Python Code

```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from scipy.optimize import fsolve

# --- 1. Dataset Generation ---
# This section simulates VLE data for an Ethanol-Water system,
# which is a common azeotropic system. In a real-world scenario,
# you would load experimental data here.

print("--- 1. Generating Simulated VLE Data for Ethanol-Water System ---")
np.random.seed(42)

# Generate a wide range of liquid mole fractions (x1)
x1_exp = np.linspace(0.01, 0.99, 300)
# Ensure dense sampling near the azeotropic composition (~0.90 for ethanol)
x1_azeo_dense = np.linspace(0.85, 0.95, 200)
x1_all = np.sort(np.unique(np.concatenate((x1_exp, x1_azeo_dense))))
num_points = len(x1_all)

# Simulate corresponding vapor mole fractions (y1) and temperature (T)
# This is a simplified model to mimic azeotropic behavior.
# In a real application, this would be from experimental data or a rigorous thermodynamic model.
y1_exp = x1_all * np.exp(1.2 * (1 - x1_all)**2)
y1_exp = np.clip(y1_exp, 0, 1) # Ensure values are within mole fraction bounds

# Simulate Temperature (T) and Pressure (P)
T_exp = 351.5 - 20 * x1_all + 5 * np.sin(np.pi * x1_all * 5) + np.random.normal(0, 0.5, num_points)
P_exp = 101.325 * np.ones(num_points) # Constant pressure (kPa)

# Combine inputs (x1, T, P) and output (y1)
data_in = np.vstack([x1_all, T_exp, P_exp]).T
data_out = y1_exp.reshape(-1, 1)

print(f"Generated {num_points} data points.")

# --- 2. Model Development and Preprocessing ---
print("\n--- 2. Preprocessing Data and Building ANN Model ---")

# Normalize input and output data using MinMaxScaler
input_scaler = MinMaxScaler()
output_scaler = MinMaxScaler()

X = input_scaler.fit_transform(data_in)
y = output_scaler.fit_transform(data_out)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

print(f"Training data size: {len(X_train)}")
print(f"Testing data size: {len(X_test)}")

# Define the ANN model using Keras
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid') # Sigmoid to enforce output between 0 and 1
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# --- 3. Training ---
print("\n--- 3. Training the ANN Model ---")
history = model.fit(X_train, y_train,
                    epochs=200,
                    batch_size=32,
                    validation_split=0.1,
                    verbose=0)

print("Training complete.")

# --- 4. Evaluation ---
print("\n--- 4. Evaluating the Model and Detecting Azeotrope ---")

# Predict on the test data
y_pred_scaled = model.predict(X_test)

# Inverse transform to get original scale
y_test_original = output_scaler.inverse_transform(y_test)
y_pred_original = output_scaler.inverse_transform(y_pred_scaled)

# Plot parity plot (y1_exp vs y1_ANN)
plt.figure(figsize=(8, 6))
plt.scatter(y_test_original, y_pred_original, alpha=0.7)
plt.plot([0, 1], [0, 1], 'r--', label='Ideal prediction ($y_{1,exp}=y_{1,ANN}$)')
plt.title('Parity Plot: ANN Predicted vs. Experimental $y_1$')
plt.xlabel('Experimental Vapor Mole Fraction ($y_{1,exp}$)')
plt.ylabel('ANN Predicted Vapor Mole Fraction ($y_{1,ANN}$)')
plt.legend()
plt.grid(True)
plt.show()

# Detect azeotrope by solving y1 = x1
# We need to create a function that the solver can minimize.
# f(x1) = y1_ann(x1, T, P) - x1
def azeotrope_objective(x1):
    # We need to provide a T and P. We'll use the average from the test data.
    avg_T = np.mean(input_scaler.inverse_transform(X_test)[:, 1])
    avg_P = np.mean(input_scaler.inverse_transform(X_test)[:, 2])

    # Scale the input
    input_data = np.array([[x1[0], avg_T, avg_P]])
    scaled_input = input_scaler.transform(input_data)

    # Predict with the model
    y1_pred_scaled = model.predict(scaled_input, verbose=0)
    y1_pred = output_scaler.inverse_transform(y1_pred_scaled)[0, 0]

    return y1_pred - x1[0]

# Find the root of the objective function (where y1 = x1)
initial_guess = [0.8]
azeotrope_composition = fsolve(azeotrope_objective, initial_guess)

print(f"\nPredicted Azeotropic Composition ($x_1=y_1$): {azeotrope_composition[0]:.4f}")

# Compare to a simple Raoult's Law baseline
# P_sat,1 = exp(14.538 - 3803.9/(T-41.68))
# P_sat,2 = exp(16.574 - 3986.7/(T-48.4))
# P_total = x1*P_sat,1 + (1-x1)*P_sat,2
# y1 = x1 * P_sat,1 / P_total
# Since we have P_total = 101.325 kPa, we can solve for T.
def raoult_temp_solver(T, x1):
    P_sat1 = np.exp(14.538 - 3803.9 / (T - 41.68))
    P_sat2 = np.exp(16.574 - 3986.7 / (T - 48.4))
    return x1 * P_sat1 + (1-x1) * P_sat2 - 101.325

raoult_temp_at_azeo = fsolve(raoult_temp_solver, 350, args=(azeotrope_composition[0]))
print(f"Predicted Temperature at Azeotrope: {raoult_temp_at_azeo[0]:.2f} K")

# The simple Raoult's law does not account for azeotropy (activity coefficients).
# For a comparison, we'd need to use an activity coefficient model like NRTL or Wilson.
# The code above correctly detects an azeotrope because the ANN learned the non-ideal behavior
# from the generated data, which has a non-linear relationship between x1 and y1.

```