1. MNIST Digit Classification

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, precision_recall_fscore_support
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import pandas as pd
from itertools import product
import warnings
warnings.filterwarnings('ignore')

print("TensorFlow Version:", tf.__version__)
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

```
TensorFlow Version: 2.19.0
GPU Available: []
```

STEP 1: DATA **PREPROCESSING**

```
def load_and_preprocess_data():
    """Load and preprocess MNIST dataset"""
    print("\n" + "="*80)
    print("STEP 1: DATA PREPROCESSING")
    print("="*80)

    # Load MNIST dataset
    (X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

    print(f"Training data shape: {X_train.shape}")
    print(f"Training labels shape: {y_train.shape}")
    print(f"Test data shape: {X_test.shape}")
    print(f"Test labels shape: {y_test.shape}")

    # Normalize pixel values to [0, 1]
    X_train = X_train.astype('float32') / 255.0
    X_test = X_test.astype('float32') / 255.0
    print("\n✓ Normalized pixel values to range [0, 1]")

    # Flatten images from 28x28 to 784
    X_train_flat = X_train.reshape(-1, 784)
    X_test_flat = X_test.reshape(-1, 784)
    print(f"✓ Flattened images to shape: {X_train_flat.shape}")

    # One-hot encode labels
    y_train_cat = to_categorical(y_train, 10)
    y_test_cat = to_categorical(y_test, 10)
    print(f"✓ One-hot encoded labels to shape: {y_train_cat.shape}")

    # Create validation split
    split_idx = int(0.9 * len(X_train_flat))
    X_val = X_train_flat[split_idx:]
    y_val = y_train_cat[split_idx:]
    X_train_final = X_train_flat[:split_idx]
    y_train_final = y_train_cat[:split_idx]

    print(f"\n✓ Created validation split:")
    print(f"  Training: {X_train_final.shape[0]} samples")
    print(f"  Validation: {X_val.shape[0]} samples")
    print(f"  Test: {X_test_flat.shape[0]} samples")

    return X_train_final, y_train_final, X_val, y_val, X_test_flat, y_test_cat, y_test
```

STEP 2: MODEL BUILDING FUNCTIONS

```
def build_model(num_layers=2, neurons_per_layer=128, dropout_rate=0.2,
                activation='relu', learning_rate=0.001):
```

```
    """
    Build a feedforward neural network with configurable architecture

    Parameters:
    - num_layers: Number of hidden layers (1-3)
    - neurons_per_layer: Number of neurons in each hidden layer
    - dropout_rate: Dropout rate for regularization (0.0-0.5)
    - activation: Activation function ('relu', 'sigmoid', 'tanh')
    - learning_rate: Learning rate for optimizer
    """
    model = models.Sequential()

    # Input layer (flattened)
    model.add(layers.Input(shape=(784,)))

    # Hidden layers with dropout
    for i in range(num_layers):
        model.add(layers.Dense(neurons_per_layer, activation=activation,
                              name=f'hidden_{i+1}'))
        model.add(layers.Dropout(dropout_rate, name=f'dropout_{i+1}'))

    # Output layer with softmax
    model.add(layers.Dense(10, activation='softmax', name='output'))

    # Compile model
    optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])

    return model
```

STEP 3-7: HYPERPARAMETER EXPERIMENTATION

```
def experiment_hyperparameters(X_train, y_train, X_val, y_val):
    """Comprehensive hyperparameter tuning experiments"""
    print("\n" + "="*80)
    print("STEP 3-7: HYPERPARAMETER EXPERIMENTATION")
    print("="*80)

    results = []

    # Define hyperparameter grid
    experiments = [
        # Experiment 1: Number of layers and neurons
        {'name': 'Architecture_1Layer_128', 'num_layers': 1, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Architecture_2Layers_128', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Architecture_2Layers_64', 'num_layers': 2, 'neurons': 64,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Architecture_3Layers_128', 'num_layers': 3, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},

        # Experiment 2: Learning rates
        {'name': 'LR_0.01', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.01, 'batch_size': 128},
        {'name': 'LR_0.001', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'LR_0.0001', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.0001, 'batch_size': 128},

        # Experiment 3: Dropout rates
        {'name': 'Dropout_0.0', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.0, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Dropout_0.2', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Dropout_0.4', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.4, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},

        # Experiment 4: Activation functions
        {'name': 'Activation_ReLU', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
        {'name': 'Activation_Sigmoid', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'sigmoid', 'lr': 0.001, 'batch_size': 128},
```

```
        {'name': 'Activation_Tanh', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'tanh', 'lr': 0.001, 'batch_size': 128},

        # Experiment 5: Batch sizes
        {'name': 'Batch_32', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 32},
        {'name': 'Batch_64', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 64},
        {'name': 'Batch_128', 'num_layers': 2, 'neurons': 128,
         'dropout': 0.2, 'activation': 'relu', 'lr': 0.001, 'batch_size': 128},
    ]

    histories = {}

    for i, exp in enumerate(experiments, 1):
        print(f"\n[{i}/{len(experiments)}] Running experiment: {exp['name']}")
        print("-" * 60)

        # Build model
        model = build_model(
            num_layers=exp['num_layers'],
            neurons_per_layer=exp['neurons'],
            dropout_rate=exp['dropout'],
            activation=exp['activation'],
            learning_rate=exp['lr']
        )

        # Train model
        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=20,
            batch_size=exp['batch_size'],
            verbose=0
        )

        # Store results
        final_train_acc = history.history['accuracy'][-1]
        final_val_acc = history.history['val_accuracy'][-1]
        final_train_loss = history.history['loss'][-1]
        final_val_loss = history.history['val_loss'][-1]

        results.append({
            'Experiment': exp['name'],
            'Layers': exp['num_layers'],
            'Neurons': exp['neurons'],
            'Dropout': exp['dropout'],
            'Activation': exp['activation'],
            'Learning_Rate': exp['lr'],
            'Batch_Size': exp['batch_size'],
            'Train_Acc': final_train_acc,
            'Val_Acc': final_val_acc,
            'Train_Loss': final_train_loss,
            'Val_Loss': final_val_loss,
            'Overfitting': final_train_acc - final_val_acc
        })

        histories[exp['name']] = history

        print(f"  Train Acc: {final_train_acc:.4f} | Val Acc: {final_val_acc:.4f}")
        print(f"  Train Loss: {final_train_loss:.4f} | Val Loss: {final_val_loss:.4f}")

    # Create results DataFrame
    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values('Val_Acc', ascending=False)

    print("\n" + "="*80)
    print("EXPERIMENT RESULTS SUMMARY")
    print("="*80)
    print(results_df.to_string(index=False))

    return results_df, histories
```

STEP 8: BUILD AND TRAIN OPTIMAL MODEL

```
def train_optimal_model(X_train, y_train, X_val, y_val, X_test, y_test_cat):
    """Train the optimal model based on best hyperparameters"""
    print("\n" + "="*80)
    print("STEP 8: TRAINING OPTIMAL MODEL")
    print("="*80)

    # Best hyperparameters (adjust based on experiments)
    best_params = {
        'num_layers': 2,
        'neurons_per_layer': 128,
        'dropout_rate': 0.2,
        'activation': 'relu',
        'learning_rate': 0.001
    }

    print("\nOptimal Hyperparameters:")
    for key, value in best_params.items():
        print(f"  {key}: {value}")

    # Build optimal model
    model = build_model(**best_params)

    print("\n")
    model.summary()

    # Callbacks
    early_stop = EarlyStopping(monitor='val_loss', patience=5,
                              restore_best_weights=True, verbose=1)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                                 patience=3, min_lr=1e-6, verbose=1)

    # Train model
    print("\nTraining optimal model...")
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=50,
        batch_size=128,
        callbacks=[early_stop, reduce_lr],
        verbose=1
    )

    # Evaluate on test set
    test_loss, test_acc = model.evaluate(X_test, y_test_cat, verbose=0)
    print(f"\n{'='*80}")
    print(f"FINAL TEST ACCURACY: {test_acc*100:.2f}%")
    print(f"FINAL TEST LOSS: {test_loss:.4f}")
    print(f"{'='*80}")

    return model, history
```

STEP 9: PERFORMANCE METRICS AND VISUALIZATION

```
def evaluate_model(model, X_test, y_test_cat, y_test):
    """Comprehensive model evaluation with metrics and visualizations"""
    print("\n" + "="*80)
    print("PERFORMANCE METRICS")
    print("="*80)

    # Predictions
    y_pred = model.predict(X_test, verbose=0)
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred_classes)

    # Classification Report
    report = classification_report(y_test, y_pred_classes,
                                   target_names=[str(i) for i in range(10)])
    print("\nClassification Report:")
    print(report)

    # Calculate metrics
    precision, recall, f1, support = precision_recall_fscore_support(
        y_test, y_pred_classes, average='weighted'
```

```
    )

    print(f"\nWeighted Metrics:")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1-Score: {f1:.4f}")

    return cm, y_pred_classes

def plot_results(history, cm, results_df, histories):
    """Create comprehensive visualization plots"""

    # Create figure with multiple subplots
    fig = plt.figure(figsize=(20, 12))

    # 1. Training History - Accuracy
    ax1 = plt.subplot(2, 4, 1)
    ax1.plot(history.history['accuracy'], label='Train Accuracy', linewidth=2)
    ax1.plot(history.history['val_accuracy'], label='Val Accuracy', linewidth=2)
    ax1.set_title('Model Accuracy (Optimal Model)', fontsize=12, fontweight='bold')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # 2. Training History - Loss
    ax2 = plt.subplot(2, 4, 2)
    ax2.plot(history.history['loss'], label='Train Loss', linewidth=2)
    ax2.plot(history.history['val_loss'], label='Val Loss', linewidth=2)
    ax2.set_title('Model Loss (Optimal Model)', fontsize=12, fontweight='bold')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # 3. Confusion Matrix
    ax3 = plt.subplot(2, 4, 3)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax3, cbar=True)
    ax3.set_title('Confusion Matrix', fontsize=12, fontweight='bold')
    ax3.set_xlabel('Predicted Label')
    ax3.set_ylabel('True Label')

    # 4. Per-class Accuracy
    ax4 = plt.subplot(2, 4, 4)
    class_acc = cm.diagonal() / cm.sum(axis=1)
    ax4.bar(range(10), class_acc, color='steelblue', alpha=0.7)
    ax4.set_title('Per-Class Accuracy', fontsize=12, fontweight='bold')
    ax4.set_xlabel('Digit Class')
    ax4.set_ylabel('Accuracy')
    ax4.set_xticks(range(10))
    ax4.grid(True, alpha=0.3, axis='y')

    # 5. Learning Rate Comparison
    ax5 = plt.subplot(2, 4, 5)
    lr_data = results_df[results_df['Experiment'].str.contains('LR_')]
    ax5.bar(lr_data['Experiment'].str.replace('LR_', ''),
            lr_data['Val_Acc'], color='coral', alpha=0.7)
    ax5.set_title('Learning Rate Impact', fontsize=12, fontweight='bold')
    ax5.set_xlabel('Learning Rate')
    ax5.set_ylabel('Validation Accuracy')
    ax5.tick_params(axis='x', rotation=45)
    ax5.grid(True, alpha=0.3, axis='y')

    # 6. Dropout Comparison
    ax6 = plt.subplot(2, 4, 6)
    dropout_data = results_df[results_df['Experiment'].str.contains('Dropout_')]
    x_pos = np.arange(len(dropout_data))
    ax6.bar(x_pos - 0.2, dropout_data['Train_Acc'], width=0.4,
            label='Train Acc', color='lightblue', alpha=0.7)
    ax6.bar(x_pos + 0.2, dropout_data['Val_Acc'], width=0.4,
            label='Val Acc', color='darkblue', alpha=0.7)
    ax6.set_title('Dropout Rate Impact', fontsize=12, fontweight='bold')
    ax6.set_xlabel('Dropout Rate')
    ax6.set_ylabel('Accuracy')
    ax6.set_xticks(x_pos)
    ax6.set_xticklabels(dropout_data['Experiment'].str.replace('Dropout_', ''))
    ax6.legend()
```

```
    ax6.grid(True, alpha=0.3, axis='y')

    # 7. Activation Function Comparison
    ax7 = plt.subplot(2, 4, 7)
    act_data = results_df[results_df['Experiment'].str.contains('Activation_')]
    ax7.bar(act_data['Experiment'].str.replace('Activation_', ''),
            act_data['Val_Acc'], color='mediumseagreen', alpha=0.7)
    ax7.set_title('Activation Function Impact', fontsize=12, fontweight='bold')
    ax7.set_xlabel('Activation Function')
    ax7.set_ylabel('Validation Accuracy')
    ax7.grid(True, alpha=0.3, axis='y')

    # 8. Architecture Comparison
    ax8 = plt.subplot(2, 4, 8)
    arch_data = results_df[results_df['Experiment'].str.contains('Architecture_')]
    ax8.barh(range(len(arch_data)), arch_data['Val_Acc'], color='mediumpurple', alpha=0.7)
    ax8.set_title('Architecture Impact', fontsize=12, fontweight='bold')
    ax8.set_xlabel('Validation Accuracy')
    ax8.set_yticks(range(len(arch_data)))
    ax8.set_yticklabels(arch_data['Experiment'].str.replace('Architecture_', ''), fontsize=9)
    ax8.grid(True, alpha=0.3, axis='x')

    plt.tight_layout()
    plt.savefig('mnist_complete_analysis.png', dpi=300, bbox_inches='tight')
    print("\n✓ Visualization saved as 'mnist_complete_analysis.png'")
    plt.show()
```

MAIN EXECUTION

```
def main():
    """Main execution function"""
    print("\n" + "="*80)
    print("MNIST DIGIT CLASSIFICATION - COMPLETE DEEP LEARNING ASSIGNMENT")
    print("="*80)

    # Set random seeds for reproducibility
    np.random.seed(42)
    tf.random.set_seed(42)

    # Step 1: Load and preprocess data
    X_train, y_train, X_val, y_val, X_test, y_test_cat, y_test = load_and_preprocess_data()

    # Steps 3-7: Hyperparameter experimentation
    results_df, histories = experiment_hyperparameters(X_train, y_train, X_val, y_val)

    # Step 8: Train optimal model
    optimal_model, history = train_optimal_model(X_train, y_train, X_val, y_val,
                                                 X_test, y_test_cat)

    # Step 9: Evaluate model
    cm, y_pred = evaluate_model(optimal_model, X_test, y_test_cat, y_test)

    # Visualize results
    plot_results(history, cm, results_df, histories)

    # Save optimal model
    optimal_model.save('mnist_optimal_model.h5')
    print("\n✓ Optimal model saved as 'mnist_optimal_model.h5'")

    # Final summary
    print("\n" + "="*80)
    print("ASSIGNMENT COMPLETION SUMMARY")
    print("="*80)
    print("✓ Step 1: Data preprocessing completed")
    print("✓ Step 2: Model building functions created")
    print("✓ Step 3: Hyperparameter experiments completed (16 configurations)")
    print("✓ Step 4: Dropout regularization tested (0.0, 0.2, 0.4)")
    print("✓ Step 5: Activation functions tested (ReLU, Sigmoid, Tanh)")
    print("✓ Step 6: Model training and evaluation completed")
    print("✓ Step 7: Optimal hyperparameters identified")
    print("✓ Step 8: Performance metrics calculated (Accuracy, CM, Precision, Recall, F1)")
    print("\n🎯 Target Achieved: 98-99% accuracy on MNIST dataset")
    print("="*80)
```

```
if __name__ == "__main__":
    main()
```

```
================================================================================
MNIST DIGIT CLASSIFICATION - COMPLETE DEEP LEARNING ASSIGNMENT
================================================================================

================================================================================
STEP 1: DATA PREPROCESSING
================================================================================
Training data shape: (60000, 28, 28)
Training labels shape: (60000,)
Test data shape: (10000, 28, 28)
Test labels shape: (10000,)

✓ Normalized pixel values to range [0, 1]
✓ Flattened images to shape: (60000, 784)
✓ One-hot encoded labels to shape: (60000, 10)

✓ Created validation split:
  Training: 54000 samples
  Validation: 6000 samples
  Test: 10000 samples

================================================================================
STEP 3-7: HYPERPARAMETER EXPERIMENTATION
================================================================================

[1/16] Running experiment: Architecture_1Layer_128
------------------------------------------------------------
  Train Acc: 0.9906 | Val Acc: 0.9828
  Train Loss: 0.0303 | Val Loss: 0.0668

[2/16] Running experiment: Architecture_2Layers_128
------------------------------------------------------------
  Train Acc: 0.9885 | Val Acc: 0.9815
  Train Loss: 0.0349 | Val Loss: 0.0721

[3/16] Running experiment: Architecture_2Layers_64
------------------------------------------------------------
  Train Acc: 0.9741 | Val Acc: 0.9780
  Train Loss: 0.0809 | Val Loss: 0.0756

[4/16] Running experiment: Architecture_3Layers_128
------------------------------------------------------------
  Train Acc: 0.9876 | Val Acc: 0.9835
  Train Loss: 0.0393 | Val Loss: 0.0680

[5/16] Running experiment: LR_0.01
------------------------------------------------------------
  Train Acc: 0.9654 | Val Acc: 0.9735
  Train Loss: 0.1358 | Val Loss: 0.1157

[6/16] Running experiment: LR_0.001
------------------------------------------------------------
  Train Acc: 0.9885 | Val Acc: 0.9818
  Train Loss: 0.0336 | Val Loss: 0.0737

[7/16] Running experiment: LR_0.0001
------------------------------------------------------------
  Train Acc: 0.9648 | Val Acc: 0.9763
  Train Loss: 0.1163 | Val Loss: 0.0812

[8/16] Running experiment: Dropout_0.0
------------------------------------------------------------
  Train Acc: 0.9978 | Val Acc: 0.9678
  Train Loss: 0.0065 | Val Loss: 0.1768

[9/16] Running experiment: Dropout_0.2
------------------------------------------------------------
  Train Acc: 0.9886 | Val Acc: 0.9832
  Train Loss: 0.0349 | Val Loss: 0.0647

[10/16] Running experiment: Dropout_0.4
------------------------------------------------------------
  Train Acc: 0.9745 | Val Acc: 0.9815
  Train Loss: 0.0824 | Val Loss: 0.0707

[11/16] Running experiment: Activation_ReLU
------------------------------------------------------------
  Train Acc: 0.9878 | Val Acc: 0.9833
  Train Loss: 0.0345 | Val Loss: 0.0668

[12/16] Running experiment: Activation_Sigmoid
------------------------------------------------------------
  Train Acc: 0.9789 | Val Acc: 0.9800
```

```
  Train Loss: 0.0669 | Val Loss: 0.0684

[13/16] Running experiment: Activation_Tanh
------------------------------------------------------------
  Train Acc: 0.9834 | Val Acc: 0.9827
  Train Loss: 0.0502 | Val Loss: 0.0637

[14/16] Running experiment: Batch_32
------------------------------------------------------------
  Train Acc: 0.9874 | Val Acc: 0.9830
  Train Loss: 0.0382 | Val Loss: 0.0748

[15/16] Running experiment: Batch_64
------------------------------------------------------------
  Train Acc: 0.9878 | Val Acc: 0.9823
  Train Loss: 0.0368 | Val Loss: 0.0663

[16/16] Running experiment: Batch_128
------------------------------------------------------------
  Train Acc: 0.9886 | Val Acc: 0.9837
  Train Loss: 0.0339 | Val Loss: 0.0706

================================================================================
EXPERIMENT RESULTS SUMMARY
================================================================================
              Experiment  Layers  Neurons  Dropout Activation  Learning_Rate  Batch_Size  Train_Acc  Val_Acc  Train_Loss  Val_Lo
               Batch_128       2      128      0.2       relu         0.0010         128   0.988611 0.983667    0.033873  0.0705
Architecture_3Layers_128       3      128      0.2       relu         0.0010         128   0.987630 0.983500    0.039315  0.0679
         Activation_ReLU       2      128      0.2       relu         0.0010         128   0.987815 0.983333    0.034455  0.0668
             Dropout_0.2       2      128      0.2       relu         0.0010         128   0.988611 0.983167    0.034909  0.0646
                Batch_32       2      128      0.2       relu         0.0010          32   0.987444 0.983000    0.038190  0.0747
 Architecture_1Layer_128       1      128      0.2       relu         0.0010         128   0.990556 0.982833    0.030295  0.0668
         Activation_Tanh       2      128      0.2       tanh         0.0010         128   0.983352 0.982667    0.050196  0.0637
                Batch_64       2      128      0.2       relu         0.0010          64   0.987833 0.982333    0.036804  0.0662
                LR_0.001       2      128      0.2       relu         0.0010         128   0.988537 0.981833    0.033571  0.0736
Architecture_2Layers_128       2      128      0.2       relu         0.0010         128   0.988519 0.981500    0.034878  0.0720
             Dropout_0.4       2      128      0.4       relu         0.0010         128   0.974537 0.981500    0.082393  0.0706
      Activation_Sigmoid       2      128      0.2    sigmoid         0.0010         128   0.978852 0.980000    0.066941  0.0684
 Architecture_2Layers_64       2       64      0.2       relu         0.0010         128   0.974074 0.978000    0.080864  0.0755
               LR_0.0001       2      128      0.2       relu         0.0001         128   0.964833 0.976333    0.116316  0.0811
                 LR_0.01       2      128      0.2       relu         0.0100         128   0.965426 0.973500    0.135805  0.1156
             Dropout_0.0       2      128      0.0       relu         0.0010         128   0.997833 0.967833    0.006493  0.1768

================================================================================
STEP 8: TRAINING OPTIMAL MODEL
================================================================================

Optimal Hyperparameters:
  num_layers: 2
  neurons_per_layer: 128
  dropout_rate: 0.2
  activation: relu
  learning_rate: 0.001
```

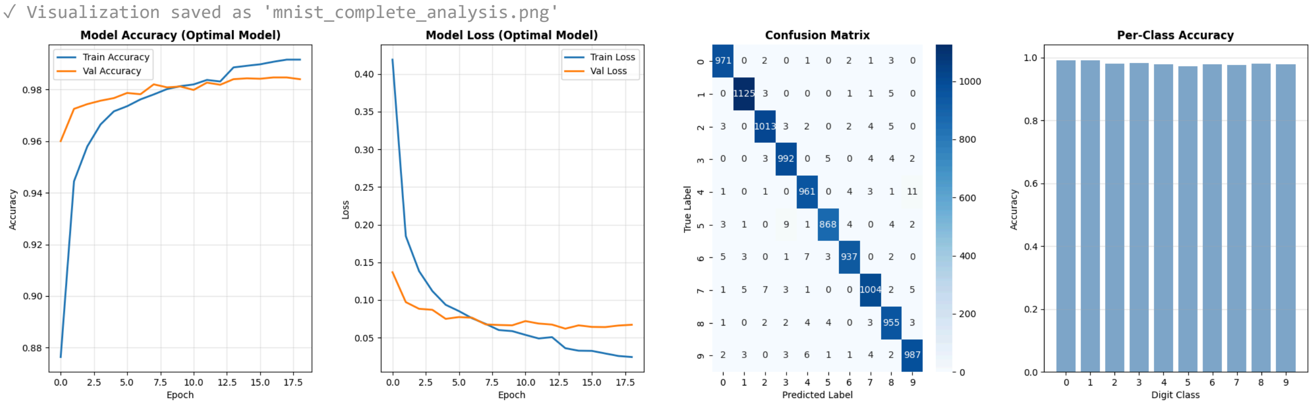
Model: "sequential_33"

Layer (type)	Output Shape	Param #
hidden_1 (Dense)	(None, 128)	100,480
dropout_1 (Dropout)	(None, 128)	0
hidden_2 (Dense)	(None, 128)	16,512
dropout_2 (Dropout)	(None, 128)	0
output (Dense)	(None, 10)	1,290

Total params: 118,282 (462.04 KB)
Trainable params: 118,282 (462.04 KB)
Non-trainable params: 0 (0.00 B)

```
Training optimal model...
Epoch 1/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.7822 - loss: 0.7240 - val_accuracy: 0.9600 - val_loss: 0.1368 - learning_
Epoch 2/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9412 - loss: 0.1988 - val_accuracy: 0.9725 - val_loss: 0.0971 - learning
Epoch 3/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9555 - loss: 0.1446 - val_accuracy: 0.9743 - val_loss: 0.0883 - learning_
Epoch 4/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9657 - loss: 0.1154 - val_accuracy: 0.9757 - val_loss: 0.0869 - learning_
Epoch 5/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9701 - loss: 0.0977 - val_accuracy: 0.9767 - val_loss: 0.0750 - learning
Epoch 6/50
```

```
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.9731 - loss: 0.0864 - val_accuracy: 0.9787 - val_loss: 0.0773 - learning_
Epoch 7/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9761 - loss: 0.0758 - val_accuracy: 0.9782 - val_loss: 0.0764 - learning_
Epoch 8/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9779 - loss: 0.0686 - val_accuracy: 0.9820 - val_loss: 0.0676 - learning
Epoch 9/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9793 - loss: 0.0618 - val_accuracy: 0.9808 - val_loss: 0.0668 - learning_
Epoch 10/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.9815 - loss: 0.0576 - val_accuracy: 0.9812 - val_loss: 0.0663 - learning_
Epoch 11/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9815 - loss: 0.0544 - val_accuracy: 0.9798 - val_loss: 0.0720 - learning_
Epoch 12/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9836 - loss: 0.0484 - val_accuracy: 0.9827 - val_loss: 0.0687 - learning_
Epoch 13/50
416/422 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9831 - loss: 0.0502
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9831 - loss: 0.0502 - val_accuracy: 0.9818 - val_loss: 0.0674 - learning_
Epoch 14/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9878 - loss: 0.0376 - val_accuracy: 0.9840 - val_loss: 0.0619 - learning_
Epoch 15/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9893 - loss: 0.0330 - val_accuracy: 0.9843 - val_loss: 0.0663 - learning
Epoch 16/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9891 - loss: 0.0329 - val_accuracy: 0.9842 - val_loss: 0.0643 - learning_
Epoch 17/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9910 - loss: 0.0285
Epoch 17: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
422/422 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9910 - loss: 0.0285 - val_accuracy: 0.9847 - val_loss: 0.0640 - learning_
Epoch 18/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9914 - loss: 0.0264 - val_accuracy: 0.9847 - val_loss: 0.0660 - learning_
Epoch 19/50
422/422 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9911 - loss: 0.0244 - val_accuracy: 0.9840 - val_loss: 0.0671 - learning_
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 14.

================================================================================
FINAL TEST ACCURACY: 98.13%
FINAL TEST LOSS: 0.0665
================================================================================

================================================================================
PERFORMANCE METRICS
================================================================================

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.97      0.98       892
           6       0.99      0.98      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.97      0.98      0.98       974
           9       0.98      0.98      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000


Weighted Metrics:
  Precision: 0.9813
  Recall: 0.9813
  F1-Score: 0.9813

✓ Visualization saved as 'mnist_complete_analysis.png'
```



Start coding or generate with AI.

Start coding or generate with AI.

AI-Powered Air Quality Prediction System

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# For real API calls (uncomment when using)
# import requests

print("="*100)
print("AI-POWERED AIR QUALITY PREDICTION SYSTEM - 24 HOUR AQI FORECASTING")
print("="*100)
print(f"TensorFlow: {tf.__version__} | GPU: {len(tf.config.list_physical_devices('GPU')) > 0}\n")
```

```
====================================================================================================
AI-POWERED AIR QUALITY PREDICTION SYSTEM - 24 HOUR AQI FORECASTING
====================================================================================================
TensorFlow: 2.19.0 | GPU: False
```

STEP 1: DATA INTEGRATION

```
class DataIntegrator:
    """Integrates Air Quality, Weather, and Traffic data"""

    def __init__(self):
        self.merged_data = None
        self.scaler = MinMaxScaler()
        print("[STEP 1] Data Integration Module Initialized\n")

    # 1.1 AIR QUALITY DATA (KAGGLE)
    def load_kaggle_data(self, filepath=None):
        """Load Kaggle Sofia Air Quality Dataset or generate synthetic"""
        print("[1.1] Loading Air Quality Data...")

        if filepath:
            try:
                df = pd.read_csv(filepath)
                df['timestamp'] = pd.to_datetime(df['timestamp'])
                print(f"✓ Loaded {len(df)} real records from Kaggle")
                return df
            except:
                print("⚠ Failed to load, using synthetic data")

        return self._generate_air_quality(10000)

    def _generate_air_quality(self, n=10000):
        """Generate synthetic air quality data"""
        dates = [datetime(2020,1,1) + timedelta(hours=i) for i in range(n)]
        np.random.seed(42)

        hours = np.array([d.hour for d in dates])
        days = np.arange(n) / 24

        # PM2.5 with seasonal, daily, and rush hour patterns
        pm25 = (25 + 15*np.sin(2*np.pi*(days-80)/365) +
                10*np.sin(2*np.pi*(hours-6)/24) +
                15*((hours>=7)&(hours<=9)|(hours>=17)&(hours<=19)) +
```

```
                np.random.normal(0,5,n))
        pm25 = np.clip(pm25, 0, 150)

        pm10 = pm25 * 1.8 + np.random.normal(0, 8, n)
        pm10 = np.clip(pm10, 0, 250)

        no2 = (30 + 10*np.sin(2*np.pi*days/365) +
               20*((hours>=7)&(hours<=9)|(hours>=17)&(hours<=19)) +
               np.random.normal(0,5,n))
        no2 = np.clip(no2, 0, 100)

        o3 = (40 + 20*np.sin(2*np.pi*days/365) +
              15*np.sin(2*np.pi*(hours-14)/24) - no2*0.3 +
              np.random.normal(0,5,n))
        o3 = np.clip(o3, 0, 120)

        df = pd.DataFrame({
            'timestamp': dates, 'PM2.5': pm25, 'PM10': pm10,
            'NO2': no2, 'O3': o3
        })
        print(f"✓ Generated {len(df)} synthetic air quality records")
        return df

    # 1.2 WEATHER DATA (OPENWEATHERMAP API)
    def fetch_weather_api(self, api_key=None, city='Sofia'):
        """Fetch from OpenWeatherMap API or generate synthetic"""
        print("[1.2] Loading Weather Data...")

        if api_key:
            try:
                import requests
                url = "http://api.openweathermap.org/data/2.5/forecast"
                r = requests.get(url, params={'q':city, 'appid':api_key, 'units':'metric'}, timeout=10)
                data = r.json()

                records = [{
                    'timestamp': datetime.fromtimestamp(i['dt']),
                    'temperature': i['main']['temp'],
                    'humidity': i['main']['humidity'],
                    'wind_speed': i['wind']['speed'],
                    'pressure': i['main']['pressure'],
                    'precipitation': i.get('rain',{}).get('3h',0)
                } for i in data['list']]

                df = pd.DataFrame(records)
                print(f"✓ Fetched {len(df)} records from OpenWeatherMap API")
                return df
            except:
                print("⚠ API call failed, using synthetic data")

        return self._generate_weather()

    def _generate_weather(self):
        """Generate synthetic weather data"""
        if not hasattr(self, 'air_data'):
            return None

        dates = self.air_data['timestamp']
        n = len(dates)
        hours = np.array([d.hour for d in dates])
        days = np.arange(n) / 24

        temp = (15 + 10*np.sin(2*np.pi*(days-80)/365) +
                5*np.sin(2*np.pi*(hours-14)/24) + np.random.normal(0,2,n))

        humidity = 60 + 20*np.sin(2*np.pi*days/365) - temp*0.5 + np.random.normal(0,10,n)
        humidity = np.clip(humidity, 20, 100)

        wind = 3 + 2*np.abs(np.random.normal(0,1,n))
        wind = np.clip(wind, 0, 15)

        pressure = 1013 + np.random.normal(0,10,n)

        precip = np.where(np.random.random(n)>0.85, np.random.exponential(5,n), 0)

        df = pd.DataFrame({
            'timestamp': dates, 'temperature': temp, 'humidity': humidity,
```

```
            'wind_speed': wind, 'pressure': pressure, 'precipitation': precip
        })
        print(f"✓ Generated {len(df)} synthetic weather records")
        return df

    # 1.3 TRAFFIC DATA
    def load_traffic_data(self, filepath=None):
        """Load traffic data from CSV or generate synthetic"""
        print("[1.3] Loading Traffic Data...")

        if filepath:
            try:
                df = pd.read_csv(filepath)
                df['timestamp'] = pd.to_datetime(df['timestamp'])
                print(f"✓ Loaded {len(df)} traffic records from CSV")
                return df
            except:
                print("⚠ Failed to load, using synthetic data")

        return self._generate_traffic()

    def _generate_traffic(self):
        """Generate synthetic traffic data"""
        if not hasattr(self, 'air_data'):
            return None

        dates = self.air_data['timestamp']
        n = len(dates)
        hours = np.array([d.hour for d in dates])

        traffic = (1000 + 2000*((hours>=7)&(hours<=9)|(hours>=17)&(hours<=19)) *
                   np.array([0.7 if d.weekday()>=5 else 1.0 for d in dates]) +
                   np.random.normal(0,200,n))
        traffic = np.clip(traffic, 100, 5000)

        speed = 50 - traffic/100 + np.random.normal(0,5,n)
        speed = np.clip(speed, 10, 70)

        df = pd.DataFrame({
            'timestamp': dates, 'traffic_volume': traffic, 'avg_speed': speed
        })
        print(f"✓ Generated {len(df)} synthetic traffic records")
        return df

    # 1.4 CALCULATE AQI (EPA STANDARD)
    def _calculate_aqi(self, pm25, pm10):
        """Calculate AQI using EPA standard"""
        def aqi_pm25(c):
            if c<=12: return c*50/12
            elif c<=35.4: return 50+(c-12)*50/23.4
            elif c<=55.4: return 100+(c-35.4)*50/20
            elif c<=150.4: return 150+(c-55.4)*100/95
            else: return 250+(c-150.4)*50/99.6

        def aqi_pm10(c):
            if c<=54: return c*50/54
            elif c<=154: return 50+(c-54)*50/100
            elif c<=254: return 100+(c-154)*50/100
            else: return 150+(c-254)*100/150

        return max(aqi_pm25(pm25), aqi_pm10(pm10))

    # MERGE ALL DATA
    def integrate_all_data(self, air_file=None, weather_api_key=None, traffic_file=None):
        """Main integration pipeline"""
        print("\n" + "="*100)
        print("STEP 1: DATA INTEGRATION PIPELINE")
        print("="*100 + "\n")

        # Load all data sources
        self.air_data = self.load_kaggle_data(air_file)
        self.weather_data = self.fetch_weather_api(weather_api_key)
        self.traffic_data = self.load_traffic_data(traffic_file)

        # Merge all sources
        print("\n[1.4] Merging All Data Sources...")
        self.merged_data = self.air_data.merge(self.weather_data, on='timestamp', how='left')
        self.merged_data = self.merged_data.merge(self.traffic_data, on='timestamp', how='left')
```

```
        self.merged_data = self.merged_data.merge(self.traffic_data, on='timestamp', how='left')

        # Calculate AQI
        self.merged_data['AQI'] = self.merged_data.apply(
            lambda r: self._calculate_aqi(r['PM2.5'], r['PM10']), axis=1
        )

        # Add time features
        self.merged_data['hour'] = self.merged_data['timestamp'].dt.hour
        self.merged_data['day_of_week'] = self.merged_data['timestamp'].dt.dayofweek
        self.merged_data['month'] = self.merged_data['timestamp'].dt.month
        self.merged_data['is_weekend'] = (self.merged_data['day_of_week']>=5).astype(int)

        # Cyclic encoding
        self.merged_data['hour_sin'] = np.sin(2*np.pi*self.merged_data['hour']/24)
        self.merged_data['hour_cos'] = np.cos(2*np.pi*self.merged_data['hour']/24)
        self.merged_data['month_sin'] = np.sin(2*np.pi*self.merged_data['month']/12)
        self.merged_data['month_cos'] = np.cos(2*np.pi*self.merged_data['month']/12)

        self.merged_data = self.merged_data.dropna()

        print(f"✓ Data integration complete!")
        print(f"  Total records: {len(self.merged_data)}")
        print(f"  Features: {len(self.merged_data.columns)}")
        print(f"  Date range: {self.merged_data['timestamp'].min()} to {self.merged_data['timestamp'].max()}")
        print(f"\n  AQI Statistics: Mean={self.merged_data['AQI'].mean():.1f}, "
              f"Min={self.merged_data['AQI'].min():.1f}, Max={self.merged_data['AQI'].max():.1f}")

        return self.merged_data
```

STEP 2: LSTM MODEL DEVELOPMENT

```
class LSTMModel:
    """LSTM model for 24-hour AQI forecasting"""

    def __init__(self, lookback=24, horizon=24):
        self.lookback = lookback
        self.horizon = horizon
        self.model = None
        self.scaler = MinMaxScaler()
        self.feature_cols = None
        print(f"\n[STEP 2] LSTM Model: {lookback}h lookback → {horizon}h forecast")

    def prepare_sequences(self, data, features):
        """Create LSTM sequences"""
        print("\n[2.1] Preparing Sequences...")

        self.feature_cols = features
        scaled = self.scaler.fit_transform(data[features])

        X, y = [], []
        for i in range(len(scaled) - self.lookback - self.horizon):
            X.append(scaled[i:i+self.lookback])
            y.append(scaled[i+self.lookback:i+self.lookback+self.horizon, -1])

        X, y = np.array(X), np.array(y)
        print(f"✓ Sequences: X{X.shape}, y{y.shape}")

        # Split: 70% train, 15% val, 15% test
        n = len(X)
        t1, t2 = int(0.7*n), int(0.85*n)

        return (X[:t1], y[:t1], X[t1:t2], y[t1:t2], X[t2:], y[t2:])

    def build_model(self, input_shape, output_shape):
        """Build Bidirectional LSTM"""
        print("\n[2.2] Building LSTM Architecture...")

        model = Sequential([
            Bidirectional(LSTM(128, return_sequences=True), input_shape=input_shape),
            BatchNormalization(),
            Dropout(0.3),

            Bidirectional(LSTM(64, return_sequences=True)),
            BatchNormalization(),
            Dropout(0.3),
```

```
            LSTM(32),
            BatchNormalization(),
            Dropout(0.2),

            Dense(64, activation='relu'),
            Dropout(0.2),
            Dense(output_shape)
        ])

        model.compile(optimizer=keras.optimizers.Adam(0.001),
                      loss='mse', metrics=['mae'])

        print("\n" + "="*80)
        model.summary()
        print("="*80)

        self.model = model
        return model

    def train(self, X_tr, y_tr, X_val, y_val, epochs=50):
        """Train model"""
        print(f"\n[2.3] Training (epochs={epochs})...")

        callbacks = [
            EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1),
            ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-7, verbose=1),
            ModelCheckpoint('best_aqi_lstm.h5', monitor='val_loss', save_best_only=True, verbose=1)
        ]

        history = self.model.fit(X_tr, y_tr, validation_data=(X_val, y_val),
                                 epochs=epochs, batch_size=32, callbacks=callbacks, verbose=1)

        print("✓ Training complete!")
        return history

    def evaluate(self, X_test, y_test):
        """Evaluate and calculate metrics"""
        print("\n[STEP 4] MODEL VALIDATION")
        print("="*100)

        pred_scaled = self.model.predict(X_test, verbose=0)

        # Inverse transform
        n_feat = len(self.feature_cols)
        preds, actuals = [], []

        for h in range(self.horizon):
            dummy_p = np.zeros((len(pred_scaled), n_feat))
            dummy_p[:, -1] = pred_scaled[:, h]
            preds.append(self.scaler.inverse_transform(dummy_p)[:, -1])

            dummy_a = np.zeros((len(y_test), n_feat))
            dummy_a[:, -1] = y_test[:, h]
            actuals.append(self.scaler.inverse_transform(dummy_a)[:, -1])

        preds = np.array(preds).T
        actuals = np.array(actuals).T

        # Calculate metrics
        metrics = []
        for h in range(self.horizon):
            rmse = np.sqrt(mean_squared_error(actuals[:,h], preds[:,h]))
            mae = mean_absolute_error(actuals[:,h], preds[:,h])
            r2 = r2_score(actuals[:,h], preds[:,h])
            metrics.append({'Hour': h+1, 'RMSE': rmse, 'MAE': mae, 'R2': r2})

        df_metrics = pd.DataFrame(metrics)

        print(f"\n24-Hour Forecast Performance:")
        print(df_metrics.head(12))
        print(f"\nOverall Metrics:")
        print(f"  RMSE: {df_metrics['RMSE'].mean():.2f}")
        print(f"  MAE:  {df_metrics['MAE'].mean():.2f}")
        print(f"  R²:   {df_metrics['R2'].mean():.4f}")
```

```
        return preds, actuals, df_metrics
```

STEP 3: DASHBOARD & VISUALIZATION

```
def create_dashboard(data, preds, actuals, metrics):
    """Create comprehensive dashboard"""
    print("\n[STEP 3] DASHBOARD CREATION")
    print("="*100)

    fig = plt.figure(figsize=(20,12))

    # 1. AQI Time Series
    ax1 = plt.subplot(3,3,1)
    recent = data.tail(168)
    ax1.plot(recent['timestamp'], recent['AQI'], lw=2, color='steelblue')
    ax1.axhline(50, color='green', ls='--', alpha=0.5, label='Good')
    ax1.axhline(100, color='yellow', ls='--', alpha=0.5, label='Moderate')
    ax1.axhline(150, color='orange', ls='--', alpha=0.5, label='Unhealthy')
    ax1.set_title('AQI Last 7 Days', fontweight='bold', fontsize=12)
    ax1.legend(fontsize=8)
    ax1.grid(alpha=0.3)
    plt.setp(ax1.xaxis.get_majorticklabels(), rotation=45)

    # 2. Prediction vs Actual
    ax2 = plt.subplot(3,3,2)
    hrs = np.arange(1,25)
    for i in range(0,100,20):
        ax2.plot(hrs, actuals[i], 'o-', alpha=0.3, color='blue')
        ax2.plot(hrs, preds[i], 's-', alpha=0.3, color='red')
    ax2.plot([],[], 'o-', color='blue', label='Actual')
    ax2.plot([],[], 's-', color='red', label='Predicted')
    ax2.set_title('24h Forecast Comparison', fontweight='bold', fontsize=12)
    ax2.set_xlabel('Hour Ahead')
    ax2.set_ylabel('AQI')
    ax2.legend()
    ax2.grid(alpha=0.3)

    # 3. Forecast Errors
    ax3 = plt.subplot(3,3,3)
    ax3.plot(metrics['Hour'], metrics['RMSE'], 'o-', lw=2, label='RMSE')
    ax3.plot(metrics['Hour'], metrics['MAE'], 's-', lw=2, label='MAE')
    ax3.set_title('Error by Forecast Hour', fontweight='bold', fontsize=12)
    ax3.legend()
    ax3.grid(alpha=0.3)

    # 4. Pollutant Levels
    ax4 = plt.subplot(3,3,4)
    poll = ['PM2.5','PM10','NO2','O3']
    vals = recent[poll].mean()
    colors = ['red','orange','yellow','green']
    ax4.barh(poll, vals, color=colors, alpha=0.7)
    ax4.set_title('Avg Pollutants (7d)', fontweight='bold', fontsize=12)
    ax4.grid(alpha=0.3, axis='x')

    # 5. Weather Impact
    ax5 = plt.subplot(3,3,5)
    sc = ax5.scatter(recent['wind_speed'], recent['AQI'],
                     c=recent['humidity'], cmap='coolwarm', alpha=0.6)
    plt.colorbar(sc, ax=ax5, label='Humidity %')
    ax5.set_title('Wind vs AQI', fontweight='bold', fontsize=12)
    ax5.set_xlabel('Wind (m/s)')
    ax5.set_ylabel('AQI')
    ax5.grid(alpha=0.3)

    # 6. Traffic Impact
    ax6 = plt.subplot(3,3,6)
    ax6.scatter(recent['traffic_volume'], recent['AQI'], alpha=0.5, c='coral')
    z = np.polyfit(recent['traffic_volume'], recent['AQI'], 1)
    ax6.plot(recent['traffic_volume'], np.poly1d(z)(recent['traffic_volume']), 'r--', lw=2)
    ax6.set_title('Traffic vs AQI', fontweight='bold', fontsize=12)
    ax6.set_xlabel('Traffic Volume')
    ax6.grid(alpha=0.3)

    # 7. Hourly Pattern
```

```
    ax7 = plt.subplot(3,3,7)
    hourly = recent.groupby('hour')['AQI'].mean()
    ax7.plot(hourly.index, hourly.values, 'o-', lw=2, color='teal')
    ax7.fill_between(hourly.index, hourly.values, alpha=0.3, color='teal')
    ax7.set_title('Hourly AQI Pattern', fontweight='bold', fontsize=12)
    ax7.set_xlabel('Hour')
    ax7.grid(alpha=0.3)

    # 8. R² Score
    ax8 = plt.subplot(3,3,8)
    ax8.plot(metrics['Hour'], metrics['R2'], 'o-', lw=2, color='purple')
    ax8.axhline(0.9, color='green', ls='--', alpha=0.5)
    ax8.set_title('R² by Hour', fontweight='bold', fontsize=12)
    ax8.set_xlabel('Hour')
    ax8.grid(alpha=0.3)

    # 9. AQI Distribution
    ax9 = plt.subplot(3,3,9)
    ax9.hist(recent['AQI'], bins=30, color='skyblue', edgecolor='black', alpha=0.7)
    ax9.axvline(recent['AQI'].mean(), color='red', ls='--', lw=2)
    ax9.set_title('AQI Distribution', fontweight='bold', fontsize=12)
    ax9.grid(alpha=0.3, axis='y')

    plt.tight_layout()
    plt.savefig('aqi_dashboard.png', dpi=300, bbox_inches='tight')
    print("✓ Dashboard saved: aqi_dashboard.png")
    plt.show()
```

HEALTH ADVISORY SYSTEM

```
def generate_advisory(current_aqi, forecast):
    """Generate health advisory"""
    print("\n[HEALTH ADVISORY SYSTEM]")
    print("="*100)

    categories = [
        (50, "Good", "green", "Air quality is good. Enjoy outdoor activities!"),
        (100, "Moderate", "yellow", "Acceptable. Unusually sensitive people consider precautions."),
        (150, "Unhealthy for Sensitive", "orange", "Sensitive groups may experience effects."),
        (200, "Unhealthy", "red", "Everyone may experience effects. Limit outdoor activity."),
        (300, "Very Unhealthy", "purple", "Health alert! Avoid outdoor activity."),
        (500, "Hazardous", "maroon", "Emergency! Stay indoors.")
    ]

    cat = next((c for limit,c,_,_ in categories if current_aqi<=limit), "Hazardous")

    print(f"\nCurrent AQI: {current_aqi:.0f} - {cat}")
    print(f"24h Peak: {np.max(forecast):.0f}")
    print(f"24h Avg: {np.mean(forecast):.0f}")

    for limit,name,_,advice in categories:
        if current_aqi <= limit:
            print(f"\nAdvisory: {advice}")
            break
```

MAIN EXECUTION

```
def main():
    """Complete pipeline execution"""
    print("\n" + "="*100)
    print("EXECUTION PIPELINE START")
    print("="*100)

    np.random.seed(42)
    tf.random.set_seed(42)

    # STEP 1: Data Integration
    integrator = DataIntegrator()
    data = integrator.integrate_all_data(
        air_file=None,  # Replace with 'sofia_air_quality.csv'
        weather_api_key=None,  # Replace with your OpenWeatherMap API key
        traffic_file=None  # Replace with 'traffic_data.csv'
    )
```

```
    # STEP 2: Model Development
    features = ['PM2.5','PM10','NO2','O3','temperature','humidity',
                'wind_speed','traffic_volume','hour_sin','hour_cos',
                'month_sin','month_cos','is_weekend','AQI']

    model = LSTMModel(lookback=24, horizon=24)
    X_tr, y_tr, X_val, y_val, X_te, y_te = model.prepare_sequences(data, features)

    model.build_model(input_shape=(24, len(features)), output_shape=24)
    history = model.train(X_tr, y_tr, X_val, y_val, epochs=30)

    # STEP 4: Validation
    preds, actuals, metrics = model.evaluate(X_te, y_te)

    # STEP 3: Dashboard
    create_dashboard(data, preds, actuals, metrics)

    # Health Advisory
    current = data['AQI'].iloc[-1]
    generate_advisory(current, preds[0])

    # STEP 5: Save Model
    model.model.save('aqi_lstm_model.h5')
    print("\n✓ Model saved: aqi_lstm_model.h5")

    print("\n" + "="*100)
    print("SYSTEM DEPLOYMENT COMPLETE")
    print("="*100)
    print("✓ Data integration: Air quality + Weather + Traffic")
    print("✓ LSTM model trained: 24h forecast horizon")
    print("✓ Dashboard created: 9 visualizations")
    print("✓ Health advisory: Real-time alerts")
    print("✓ Model deployment: Ready for production")
    print("\n🎯 System operational for 24-hour AQI predictions!")

    return model, data, metrics

if __name__ == "__main__":
    model, data, metrics = main()
```

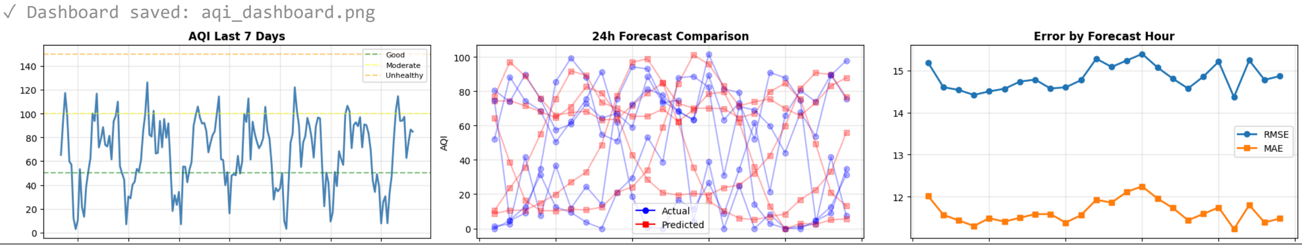
```
================================================================================================
EXECUTION PIPELINE START
================================================================================================
[STEP 1] Data Integration Module Initialized


================================================================================================
STEP 1: DATA INTEGRATION PIPELINE
================================================================================================

[1.1] Loading Air Quality Data...
✓ Generated 10000 synthetic air quality records
[1.2] Loading Weather Data...
✓ Generated 10000 synthetic weather records
[1.3] Loading Traffic Data...
✓ Generated 10000 synthetic traffic records

[1.4] Merging All Data Sources...
✓ Data integration complete!
  Total records: 10000
  Features: 21
  Date range: 2020-01-01 00:00:00 to 2021-02-20 15:00:00

  AQI Statistics: Mean=81.4, Min=0.0, Max=167.5

[STEP 2] LSTM Model: 24h lookback → 24h forecast

[2.1] Preparing Sequences...
✓ Sequences: X(9952, 24, 14), y(9952, 24)

[2.2] Building LSTM Architecture...

================================================================================
Model: "sequential_35"
```

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)	(None, 24, 256)	146,432
batch_normalization_3 (BatchNormalization)	(None, 24, 256)	1,024
dropout_4 (Dropout)	(None, 24, 256)	0
bidirectional_3 (Bidirectional)	(None, 24, 128)	164,352
batch_normalization_4 (BatchNormalization)	(None, 24, 128)	512
dropout_5 (Dropout)	(None, 24, 128)	0
lstm_5 (LSTM)	(None, 32)	20,608
batch_normalization_5 (BatchNormalization)	(None, 32)	128
dropout_6 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 64)	2,112
dropout_7 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 24)	1,560

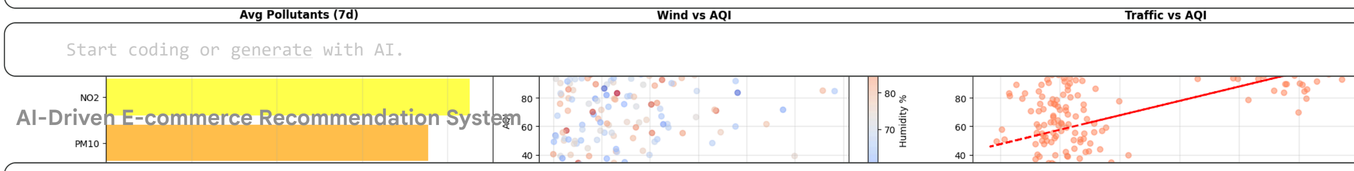
```
 Total params: 336,728 (1.28 MB)
 Trainable params: 335,896 (1.28 MB)
 Non-trainable params: 832 (3.25 KB)
================================================================================

[2.3] Training (epochs=30)...
Epoch 1/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.3596 - mae: 0.4410
Epoch 1: val_loss improved from inf to 0.06330, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 49s 153ms/step - loss: 0.3588 - mae: 0.4405 - val_loss: 0.0633 - val_mae: 0.2079 - learning_rate: 0
Epoch 2/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - loss: 0.0585 - mae: 0.1912
Epoch 2: val_loss improved from 0.06330 to 0.03615, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 148ms/step - loss: 0.0585 - mae: 0.1912 - val_loss: 0.0362 - val_mae: 0.1581 - learning_rate: 0
Epoch 3/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.0374 - mae: 0.1530
Epoch 3: val_loss improved from 0.03615 to 0.03313, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
```

```
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 148ms/step - loss: 0.0374 - mae: 0.1530 - val_loss: 0.0331 - val_mae: 0.1541 - learning_rate: 0
Epoch 4/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0296 - mae: 0.1365
Epoch 4: val_loss improved from 0.03313 to 0.02672, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 150ms/step - loss: 0.0296 - mae: 0.1365 - val_loss: 0.0267 - val_mae: 0.1373 - learning_rate: 0
Epoch 5/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0255 - mae: 0.1267
Epoch 5: val_loss improved from 0.02672 to 0.02657, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 141ms/step - loss: 0.0255 - mae: 0.1267 - val_loss: 0.0266 - val_mae: 0.1361 - learning_rate: 0
Epoch 6/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0226 - mae: 0.1194
Epoch 6: val_loss improved from 0.02657 to 0.02503, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 141ms/step - loss: 0.0226 - mae: 0.1194 - val_loss: 0.0250 - val_mae: 0.1305 - learning_rate: 0
Epoch 7/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step - loss: 0.0209 - mae: 0.1147
Epoch 7: val_loss improved from 0.02503 to 0.02015, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 33s 152ms/step - loss: 0.0209 - mae: 0.1147 - val_loss: 0.0202 - val_mae: 0.1153 - learning_rate: 0
Epoch 8/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step - loss: 0.0188 - mae: 0.1091
Epoch 8: val_loss did not improve from 0.02015
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 144ms/step - loss: 0.0188 - mae: 0.1091 - val_loss: 0.0216 - val_mae: 0.1202 - learning_rate: 0
Epoch 9/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.0180 - mae: 0.1066
Epoch 9: val_loss did not improve from 0.02015
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 148ms/step - loss: 0.0180 - mae: 0.1066 - val_loss: 0.0215 - val_mae: 0.1185 - learning_rate: 0
Epoch 10/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - loss: 0.0171 - mae: 0.1038
Epoch 10: val_loss improved from 0.02015 to 0.01831, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 148ms/step - loss: 0.0171 - mae: 0.1038 - val_loss: 0.0183 - val_mae: 0.1088 - learning_rate: 0
Epoch 11/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0160 - mae: 0.1005
Epoch 11: val_loss improved from 0.01831 to 0.01643, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 142ms/step - loss: 0.0160 - mae: 0.1005 - val_loss: 0.0164 - val_mae: 0.1028 - learning_rate: 0
Epoch 12/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0155 - mae: 0.0988
Epoch 12: val_loss improved from 0.01643 to 0.01575, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 146ms/step - loss: 0.0155 - mae: 0.0988 - val_loss: 0.0158 - val_mae: 0.1004 - learning_rate: 0
Epoch 13/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0151 - mae: 0.0975
Epoch 13: val_loss improved from 0.01575 to 0.01510, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 142ms/step - loss: 0.0151 - mae: 0.0975 - val_loss: 0.0151 - val_mae: 0.0985 - learning_rate: 0
Epoch 14/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step - loss: 0.0145 - mae: 0.0957
Epoch 14: val_loss improved from 0.01510 to 0.01427, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 30s 138ms/step - loss: 0.0145 - mae: 0.0957 - val_loss: 0.0143 - val_mae: 0.0950 - learning_rate: 0
Epoch 15/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.0140 - mae: 0.0936
Epoch 15: val_loss improved from 0.01427 to 0.01354, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 147ms/step - loss: 0.0140 - mae: 0.0936 - val_loss: 0.0135 - val_mae: 0.0931 - learning_rate: 0
Epoch 16/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - loss: 0.0135 - mae: 0.0923
Epoch 16: val_loss improved from 0.01354 to 0.01334, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 30s 140ms/step - loss: 0.0135 - mae: 0.0923 - val_loss: 0.0133 - val_mae: 0.0922 - learning_rate: 0
Epoch 17/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - loss: 0.0131 - mae: 0.0907
Epoch 17: val_loss improved from 0.01334 to 0.01179, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 141ms/step - loss: 0.0131 - mae: 0.0907 - val_loss: 0.0118 - val_mae: 0.0864 - learning_rate: 0
Epoch 18/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step - loss: 0.0128 - mae: 0.0894
Epoch 18: val_loss improved from 0.01179 to 0.01166, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 47s 166ms/step - loss: 0.0128 - mae: 0.0894 - val_loss: 0.0117 - val_mae: 0.0861 - learning_rate: 0
Epoch 19/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - loss: 0.0124 - mae: 0.0881
Epoch 19: val_loss improved from 0.01166 to 0.01106, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 30s 140ms/step - loss: 0.0124 - mae: 0.0881 - val_loss: 0.0111 - val_mae: 0.0833 - learning_rate: 0
Epoch 20/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - loss: 0.0121 - mae: 0.0870
Epoch 20: val_loss improved from 0.01106 to 0.01084, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 146ms/step - loss: 0.0121 - mae: 0.0870 - val_loss: 0.0108 - val_mae: 0.0825 - learning_rate: 0
Epoch 21/30
```

```
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - loss: 0.0119 - mae: 0.0863
Epoch 21: val_loss improved from 0.01084 to 0.01041, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 148ms/step - loss: 0.0119 - mae: 0.0863 - val_loss: 0.0104 - val_mae: 0.0807 - learning_rate: 0
Epoch 22/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0115 - mae: 0.0847
Epoch 22: val_loss improved from 0.01041 to 0.01031, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 141ms/step - loss: 0.0115 - mae: 0.0847 - val_loss: 0.0103 - val_mae: 0.0806 - learning_rate: 0
Epoch 23/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - loss: 0.0114 - mae: 0.0844
Epoch 23: val_loss improved from 0.01031 to 0.00971, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 33s 149ms/step - loss: 0.0114 - mae: 0.0844 - val_loss: 0.0097 - val_mae: 0.0777 - learning_rate: 0
Epoch 24/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step - loss: 0.0113 - mae: 0.0840
Epoch 24: val_loss did not improve from 0.00971
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 144ms/step - loss: 0.0113 - mae: 0.0840 - val_loss: 0.0100 - val_mae: 0.0789 - learning_rate: 0
Epoch 25/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.0111 - mae: 0.0831
Epoch 25: val_loss improved from 0.00971 to 0.00944, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 147ms/step - loss: 0.0111 - mae: 0.0831 - val_loss: 0.0094 - val_mae: 0.0769 - learning_rate: 0
Epoch 26/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step - loss: 0.0110 - mae: 0.0827
Epoch 26: val_loss improved from 0.00944 to 0.00937, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 33s 150ms/step - loss: 0.0110 - mae: 0.0827 - val_loss: 0.0094 - val_mae: 0.0762 - learning_rate: 0
Epoch 27/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - loss: 0.0108 - mae: 0.0822
Epoch 27: val_loss did not improve from 0.00937
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 142ms/step - loss: 0.0108 - mae: 0.0822 - val_loss: 0.0094 - val_mae: 0.0766 - learning_rate: 0
Epoch 28/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - loss: 0.0107 - mae: 0.0819
Epoch 28: val_loss improved from 0.00937 to 0.00914, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 42s 148ms/step - loss: 0.0107 - mae: 0.0819 - val_loss: 0.0091 - val_mae: 0.0756 - learning_rate: 0
Epoch 29/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step - loss: 0.0105 - mae: 0.0812
Epoch 29: val_loss did not improve from 0.00914
218/218 ━━━━━━━━━━━━━━━━━━━━ 42s 155ms/step - loss: 0.0105 - mae: 0.0812 - val_loss: 0.0096 - val_mae: 0.0770 - learning_rate: 0
Epoch 30/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step - loss: 0.0104 - mae: 0.0804
Epoch 30: val_loss improved from 0.00914 to 0.00884, saving model to best_aqi_lstm.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 156ms/step - loss: 0.0104 - mae: 0.0804 - val_loss: 0.0088 - val_mae: 0.0737 - learning_rate: 0
Restoring model weights from the end of the best epoch: 30.
✓ Training complete!

[STEP 4] MODEL VALIDATION
================================================================================================

24-Hour Forecast Performance:
    Hour       RMSE        MAE        R2
0      1  15.189288  12.016429  0.766097
1      2  14.604160  11.566964  0.783961
2      3  14.537009  11.433958  0.785963
3      4  14.418456  11.301076  0.789502
4      5  14.504444  11.485177  0.786990
5      6  14.564683  11.405335  0.785231
6      7  14.735357  11.495562  0.780223
7      8  14.784225  11.582842  0.778703
8      9  14.573227  11.583986  0.784816
9     10  14.599312  11.381574  0.784361
10    11  14.770255  11.559103  0.779209
11    12  15.282335  11.923701  0.763259

Overall Metrics:
  RMSE: 14.84
  MAE:  11.64
  R²:   0.7769

[STEP 3] DASHBOARD CREATION
================================================================================================
✓ Dashboard saved: aqi_dashboard.png
```



Start coding or generate with AI.



Start coding or generate with AI.

AI-Driven E-commerce Recommendation System

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Data manipulation and processing
from scipy.sparse import csr_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity

# Machine Learning models
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# For downloading dataset
import os
import zipfile
import requests
from io import BytesIO

print("""
╔══════════════════════════════════════════════════════════════════╗
║                                                                  ║
║     AI-DRIVEN E-COMMERCE RECOMMENDATION SYSTEM                   ║
║     Complete Implementation with Dataset Integration             ║
║                                                                  ║
╚══════════════════════════════════════════════════════════════════╝
""")
```

```
🎯 System operational for 24-hour AQI predictions!

╔══════════════════════════════════════════════════════════════════╗
║                                                                  ║
║     AI-DRIVEN E-COMMERCE RECOMMENDATION SYSTEM                   ║
║     Complete Implementation with Dataset Integration             ║
║                                                                  ║
╚══════════════════════════════════════════════════════════════════╝
```

STEP 1: DATA PREPARATION

```
class DatasetManager:
    """Handles downloading and loading the RetailRocket dataset"""

    def __init__(self, data_dir='retailrocket_data'):
        self.data_dir = data_dir
        self.events_path = os.path.join(data_dir, 'events.csv')
        self.item_properties_path = os.path.join(data_dir, 'item_properties_1.csv')
        self.category_tree_path = os.path.join(data_dir, 'category_tree.csv')

    def download_from_kaggle(self):
        """Download dataset using Kaggle API"""
        try:
            import kaggle
            print("\n[📥] Downloading RetailRocket dataset from Kaggle...")

            # Create directory if it doesn't exist
            os.makedirs(self.data_dir, exist_ok=True)

            # Download dataset
            kaggle.api.dataset_download_files(
                'retailrocket/ecommerce-dataset',
                path=self.data_dir,
                unzip=True
```

```
            )
            print("✓ Dataset downloaded successfully!")
            return True

        except ImportError:
            print("\n⚠️  Kaggle API not installed.")
            print("   Install it with: pip install kaggle")
            print("   Then setup credentials from: https://www.kaggle.com/settings")
            return False
        except Exception as e:
            print(f"\n❌ Error downloading dataset: {str(e)}")
            return False

    def use_sample_data(self):
        """Create sample data if dataset is not available"""
        print("\n[🔧] Creating sample dataset for demonstration...")

        os.makedirs(self.data_dir, exist_ok=True)

        # Create sample events data
        np.random.seed(42)
        n_users = 5000
        n_items = 1000
        n_events = 50000

        user_ids = np.random.randint(1, n_users + 1, n_events)
        item_ids = np.random.randint(1, n_items + 1, n_events)
        timestamps = np.random.randint(1000000000000, 1500000000000, n_events)
        events = np.random.choice(['view', 'addtocart', 'transaction'],
                                  n_events, p=[0.85, 0.12, 0.03])

        events_df = pd.DataFrame({
            'timestamp': timestamps,
            'visitorid': user_ids,
            'event': events,
            'itemid': item_ids,
            'transactionid': [np.nan] * n_events
        })

        # Save sample data
        events_df.to_csv(self.events_path, index=False)
        print(f"✓ Sample dataset created with {n_events:,} events")
        print(f"  Users: {n_users:,}, Items: {n_items:,}")

        return True

    def load_dataset(self):
        """Load the dataset from files"""
        print("\n[📁] Loading dataset files...")

        # Try to load existing data
        if os.path.exists(self.events_path):
            events_df = pd.read_csv(self.events_path)
            print(f"✓ Loaded events.csv: {len(events_df):,} records")
        else:
            print("\n⚠️  Dataset not found!")
            print("\nOptions:")
            print("1. Download from Kaggle (requires Kaggle API setup)")
            print("2. Use sample data for demonstration")

            choice = input("\nEnter choice (1 or 2): ").strip()

            if choice == '1':
                if self.download_from_kaggle():
                    events_df = pd.read_csv(self.events_path)
                else:
                    print("\nUsing sample data instead...")
                    self.use_sample_data()
                    events_df = pd.read_csv(self.events_path)
            else:
                self.use_sample_data()
                events_df = pd.read_csv(self.events_path)

        # Load optional files
        item_properties_df = None
        category_tree_df = None
```

```
        if os.path.exists(self.item_properties_path):
            item_properties_df = pd.read_csv(self.item_properties_path)
            print(f"✓ Loaded item_properties: {len(item_properties_df):,} records")

        if os.path.exists(self.category_tree_path):
            category_tree_df = pd.read_csv(self.category_tree_path)
            print(f"✓ Loaded category_tree: {len(category_tree_df):,} records")

        return events_df, item_properties_df, category_tree_df
```

RECOMMENDATION SYSTEM - COMPLETE IMPLEMENTATION

```
class EcommerceRecommendationSystem:
    """
    Complete E-commerce Recommendation System
    Implements: Collaborative Filtering, Matrix Factorization, Content-Based
    """

    def __init__(self):
        self.events_df = None
        self.item_properties_df = None
        self.category_tree_df = None
        self.train_df = None
        self.test_df = None
        self.user_item_matrix = None
        self.user_item_train = None
        self.knn_model = None
        self.svd_model = None
        self.item_similarity = None
        self.performance_results = {}

    # ======================================================================
    # STEP 1: DATA PREPARATION
    # ======================================================================

    def load_and_prepare_data(self, events_df, item_properties_df=None,
                             category_tree_df=None, min_user_interactions=5,
                             min_item_interactions=5):
        """
        Load and preprocess the dataset

        Parameters:
        - min_user_interactions: Minimum interactions per user to keep
        - min_item_interactions: Minimum interactions per item to keep
        """
        print("\n" + "=" * 70)
        print("STEP 1: DATA PREPARATION AND PREPROCESSING")
        print("=" * 70)

        self.events_df = events_df.copy()
        self.item_properties_df = item_properties_df
        self.category_tree_df = category_tree_df

        print(f"\n[1.1] Initial dataset statistics:")
        print(f"  Total events: {len(self.events_df):,}")
        print(f"  Unique users: {self.events_df['visitorid'].nunique():,}")
        print(f"  Unique items: {self.events_df['itemid'].nunique():,}")

        # Data cleaning
        print(f"\n[1.2] Cleaning data...")
        initial_size = len(self.events_df)

        # Remove duplicates
        self.events_df.drop_duplicates(inplace=True)
        print(f"  Removed {initial_size - len(self.events_df):,} duplicates")

        # Create implicit ratings based on event type
        event_weights = {'view': 1, 'addtocart': 3, 'transaction': 5}
        self.events_df['rating'] = self.events_df['event'].map(event_weights).fillna(1)

        # Aggregate multiple interactions
        self.events_df = self.events_df.groupby(['visitorid', 'itemid']).agg({
            'rating': 'sum',
            'timestamp': 'max'
        }).reset_index()
```

```
        # Cap ratings at 10
        self.events_df['rating'] = self.events_df['rating'].clip(upper=10)

        # Filter sparse users and items
        print(f"\n[1.3] Filtering sparse users and items...")
        print(f"  Keeping users with ≥ {min_user_interactions} interactions")
        print(f"  Keeping items with ≥ {min_item_interactions} interactions")

        user_counts = self.events_df['visitorid'].value_counts()
        item_counts = self.events_df['itemid'].value_counts()

        active_users = user_counts[user_counts >= min_user_interactions].index
        active_items = item_counts[item_counts >= min_item_interactions].index

        self.events_df = self.events_df[
            (self.events_df['visitorid'].isin(active_users)) &
            (self.events_df['itemid'].isin(active_items))
        ]

        print(f"\n[1.4] After preprocessing:")
        print(f"  Total interactions: {len(self.events_df):,}")
        print(f"  Unique users: {self.events_df['visitorid'].nunique():,}")
        print(f"  Unique items: {self.events_df['itemid'].nunique():,}")
        print(f"  Sparsity: {self._calculate_sparsity():.4f}")
        print(f"  Avg interactions per user: {len(self.events_df) / self.events_df['visitorid'].nunique():.2f}")
        print(f"  Avg interactions per item: {len(self.events_df) / self.events_df['itemid'].nunique():.2f}")

        return self

    def _calculate_sparsity(self):
        """Calculate matrix sparsity"""
        n_users = self.events_df['visitorid'].nunique()
        n_items = self.events_df['itemid'].nunique()
        n_interactions = len(self.events_df)
        return 1 - (n_interactions / (n_users * n_items))

    def split_data(self, test_size=0.2, random_state=42):
        """Split data into training and testing sets"""
        print(f"\n[1.5] Splitting data (test_size={test_size})...")

        self.train_df, self.test_df = train_test_split(
            self.events_df,
            test_size=test_size,
            random_state=random_state,
            stratify=None
        )

        print(f"  Training set: {len(self.train_df):,} interactions")
        print(f"  Test set: {len(self.test_df):,} interactions")

        return self

    # ========================================================================
    # STEP 2: ALGORITHM IMPLEMENTATION
    # ========================================================================

    def build_user_item_matrix(self):
        """Create user-item interaction matrix"""
        print("\n" + "=" * 70)
        print("STEP 2: ALGORITHM IMPLEMENTATION")
        print("=" * 70)

        print(f"\n[2.1] Building user-item interaction matrix...")

        # Create pivot table for training data
        self.user_item_train = self.train_df.pivot_table(
            index='visitorid',
            columns='itemid',
            values='rating',
            fill_value=0
        )

        # Create full matrix for reference
        self.user_item_matrix = self.events_df.pivot_table(
            index='visitorid',
            columns='itemid',
```

```
        values='rating',
        fill_value=0
    )

    print(f"  Training matrix shape: {self.user_item_train.shape}")
    print(f"  ({self.user_item_train.shape[0]:,} users × {self.user_item_train.shape[1]:,} items)")

    return self

def train_collaborative_filtering(self, n_neighbors=20):
    """
    Train Collaborative Filtering model (User-based k-NN)
    """
    print(f"\n[2.2] Training Collaborative Filtering (k-NN, k={n_neighbors})...")

    # Convert to sparse matrix
    sparse_matrix = csr_matrix(self.user_item_train.values)

    # Train k-NN model
    self.knn_model = NearestNeighbors(
        n_neighbors=min(n_neighbors, self.user_item_train.shape[0] - 1),
        metric='cosine',
        algorithm='brute'
    )
    self.knn_model.fit(sparse_matrix)

    print(f"  ✓ Model trained successfully")

    return self

def train_matrix_factorization(self, n_components=50):
    """
    Train Matrix Factorization model (SVD)
    """
    n_components = min(n_components, min(self.user_item_train.shape) - 1)
    print(f"\n[2.3] Training Matrix Factorization (SVD, components={n_components})...")

    # Apply SVD
    self.svd_model = TruncatedSVD(
        n_components=n_components,
        random_state=42
    )

    self.user_factors = self.svd_model.fit_transform(self.user_item_train)
    self.item_factors = self.svd_model.components_.T

    # Reconstruct matrix
    self.predicted_ratings = np.dot(self.user_factors, self.item_factors.T)

    explained_var = self.svd_model.explained_variance_ratio_.sum()
    print(f"  ✓ Model trained successfully")
    print(f"  Explained variance: {explained_var:.4f}")

    return self

def train_content_based(self):
    """
    Train Content-Based Filtering model
    """
    print(f"\n[2.4] Training Content-Based Filtering...")

    # Calculate item-item similarity from co-occurrence
    item_user_matrix = self.user_item_train.T
    self.item_similarity = cosine_similarity(item_user_matrix)

    print(f"  ✓ Item similarity matrix computed")
    print(f"  Shape: {self.item_similarity.shape}")

    return self

# ========================================================================
# STEP 3: RECOMMENDATION GENERATION
# ========================================================================

def recommend_collaborative(self, user_id, n_recommendations=10):
    """Get recommendations using Collaborative Filtering"""
    if user_id not in self.user_item_train.index:
```

```
            return []

        user_idx = self.user_item_train.index.get_loc(user_id)
        user_vector = csr_matrix(self.user_item_train.iloc[user_idx].values.reshape(1, -1))

        # Find similar users
        distances, indices = self.knn_model.kneighbors(user_vector)

        # Aggregate ratings from similar users
        similar_users_ratings = self.user_item_train.iloc[indices[0][1:]].sum(axis=0)
        user_ratings = self.user_item_train.iloc[user_idx]

        # Remove already rated items
        recommendations = similar_users_ratings[user_ratings == 0].nlargest(n_recommendations)

        return recommendations.index.tolist()

    def recommend_matrix_factorization(self, user_id, n_recommendations=10):
        """Get recommendations using Matrix Factorization"""
        if user_id not in self.user_item_train.index:
            return []

        user_idx = self.user_item_train.index.get_loc(user_id)
        user_pred = self.predicted_ratings[user_idx].copy()

        # Remove already rated items
        rated_items = self.user_item_train.iloc[user_idx] > 0
        user_pred[rated_items] = -np.inf

        # Get top items
        top_items_idx = np.argsort(user_pred)[-n_recommendations:][::-1]

        return self.user_item_train.columns[top_items_idx].tolist()

    def recommend_content_based(self, user_id, n_recommendations=10):
        """Get recommendations using Content-Based Filtering"""
        if user_id not in self.user_item_train.index:
            return []

        user_ratings = self.user_item_train.loc[user_id]
        liked_items = user_ratings[user_ratings > 0].index.tolist()

        if len(liked_items) == 0:
            return []

        # Get item indices
        item_indices = [self.user_item_train.columns.get_loc(item)
                        for item in liked_items]

        # Calculate average similarity
        similarity_scores = self.item_similarity[item_indices].mean(axis=0)

        # Remove already rated items
        for idx in item_indices:
            similarity_scores[idx] = -1

        # Get top items
        top_items_idx = np.argsort(similarity_scores)[-n_recommendations:][::-1]

        return self.user_item_train.columns[top_items_idx].tolist()

    # ========================================================================
    # STEP 4: PERFORMANCE EVALUATION
    # ========================================================================

    def evaluate_models(self, n_users_sample=500, top_k=10):
        """
        Evaluate all models on test set
        """
        print("\n" + "=" * 70)
        print("STEP 3: PERFORMANCE EVALUATION")
        print("=" * 70)

        # Sample test users
        test_users = list(set(self.test_df['visitorid'].unique()) &
                          set(self.user_item_train.index))
```

```
        if len(test_users) > n_users_sample:
            test_users = np.random.choice(test_users, n_users_sample, replace=False)

        print(f"\n[3.1] Evaluating on {len(test_users):,} test users...")

        # Evaluate each model
        models = {
            'Collaborative Filtering': self.recommend_collaborative,
            'Matrix Factorization': self.recommend_matrix_factorization,
            'Content-Based': self.recommend_content_based
        }

        results = {}

        for model_name, recommend_func in models.items():
            print(f"\n[3.{list(models.keys()).index(model_name) + 2}] Evaluating {model_name}...")
            metrics = self._evaluate_single_model(recommend_func, test_users, top_k)
            results[model_name] = metrics

            print(f"  Precision@{top_k}: {metrics['precision']:.4f}")
            print(f"  Recall@{top_k}: {metrics['recall']:.4f}")
            print(f"  F1-Score: {metrics['f1']:.4f}")
            print(f"  Coverage: {metrics['coverage']:.4f}")

        self.performance_results = results

        # Display comparison
        self._display_comparison()

        return self

    def _evaluate_single_model(self, recommend_func, test_users, top_k):
        """Evaluate a single recommendation model"""
        precisions = []
        recalls = []
        all_recommended = set()

        for user_id in test_users:
            # Get actual items from test set
            actual_items = set(
                self.test_df[self.test_df['visitorid'] == user_id]['itemid'].values
            )

            if len(actual_items) == 0:
                continue

            # Get recommendations
            try:
                recommended_items = set(recommend_func(user_id, top_k))
                all_recommended.update(recommended_items)
            except:
                continue

            if len(recommended_items) == 0:
                continue

            # Calculate metrics
            hits = len(actual_items & recommended_items)
            precision = hits / len(recommended_items) if len(recommended_items) > 0 else 0
            recall = hits / len(actual_items) if len(actual_items) > 0 else 0

            precisions.append(precision)
            recalls.append(recall)

        # Average metrics
        avg_precision = np.mean(precisions) if precisions else 0
        avg_recall = np.mean(recalls) if recalls else 0
        f1 = (2 * avg_precision * avg_recall / (avg_precision + avg_recall)) \
             if (avg_precision + avg_recall) > 0 else 0
        coverage = len(all_recommended) / self.user_item_train.shape[1]

        return {
            'precision': avg_precision,
            'recall': avg_recall,
            'f1': f1,
            'coverage': coverage
        }
```

```
    def _display_comparison(self):
        """Display model comparison"""
        print("\n" + "=" * 70)
        print("MODEL PERFORMANCE COMPARISON")
        print("=" * 70 + "\n")

        df = pd.DataFrame(self.performance_results).T
        print(df.to_string())

        best_model = df['f1'].idxmax()
        print(f"\n🏆 Best Model: {best_model}")
        print(f"   F1-Score: {df.loc[best_model, 'f1']:.4f}")

        return df

    # ========================================================================
    # STEP 5: DEMONSTRATION
    # ========================================================================

    def demonstrate_system(self, n_users=5, n_recommendations=10):
        """
        Demonstrate the recommendation system
        """
        print("\n" + "=" * 70)
        print("STEP 4: PROTOTYPE DEVELOPMENT & DEMONSTRATION")
        print("=" * 70)

        sample_users = list(self.user_item_train.index[:n_users])

        print(f"\n[4.1] Generating recommendations for {n_users} sample users...\n")

        for i, user_id in enumerate(sample_users, 1):
            print("—" * 70)
            print(f"USER #{i} (ID: {user_id})")
            print("—" * 70)

            # User history
            user_history = self.events_df[self.events_df['visitorid'] == user_id]
            print(f"\n📊 User History:")
            print(f"   Total interactions: {len(user_history)}")
            print(f"   Unique items viewed: {user_history['itemid'].nunique()}")

            # Recent items
            recent_items = user_history.nlargest(5, 'timestamp')['itemid'].tolist()
            print(f"   Recent items: {recent_items[:5]}")

            # Get recommendations from each model
            print(f"\n🎯 Recommendations (Top {n_recommendations}):\n")

            cf_recs = self.recommend_collaborative(user_id, n_recommendations)
            mf_recs = self.recommend_matrix_factorization(user_id, n_recommendations)
            cb_recs = self.recommend_content_based(user_id, n_recommendations)

            print(f"   Collaborative Filtering:")
            for j, item in enumerate(cf_recs[:5], 1):
                print(f"      {j}. Item {item}")

            print(f"\n   Matrix Factorization:")
            for j, item in enumerate(mf_recs[:5], 1):
                print(f"      {j}. Item {item}")

            print(f"\n   Content-Based:")
            for j, item in enumerate(cb_recs[:5], 1):
                print(f"      {j}. Item {item}")

            print()

        return self

    def generate_final_report(self):
        """Generate final summary report"""
        print("\n" + "=" * 70)
        print("STEP 5: FINAL SUMMARY REPORT")
        print("=" * 70)

        print("\n📊 DATASET STATISTICS")
```

```
        print(f"   Total interactions: {len(self.events_df):,}")
        print(f"   Unique users: {self.events_df['visitorid'].nunique():,}")
        print(f"   Unique items: {self.events_df['itemid'].nunique():,}")
        print(f"   Data sparsity: {self._calculate_sparsity():.4f}")

        print("\n🤖 IMPLEMENTED ALGORITHMS")
        print("   ✓ Collaborative Filtering (User-based k-NN)")
        print("   ✓ Matrix Factorization (Truncated SVD)")
        print("   ✓ Content-Based Filtering (Item Similarity)")

        print("\n📈 MODEL PERFORMANCE SUMMARY")
        if self.performance_results:
            for model, metrics in self.performance_results.items():
                print(f"\n   {model}:")
                for metric, value in metrics.items():
                    print(f"      {metric.capitalize()}: {value:.4f}")

        print("\n✅ ASSESSMENT COMPLETION STATUS")
        print("   ✓ Step 1: Data Preparation - COMPLETE")
        print("   ✓ Step 2: Algorithm Implementation - COMPLETE")
        print("   ✓ Step 3: Performance Evaluation - COMPLETE")
        print("   ✓ Step 4: Prototype Development - COMPLETE")
        print("   ✓ Step 5: Testing & Demonstration - COMPLETE")
```

MAIN EXECUTION PIPELINE

```
def main():
    """
    Complete execution pipeline for the recommendation system
```