The information provided is quite comprehensive for generating a Python pipeline code for the described research problem. Below is a structured Python code that follows the proposed pipeline for automating and improving crop yield predictions using a deep learning approach.

### Python Code for Crop Yield Prediction Pipeline

```python

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

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

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

import matplotlib.pyplot as plt

# Step 1: Data Preprocessing

def load\_and\_preprocess\_data(file\_path):

# Load the dataset

data = pd.read\_csv(file\_path)

# Handle missing values (imputation)

data.fillna(data.mean(), inplace=True)

# Remove duplicates

data.drop\_duplicates(inplace=True)

# Identify and treat outliers (using Z-score method)

from scipy import stats

z\_scores = np.abs(stats.zscore(data.select\_dtypes(include=[np.number])))

data = data[(z\_scores < 3).all(axis=1)]

# Normalize numerical features

scaler = StandardScaler()

numerical\_features = data.select\_dtypes(include=[np.number])

data[numerical\_features.columns] = scaler.fit\_transform(numerical\_features)

return data

# Step 2: Model Selection and Architecture

def build\_model(input\_shape):

model = keras.Sequential()

model.add(layers.Input(shape=input\_shape))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1)) # Output layer for yield prediction

model.compile(optimizer='adam', loss='mean\_squared\_error')

return model

# Step 3: Main function to execute the pipeline

def main():

# Load and preprocess data

file\_path = 'path/to/your/crop\_yield\_data.csv' # Update with your file path

data = load\_and\_preprocess\_data(file\_path)

# Split data into features and target variable

X = data.drop('yield', axis=1) # Assuming 'yield' is the target variable

y = data['yield']

# Split into training and testing sets

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

# Build the model

model = build\_model(input\_shape=(X\_train.shape[1],))

# Step 4: Model Training

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_split=0.2)

# Step 5: Evaluation

y\_pred = model.predict(X\_test)

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

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f'R²: {r2:.4f}')

print(f'Mean Absolute Error: {mae:.4f}')

print(f'Root Mean Square Error: {rmse:.4f}')

# Plot training history

plt.plot(history.history['loss'], label='train')

plt.plot(history.history['val\_loss'], label='validation')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend()

plt.show()

# Step 6: Deployment (Placeholder for deployment code)

def deploy\_model(model):

# Code for deployment goes here (e.g., saving the model, creating an API, etc.)

model.save('crop\_yield\_model.h5')

if \_\_name\_\_ == "\_\_main\_\_":

main()

```

### Notes:

1. \*\*File Path:\*\* Update the `file\_path` variable with the actual path to your CSV file.

2. \*\*Outlier Treatment:\*\* The outlier treatment method used here is based on the Z-score. You may want to adjust this based on your specific dataset and requirements.

3. \*\*Model Training:\*\* The number of epochs and batch size can be adjusted based on the performance and available resources.

4. \*\*Deployment:\*\* The deployment function is a placeholder. You can implement it based on your deployment strategy (e.g., saving the model, creating a REST API, etc.).

This code provides a complete pipeline from data preprocessing to model evaluation, following the steps outlined in your proposed pipeline. If you need any further modifications or additional features, feel free to ask!