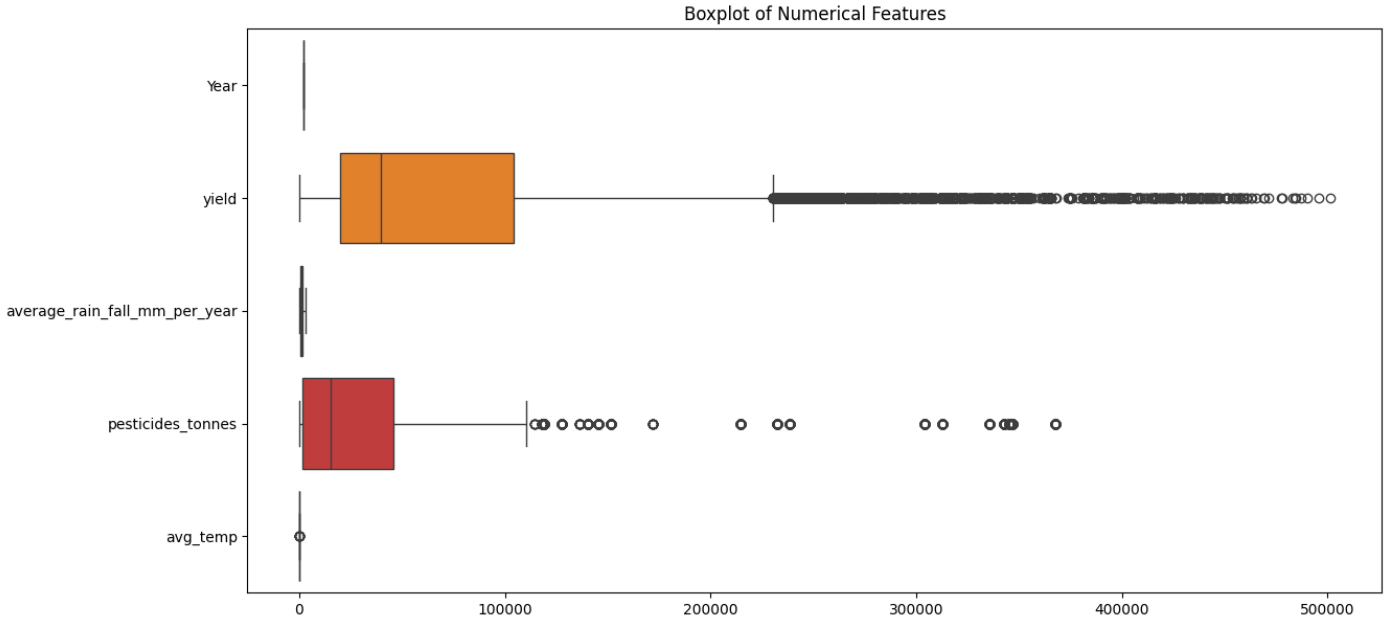
4. Exploratory Data Analysis (EDA):

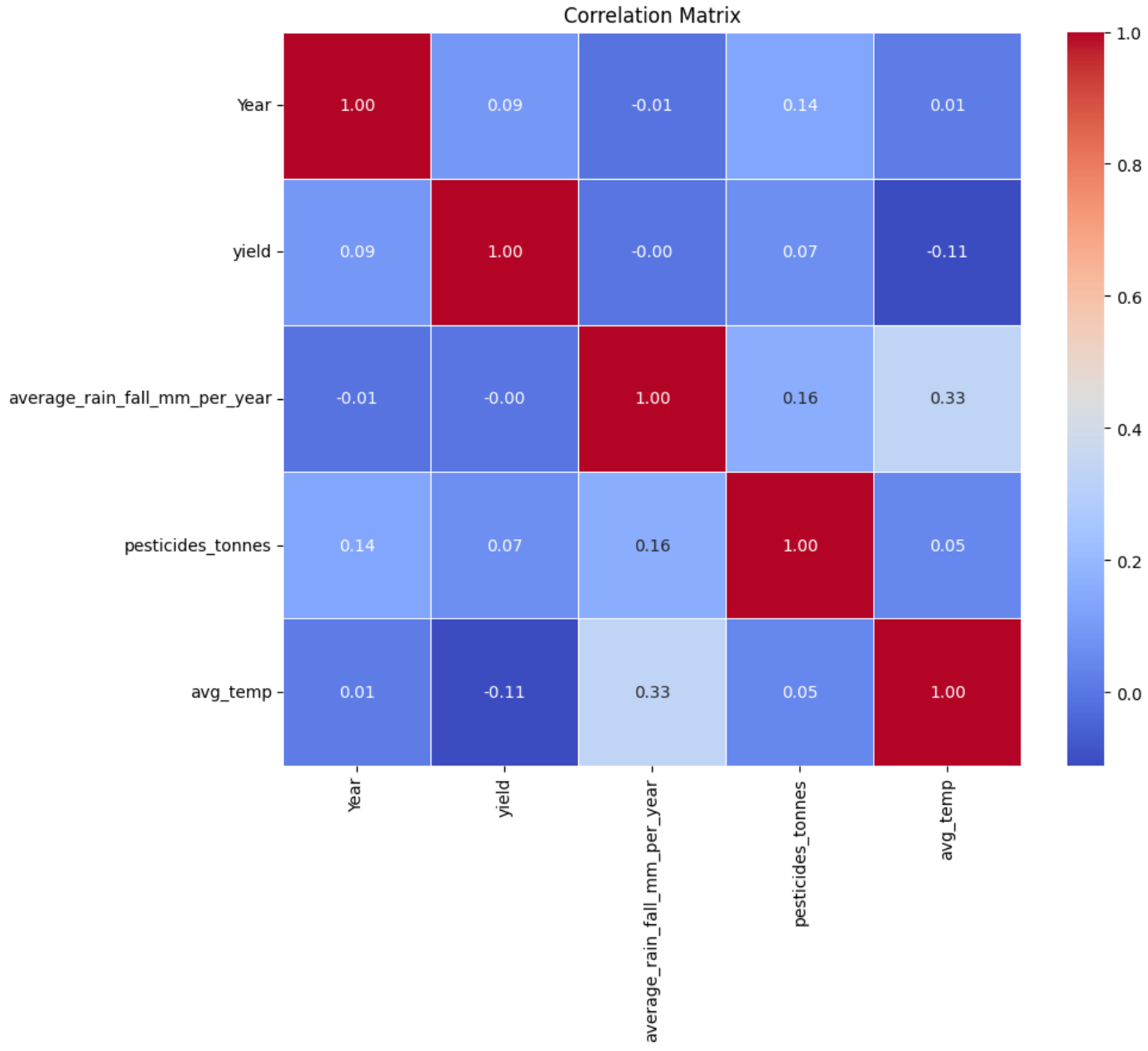
EDA was performed to better understand the data distribution, detect patterns, and identify relationships between variables. Key insights from the analysis include:

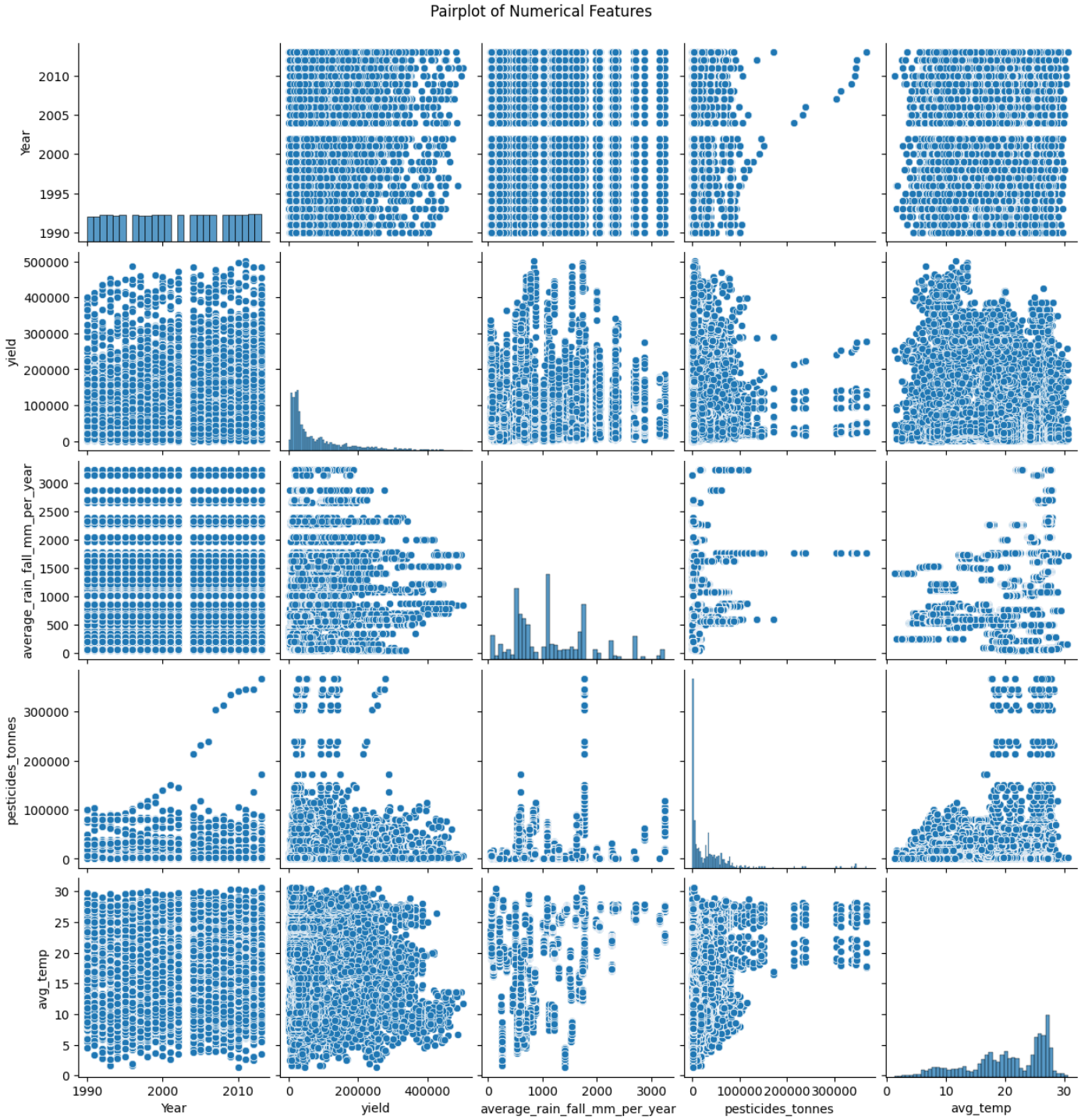
- **Distribution of numerical features**: Histograms were created to visualize the distribution of features like temperature, rainfall, and yield.
- **Correlation analysis**: A correlation matrix was plotted to examine relationships between features. For instance, a strong correlation between rainfall and yield was observed.
- **Boxplots and scatter plots**: Used to detect outliers and study relationships between yield and other features like temperature and rainfall.

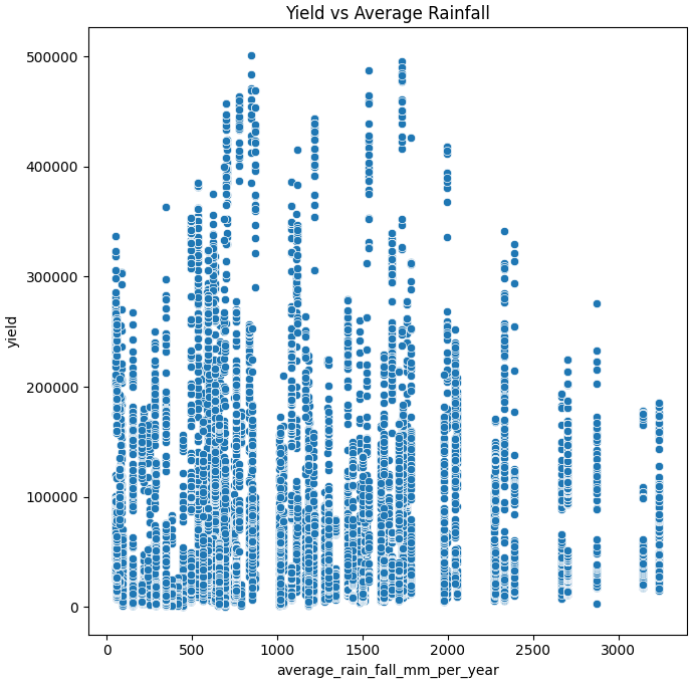
```
# Histogram of numerical features
plt.figure(figsize=(14, 7))
df.hist(bins=30, figsize=(14, 7), layout=(2, 4), edgecolor='black')
plt.suptitle('Distribution of Numerical Features')
plt.show()

# Correlation matrix
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

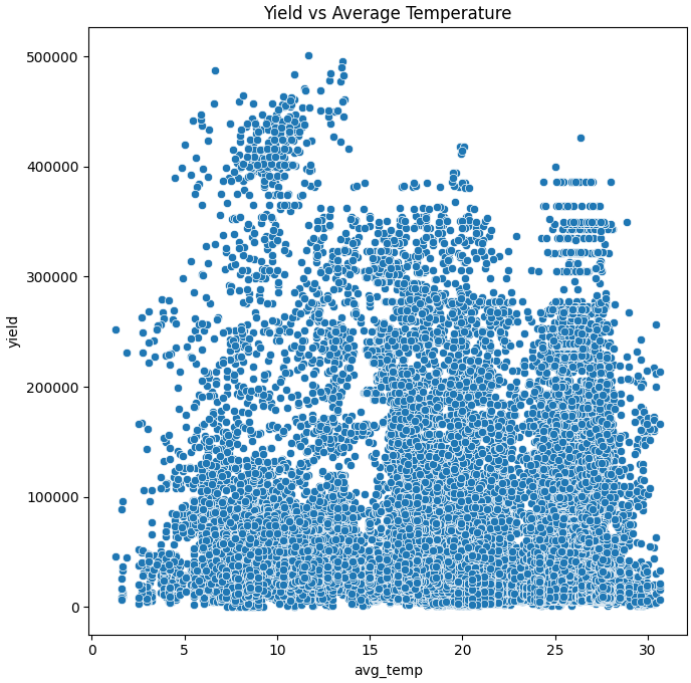


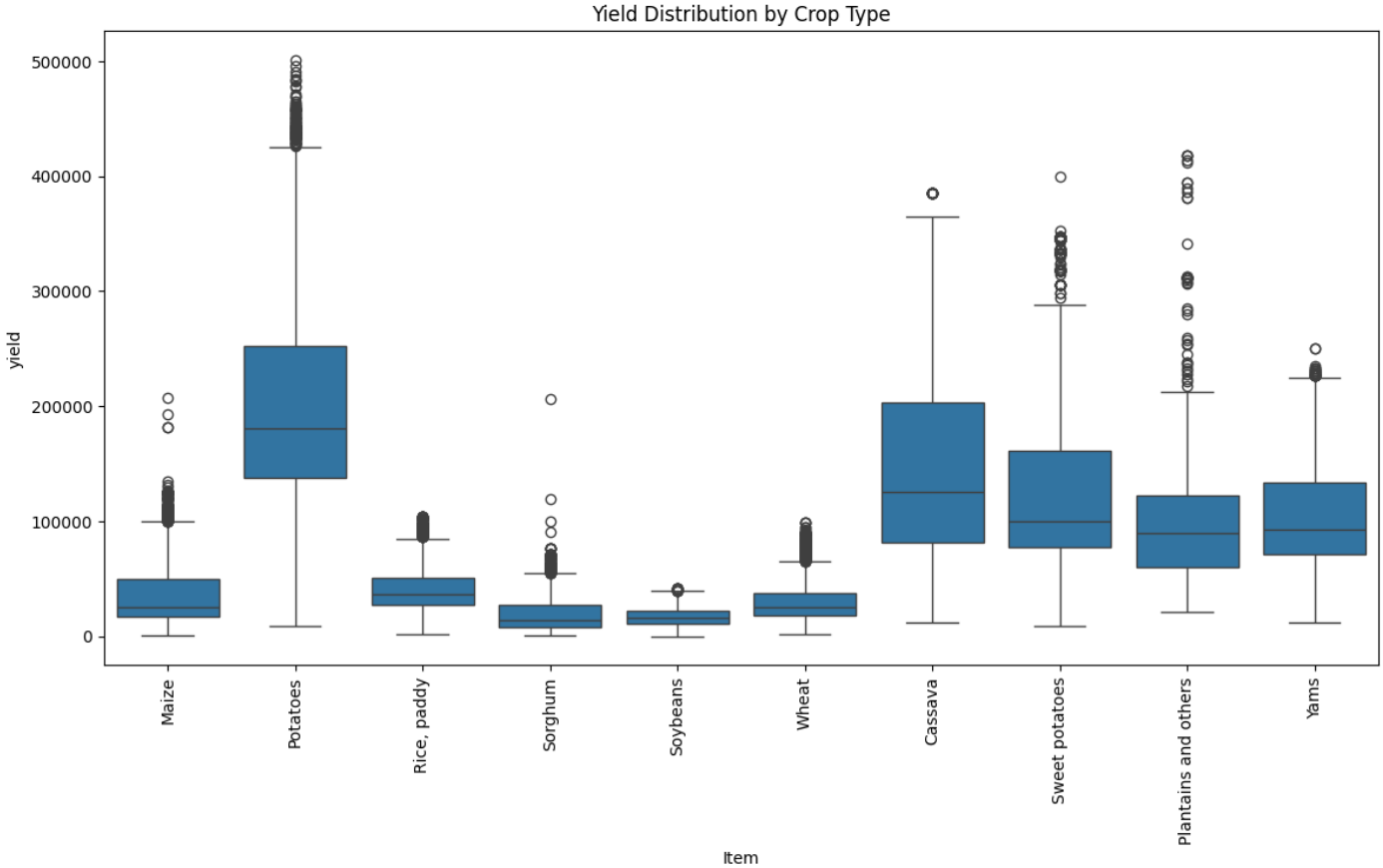

Boxplot of Numerical Features
Year
yield
average_rain_fall_mm_per_year
pesticides_tonnes
avg_temp
0
100000
200000
300000
400000
500000


Correlation Matrix
Year
yield
average_rain_fall_mm_per_year
pesticides_tonnes
avg_temp
1.00 0.09 -0.01 0.14 0.01
0.09 1.00 -0.00 0.07 -0.11
-0.01 -0.00 1.00 0.16 0.33
0.14 0.07 0.16 1.00 0.05
0.01 -0.11 0.33 0.05 1.00
1.0
0.8
0.6
0.4
0.2
0.0


Pairplot of Numerical Features
Year
yield
average_rain_fall_mm_per_year
pesticides_tonnes
avg_temp


Yield vs Average Rainfall
yield
average_rain_fall_mm_per_year
0
100000
200000
300000
400000
500000
0
500
1000
1500
2000
2500
3000


Yield vs Average Temperature
yield
avg_temp
0
100000
200000
300000
400000
500000
0
5
10
15
20
25
30


Yield Distribution by Crop Type
yield
Item
0
100000
200000
300000
400000
500000
Maize
Potatoes
Rice, paddy
Sorghum
Soybeans
Wheat
Cassava
Sweet potatoes
Plantains and others
Yams

5. Feature Engineering:

Feature engineering was applied to enhance model performance by transforming existing features:

- **One-Hot Encoding**: Categorical variables like 'Area' and 'Item' were converted into numerical format using one-hot encoding to make them compatible with machine learning models.
- **Normalization**: Numerical features such as temperature and rainfall were scaled using standardization (mean=0, std=1) to ensure that all features contributed equally to the model

```
# Preprocessing categorical and numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore', drop='first'), categorical_cols)
    ])

X_preprocessed = preprocessor.fit_transform(X)
```

6. Data Transformation:

Before model training, data transformation techniques were applied:

- **Standard Scaling**: Applied to numerical columns to normalize the data.
- **One-Hot Encoding**: Used for categorical variables to avoid bias towards any particular category.
- The transformed dataset was split into training (80%) and testing (20%) sets to evaluate model performance on unseen data.

MODEL EXPLORATION

1. Model Selection:

Given the nature of the problem (regression) and the large dataset, a **neural network model** was chosen for the project. The model's ability to capture complex relationships between input features and the target variable makes it well-suited for this task. Additionally, neural networks can handle large datasets efficiently.

Strengths:

- Neural networks excel at modeling non-linear relationships.
- Flexible architecture allows customization for feature-rich datasets.

Weaknesses:

- Neural networks require more computational resources and time to train.
- They are prone to overfitting if not properly regularized.

2. Model Training:

The neural network was implemented using **TensorFlow** and **Keras**. The architecture consisted of:

- **Input layer**: Matching the number of features.
- **Hidden layers**: Three hidden layers with 128, 64, and 32 neurons respectively, all using the ReLU activation function.
- **Output layer**: A single neuron for regression output (yield prediction).

Hyperparameters:

- **Optimizer**: Adam optimizer for efficient learning.
- **Loss function**: Mean Squared Error (MSE) was used for regression.
- **Batch size**: 32 samples per batch.
- **Epochs**: 50 epochs for training the model.

```
# Building the neural network
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)  # Output layer for regression
])

# Compiling the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Training the model
history = model.fit(X_train_scaled, y_train, epochs=50, validation_split=0.2,
batch_size=32)
```

3. Model Evaluation:

The model was evaluated using two primary metrics:

- **Mean Squared Error (MSE)**: To measure the average squared difference between the predicted and actual values.
- **R-squared (R^2)**: To determine the proportion of variance explained by the model.

```
# Making predictions on test data
y_pred = model.predict(X_test_scaled)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

4. Code Implementation:

The code for data preparation, feature engineering, and model exploration has been included in snippets throughout the report. Below is an excerpt for model training and evaluation:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2,
random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model architecture
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))

# Compile and train
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model.fit(X_train_scaled, y_train, epochs=50, validation_split=0.2,
batch_size=32)

# Evaluate
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```