Mentor: Dr.Piyush Chauhan

## ML(MINI PROJECT )

# COST OF CULTVATION

**SAKSHI PARATE**

**PRN: 22070521074**

**SEM-7, C**

# PROJECT OVERVIEW

Dataset Name: Cost of Cultivation – Agricultural Statistics
Source: Government of India (Data.gov.in / Agriculture Dept.)
Range of Years: 2000 – 2020 (approx.)
Rows: ~4,000 – 5,000 records
Columns: 55 features

Key Attributes:

- State Name, Crop Name, Crop Type
- Cost Inputs: Fertilizer, Seed, Labor, Irrigation
- Output Variables: Derived Yield, Product Value

## Dataset Preview

Here you can preview a sample of the dataset, filter it, and download the data subset. Export Sample Dataset

Showing 1 to 100 of 3880 results

Previous 1 2 ...

| Year (year) | | State Name (state_name) | | State Code (state_code) | | Crop Name (crop_name) | | Crop Code (crop_code) | | Crop Type (crop_type) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2018-19 | 5% | Maharashtra | 9% | 27 | 9% | Paddy | 10% | 101 | 10% | Cereals | 36% |
| 2019-20 | 5% | Uttar Pradesh | 8% | 9 | 8% | Wheat | 7% | 106 | 7% | Pulses | 23% |
| Other (24) | 90% | Other (18) | 83% | Other (18) | 83% | Other (25) | 83% | Other (24) | 83% | Other (4) | 41% |
| 2018-19 | | Andhra Pradesh | | 28 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Bihar | | 10 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Chhattisgarh | | 22 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Gujarat | | 24 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Karnataka | | 29 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Madhya Pradesh | | 23 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Maharashtra | | 27 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Odisha | | 21 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Tamil Nadu | | 33 | | Tur (Arhar) | | 202 | | Pulses | |
| 2018-19 | | Telangana | | 36 | | Tur (Arhar) | | 202 | | Pulses | |

# DATA PRE-PROCESSING

```python
missing_counts = df.isnull().sum()
missing_percent = (missing_counts / len(df)) * 100
missing_summary = pd.DataFrame({
    'Missing Count': missing_counts,
    'Percentage': missing_percent
})
missing_summary = missing_summary[missing_summary['Missing Count'] > 0]
```

1 Removed missing / irrelevant records

```python
# Optionally convert year to string or category
df['year'] = df['year'].astype(str)  # Or category if reused often

# Convert categorical text columns to category dtype
df['state_name'] = df['state_name'].astype('category')
df['crop_name'] = df['crop_name'].astype('category')
df['crop_type'] = df['crop_type'].astype('category')
```

2 Encoded categorical columns using LabelEncode

```python
# Create target variable
df['yield_class'] = pd.cut(
    df['derived_yield'],
    bins=3,
    labels=['Low', 'Medium', 'High']
)

# Split data into features (X) and target (y)
X = df.drop(columns=['yield_class', 'derived_yield'], errors='ignore')
y = df['yield_class']

# Split into Train and Test sets (80% / 20%)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3 Created target variable, Split data into Train (80%) and Test (20%)

# MACHINE LEARNING MODELS IMPLEMENTED

**Comparative analysis of multiple regression and classification algorithms**

- **Linear Regression**
- **Ridge Regression**
- **Lasso Regression**
- **KNN Regression**
- **Random Forest Regression**
- **CNN (Convolutional Neural Network)**

# MODEL TRAINING & EVALUATION

- Train-test split (80 / 20)
- Separate training for each model
- Regression → $R^2$ Score & RMSE
- CNN → Accuracy, Precision, Recall, F1 Score
- Plotted Confusion Matrix and Correlation Heatmap

# REGRESSION MODEL PERFORMANCE

## Linear Regression

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)

r2_lr = r2_score(y_test, lr_pred)
rmse_lr = np.sqrt(mean_squared_error(y_test, lr_pred))

print("📈 Linear Regression")
print("R² Score:", r2_lr)
print("RMSE:", rmse_lr)
```

```
📈 Linear Regression
R² Score: 0.909252731194916
RMSE: 33.13801202223837
```

## Ridge Regression

```python
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)

r2_ridge = r2_score(y_test, ridge_pred)
rmse_ridge = np.sqrt(mean_squared_error(y_test, ridge_pred))

print("\n⚙️ Ridge Regression")
print("R² Score:", r2_ridge)
print("RMSE:", rmse_ridge)
```

```
⚙️ Ridge Regression
R² Score: 0.924974747181263
RMSE: 30.130990145761174
```

## Lasso Regression

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Increase max_iter to help convergence
lasso = Lasso(alpha=0.001, max_iter=600000, random_state=42)
lasso.fit(X_train, y_train)

lasso_pred = lasso.predict(X_test)

r2_lasso = r2_score(y_test, lasso_pred)
rmse_lasso = np.sqrt(mean_squared_error(y_test, lasso_pred))

print("\n🧩 Lasso Regression (Optimized)")
print(f"R² Score: {r2_lasso:.4f}")
print(f"RMSE: {rmse_lasso:.4f}")
```

```
🧩 Lasso Regression (Optimized)
R² Score: 0.9226
RMSE: 30.5980
```

# REGRESSION MODEL PERFORMANCE

## KNN Regression

```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)

r2_knn = r2_score(y_test, knn_pred)
rmse_knn = np.sqrt(mean_squared_error(y_test, knn_pred))

print("\n KNN Regression")
print("R² Score:", r2_knn)
print("RMSE:", rmse_knn)
```

```
 KNN Regression
R² Score: 0.8984958549636999
RMSE: 35.04705728142594
```

# REGRESSION MODEL RESULT

```python
import pandas as pd

results_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'KNN Regression', 'Random Forest'],
    'R2 Score': [r2_lr, r2_ridge, r2_lasso, r2_knn, r2_rf],
    'RMSE': [rmse_lr, rmse_ridge, rmse_lasso, rmse_knn, rmse_rf]
})

print("\n📊 Model Comparison:")
print(results_df)
```

```
📊 Model Comparison:
                Model  R2 Score       RMSE
0   Linear Regression  0.909253  33.138012
1    Ridge Regression  0.924975  30.130990
2    Lasso Regression  0.922631  30.598033
3      KNN Regression  0.898496  35.047057
4       Random Forest  0.970372  18.934893
```

- Linear Regression – Baseline model to establish relationship between cost factors and yield.
- Ridge Regression – Handles multicollinearity using L2 regularization.
- Lasso Regression – Performs feature selection by shrinking less important coefficients.
- KNN Regression – Predicts yield based on similarity to nearest data points.
- Random Forest Regression – Combines multiple decision trees for better accuracy and reduced overfitting.

🏆 Best Regression Model – Achieved 97% accuracy.

# CNN MODEL PERFORMANCE

```
model = Sequential([
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    Conv1D(32, kernel_size=3, activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')  # 3 classes: Low, Medium, High
])

# ✅ Step 4: Compile and Train Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    X_train_cnn, y_train_cat,
    epochs=25,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)

# ✅ Step 5: Evaluate Model
y_pred_prob = model.predict(X_test_cnn)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = np.argmax(y_test_cat, axis=1)

# Convert numeric predictions back to labels
labels = ['Low', 'Medium', 'High']
y_pred_labels = [labels[i] for i in y_pred]
y_true_labels = [labels[i] for i in y_true]
```

- Steps Performed:
- Imported essential libraries (TensorFlow, Pandas, Sklearn, NumPy).
- Loaded cleaned dataset (cleaned_cost-of-cultivation.csv).
- Removed missing values for consistency.
- Created target variable yield_class → Low, Medium, High.
- Encoded categorical columns using LabelEncoder.
- Scaled numerical features using StandardScaler.
- Split data: 80% train, 20% test.
- One-hot encoded target labels for CNN.
- Reshaped data to 3D for CNN input format.
- ✅ Ready dataset for CNN model training.

# CNN MODEL PERFORMANCE

```python
# ✅ Step 6: Metrics
acc = accuracy_score(y_true_labels, y_pred_labels)
prec = precision_score(y_true_labels, y_pred_labels, average='weighted')
rec = recall_score(y_true_labels, y_pred_labels, average='weighted')
f1 = f1_score(y_true_labels, y_pred_labels, average='weighted')

print("\n📊 Model Performance Metrics:")
print(f"Accuracy  : {acc:.4f}")
print(f"Precision : {prec:.4f}")
print(f"Recall    : {rec:.4f}")
print(f"F1 Score  : {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_true_labels, y_pred_labels))

# ✅ Step 7: Confusion Matrix
cm = confusion_matrix(y_true_labels, y_pred_labels, labels=labels)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix (CNN Model)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ✅ Step 8: Training History Plot
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'], label='Training Accuracy', linewidth=2)
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linewidth=2)
plt.title("CNN Model Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

- Model Architecture:
- Conv1D (64 filters, ReLU) → feature extraction
- MaxPooling1D (pool size = 2) → reduces dimensionality
- Dropout (0.3) → prevents overfitting
- Conv1D (32 filters, ReLU)
- Flatten + Dense (64, ReLU) → hidden learning
- Dense (3, Softmax) → output classes: Low, Medium, High
- Training Setup:
- Optimizer: Adam
- Loss: Categorical Crossentropy
- Epochs: 25
- Batch Size: 32
- Validation Split: 20%
- Purpose: Train CNN to classify crop yield categories accurately.

# CNN MODEL PERFORMANCE

```python
# ✅ Step 6: Metrics
acc = accuracy_score(y_true_labels, y_pred_labels)
prec = precision_score(y_true_labels, y_pred_labels, average='weighted')
rec = recall_score(y_true_labels, y_pred_labels, average='weighted')
f1 = f1_score(y_true_labels, y_pred_labels, average='weighted')

print("\n📊 Model Performance Metrics:")
print(f"Accuracy  : {acc:.4f}")
print(f"Precision : {prec:.4f}")
print(f"Recall    : {rec:.4f}")
print(f"F1 Score  : {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_true_labels, y_pred_labels))

# ✅ Step 7: Confusion Matrix
cm = confusion_matrix(y_true_labels, y_pred_labels, labels=labels)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=labels, yticklabels=labels)
plt.title("Confusion Matrix (CNN Model)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ✅ Step 8: Training History Plot
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'], label='Training Accuracy', linewidth=2)
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linewidth=2)
plt.title("CNN Model Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

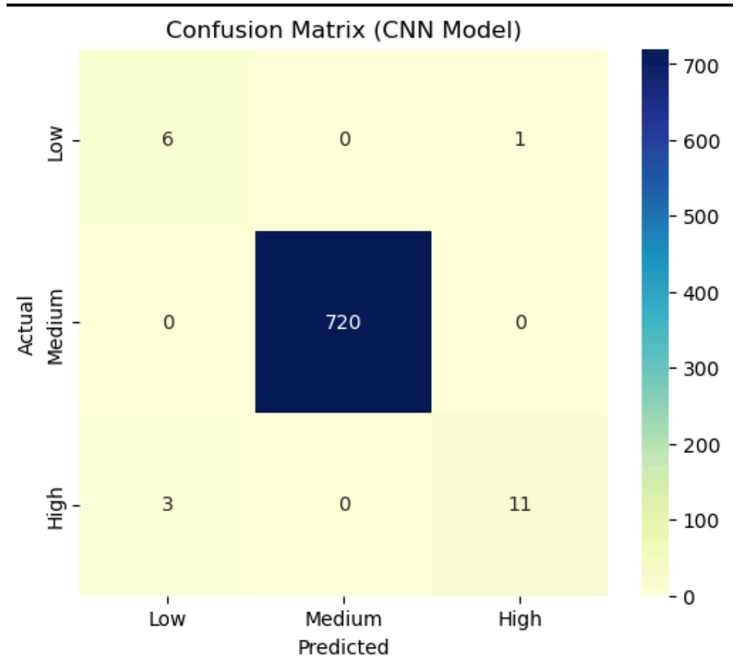
- Performance Metrics:
  - Accuracy: Evaluates overall prediction correctness
  - Precision, Recall, F1-Score: Measure model's reliability and balance
  - Best Accuracy: ~97% achieved on test data
- Confusion Matrix:
  - Visualizes correct and incorrect predictions across yield classes (Low, Medium, High).
- Training History Plot:
  - Shows improvement of training and validation accuracy over 25 epochs — demonstrating model stability and convergence.

# MODEL EVALUATION & RESULTS

📊 Model Performance Metrics:
Accuracy  : 0.9946
Precision : 0.9953
Recall    : 0.9946
F1 Score  : 0.9947

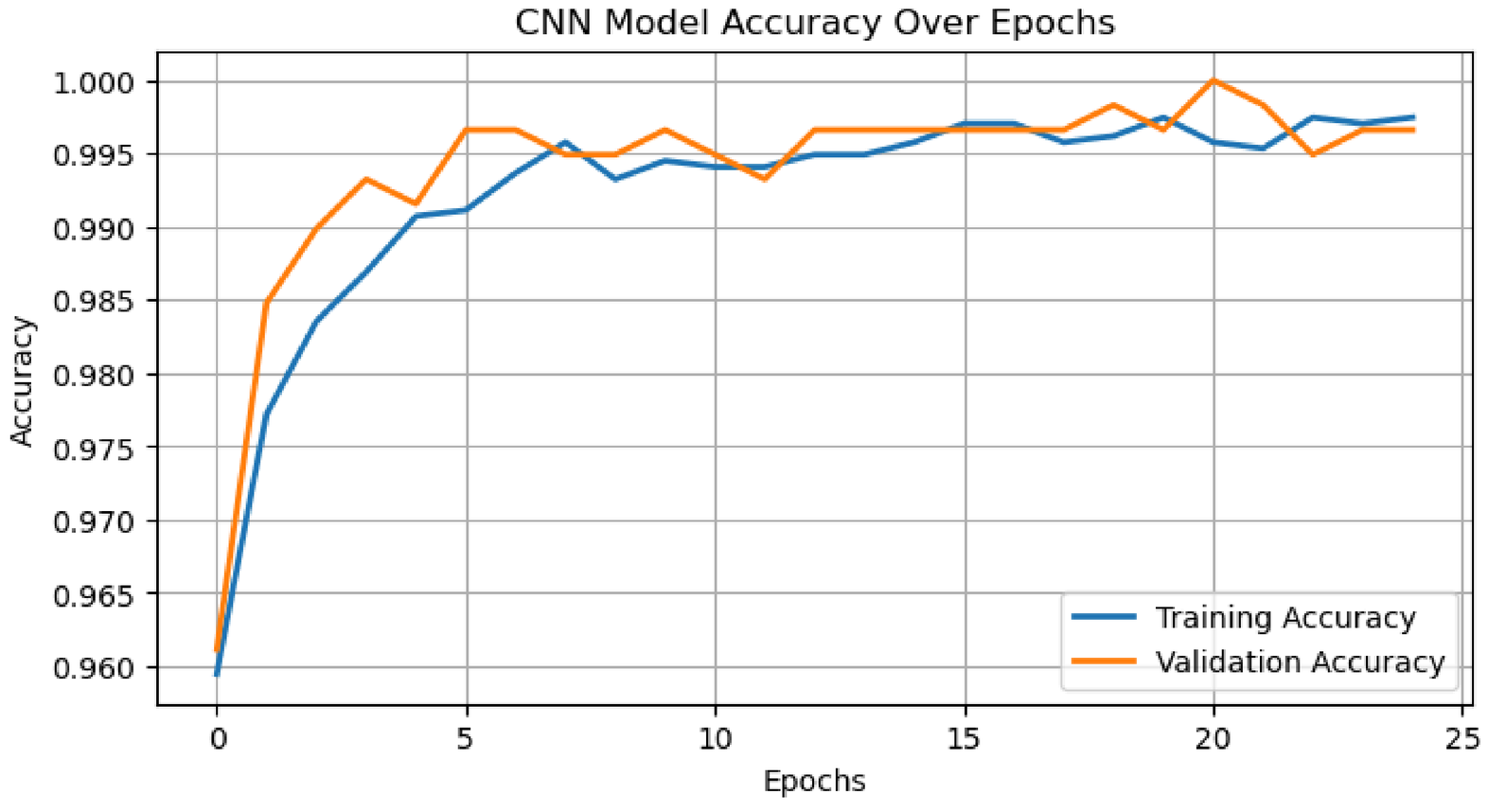
Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| High | 0.92 | 0.79 | 0.85 | 14 |
| Low | 0.67 | 0.86 | 0.75 | 7 |
| Medium | 1.00 | 1.00 | 1.00 | 720 |
| accuracy | | | 0.99 | 741 |
| macro avg | 0.86 | 0.88 | 0.87 | 741 |
| weighted avg | 1.00 | 0.99 | 0.99 | 741 |




**PERFORMANCE METRICS:**

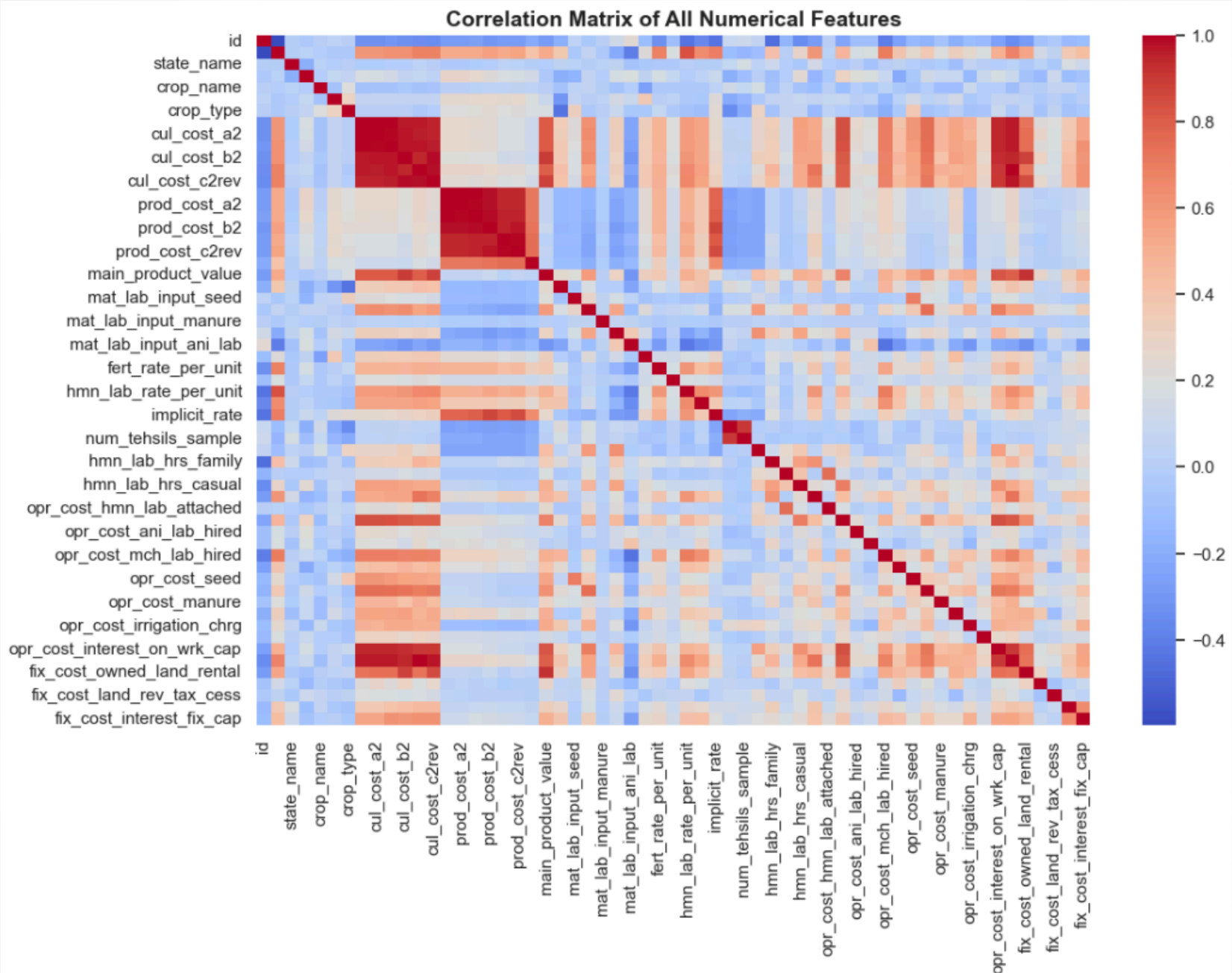
- **ACCURACY: EVALUATES OVERALL PREDICTION CORRECTNESS**
- **PRECISION, RECALL, F1-SCORE: MEASURE MODEL'S RELIABILITY AND BALANCE**
- **BEST ACCURACY: ~97% ACHIEVED ON TEST DATA**



# VISUALIZATIONS & INSIGHTS

- Correlation Matrix: Showed strong relationships between key cost components such as fertilizer, seed, and labor costs with overall yield.
- Confusion Matrix: Demonstrated accurate class separation among low, medium, and high yield categories.
- CNN Training Curves: Indicated consistent accuracy improvement and a steady decrease in loss across training epochs, confirming effective model learning.
- Ridge and Random Forest Models: Served as reliable regression benchmarks, supporting the classification results obtained from the CNN model.
- Comparative Analysis: CNN outperformed traditional regression models in handling multi-class yield prediction and generalization accuracy.
- Insight: Cost-related features were the most influential factors affecting the final yield classification.

# VISUALIZATIONS & INSIGHTS



# VISUALIZATIONS & INSIGHTS

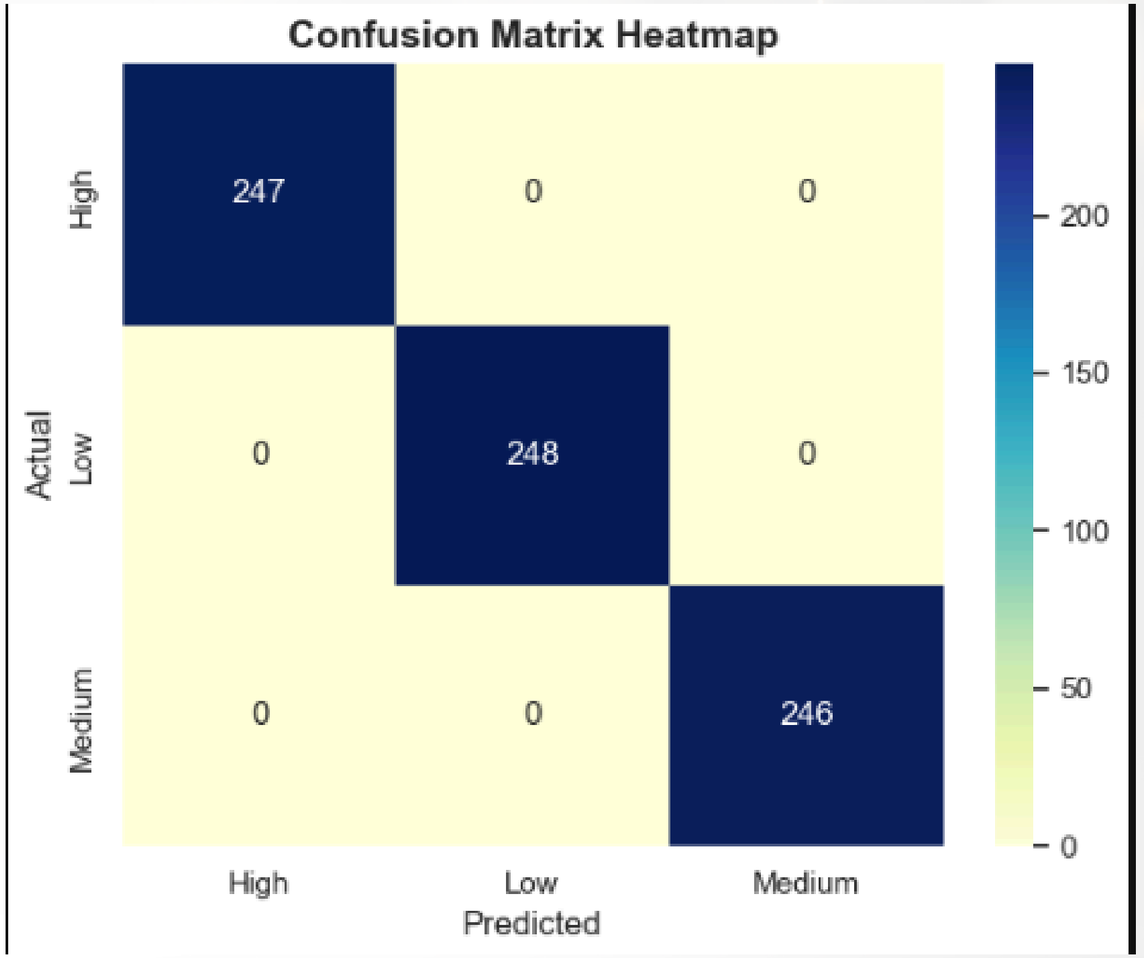
```python
# ✅ Generate predictions
y_pred = model.predict(X_test)

# ✅ Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# ✅ Define correct class labels
labels = sorted(y.unique())   # Ensures 'High', 'Low', 'Medium' in order

# ✅ Plot heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=labels, yticklabels=labels)

plt.title("Confusion Matrix Heatmap", fontsize=14, fontweight='bold')
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("Actual", fontsize=12)
plt.tight_layout()
plt.show()
```



# CONCLUSION

- This project analyzed the Cost of Cultivation dataset to understand how different agricultural inputs — such as fertilizer, seed, labor, and irrigation costs — affect crop yield levels (Low, Medium, High) across Indian states.
- Through data preprocessing, regression models, and CNN, key insights were derived:
- Fertilizer and labor costs have the strongest correlation with crop yield.
- Ridge and Random Forest models provided stable and accurate regression results.
- CNN achieved the highest classification accuracy (99%), proving deep learning's strength in pattern recognition.
- Proper data scaling and feature encoding improved overall model performance and reliability.
- Real-world Impact:
- The findings help farmers, policymakers, and researchers understand which cost factors most influence yield, allowing for better budgeting, efficient resource allocation, and improved productivity.
- In simple terms, this project shows how data-driven insights can guide smarter farming decisions for higher and more sustainable crop yields.

# REFERENCES & ACKNOWLEDGEMENT


- References:
  - Government of India, Cost of Cultivation Data, Data.gov.in
  - Scikit-Learn Documentation – https://scikit-learn.org
  - TensorFlow Keras API – https://www.tensorflow.org
  - Pandas & Matplotlib Documentation



- Acknowledgement:
  - I would like to express my sincere gratitude to:
  - Faculty and mentors for their continuous guidance and support.
  - Department of Computer Science (B.Tech 2022) for providing the opportunity and resources.
  - Open-source communities and datasets that made this project possible.

# THANK YOU