PART A: DATA LOADING & PREPROCESSING

Cell 1 — Enhanced Imports

```
# 📸 CELL 1: Install required packages
!pip install holidays optuna scikit-learn seaborn shap scipy
```

```
Requirement already satisfied: holidays in /usr/local/lib/python3.12/dist-packages (0.85)
Requirement already satisfied: optuna in /usr/local/lib/python3.12/dist-packages (4.6.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-packages (0.50.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (1.16.3)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.12/dist-packages (from holidays) (2.9.0.post0)
Requirement already satisfied: alembic>=1.5.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (1.17.2)
Requirement already satisfied: colorlog in /usr/local/lib/python3.12/dist-packages (from optuna) (6.10.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (25.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from optuna) (2.0.44)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (from optuna) (6.0.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.12/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.12/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/dist-packages (from shap) (3.1.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: Mako in /usr/local/lib/python3.12/dist-packages (from alembic>=1.5.0->optuna) (1.3.10)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->s
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->seabo
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib!=3.6.1,>=3.4->s
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba>=0.54->sha
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.2->seaborn) (2025.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil->holidays) (1.17.0
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from sqlalchemy>=1.4.2->optuna) (3.2
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.12/dist-packages (from Mako->alembic>=1.5.0->opt
```

```
# 📸 CELL 2: Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, BatchNormalization, Input
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
import tensorflow as tf
import warnings
warnings.filterwarnings('ignore')

plt.style.use("seaborn-v0_8")
sns.set_palette("husl")

print("✅ All packages loaded successfully")
```

```
✅ All packages loaded successfully
```

```
# 📸 CELL 3: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
print("Successfully mounted!")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=T
Successfully mounted!
```

Cell 2 — Load Data

```
# 📸 CELL 4: Load dataset
csv_file_path = "/content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/best-dataset/dambulla_market_dataset.csv"
# Load data
df = pd.read_csv(csv_file_path, parse_dates=["date"])
df = df.sort_values("date").reset_index(drop=True)
df.set_index("date", inplace=True)

print(f"✅ Data loaded successfully")
print(f"📊 Shape: {df.shape}")
print(f"📅 Date range: {df.index.min()} to {df.index.max()}")
print(f"📋 Columns: {len(df.columns)}")
print(f"\n🔍 First few columns:\n{df.columns.tolist()[:10]}")
print(f"\n📈 Sample data:")
print(df.head())
```

```
2020-01-03                            0.84                          3.87
2020-01-04                            0.64                          2.41
2020-01-05                            0.28                          0.73

            kandapola_mean_precipitation_mm  \
date
2020-01-01                             1.24
2020-01-02                             3.16
2020-01-03                             0.11
2020-01-04                             1.07
2020-01-05                             0.13

            nuwaraeliya_mean_precipitation_mm  ragala_mean_precipitation_mm  \
date
2020-01-01                               0.18                          0.16
2020-01-02                               0.71                          0.61
2020-01-03                               1.11                          1.01
2020-01-04                               0.76                          0.71
2020-01-05                               0.30                          0.29

            ...  hanguranketha_supply_factor  pussellawa_supply_factor  \
date        ...
2020-01-01  ...                           -1                        -1
2020-01-02  ...                            0                         0
2020-01-03  ...                            0                         0
2020-01-04  ...                           -1                        -1
2020-01-05  ...                           -1                        -1

            yatawaththa_supply_factor  thalawakale_supply_factor  \
date
2020-01-01                         -1                         -1
2020-01-02                          0                          0
2020-01-03                          0                          0
2020-01-04                         -1                         -1
2020-01-05                         -1                         -1

            mandaramnuwara_supply_factor  kandy_supply_factor  \
date
2020-01-01                            -1                   -1
2020-01-02                             0                    0
2020-01-03                             0                    0
2020-01-04                            -1                   -1
2020-01-05                            -1                   -1

            mathale_supply_factor  walimada_supply_factor  \
date
2020-01-01                     -1                      -1
2020-01-02                      0                       0
2020-01-03                      0                       0
2020-01-04                     -1                      -1
2020-01-05                     -1                      -1

            marassana_supply_factor  puttalam_supply_factor
date
2020-01-01                       -1                      -1
2020-01-02                        0                       0
2020-01-03                        0                       0
2020-01-04                       -1                      -1
```

Cell 3.5 — Remove Non-Transport Fuel Columns

Rationale: Agricultural transport uses only petrol (Lp_95, Lp_92) and diesel (lad, lsd). Kerosene (lk, lik) and furnace oils (fur_*) are for industrial/heating purposes, not relevant for carrot transportation.

```
# 📸 CELL 4B: Remove non-transport fuel columns
print("="*60)
print("🔧 FUEL COLUMN CLEANUP")
print("="*60)

# Define non-transport fuels (kerosene + furnace oils)
non_transport_fuels = [
    'lk',           # Kerosene (used for lighting/cooking, not transport)
    'lik',          # Industrial Kerosene (industrial use)
    'fur_800',      # Furnace Oil 800 (industrial heating)
    'fur_1500_high',# Furnace Oil 1500 High (industrial heating)
    'fur_1500_Low'  # Furnace Oil 1500 Low (industrial heating)
]

# Find which columns actually exist in the dataset
existing_non_transport = [col for col in non_transport_fuels if col in df.columns]

print(f"\n🗑️  Removing {len(existing_non_transport)} non-transport fuel columns:")
for col in existing_non_transport:
    print(f"   ❌ {col}")

# Remove columns
df = df.drop(columns=existing_non_transport, errors='ignore')

# Show remaining fuel columns
remaining_fuel_cols = [col for col in df.columns if any(x in col for x in ['Lp_', 'lad', 'lsd', 'fur_', 'lk', 'lik'])]

print(f"\n✅ Remaining fuel columns (transport-relevant): {len(remaining_fuel_cols)}")
for col in remaining_fuel_cols:
    if 'Lp_95' in col or 'Lp_92' in col:
        print(f"   🚗 {col} (Petrol)")
    elif 'lad' in col or 'lsd' in col:
        print(f"   🚛 {col} (Diesel)")
    else:
        print(f"   ⚠️  {col} (Check this)")

print(f"\n📊 Dataset shape after cleanup: {df.shape}")
print(f"   Removed: {len(existing_non_transport)} columns")
```

```
============================================================
🔧 FUEL COLUMN CLEANUP
============================================================

🗑️  Removing 5 non-transport fuel columns:
   ❌ lk
   ❌ lik
   ❌ fur_800
   ❌ fur_1500_high
   ❌ fur_1500_Low

✅ Remaining fuel columns (transport-relevant): 4
   🚗 Lp_95 (Petrol)
   🚗 Lp_92 (Petrol)
   🚛 lad (Diesel)
   🚛 lsd (Diesel)

📊 Dataset shape after cleanup: (2017, 41)
   Removed: 5 columns
```

Cell 3 — Data Quality Check

```
# 📸 CELL 5: Data quality check
# Check for missing values
missing_counts = df.isnull().sum()
missing_pct = (missing_counts / len(df)) * 100

if missing_counts.sum() > 0:
    print("⚠️ Missing values found:")
    print(missing_pct[missing_pct > 0])
else:
    print("✅ No missing values found")

# Check data types
```

```
print(f"\n📊 Data types:")
print(df.dtypes.value_counts())

# Basic statistics for target
print(f"\n💰 Carrot Price Statistics:")
print(df['carrot_price'].describe())

# Check for outliers
Q1 = df['carrot_price'].quantile(0.25)
Q3 = df['carrot_price'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['carrot_price'] < Q1 - 1.5*IQR) | (df['carrot_price'] > Q3 + 1.5*IQR)]
print(f"\n📉 Outliers detected: {len(outliers)} ({len(outliers)/len(df)*100:.2f}%)")
```

```
✅ No missing values found

📊 Data types:
float64    21
int64      20
Name: count, dtype: int64

💰 Carrot Price Statistics:
count    2017.000000
mean      236.795537
std       177.849380
min        53.000000
25%       125.000000
50%       175.000000
75%       305.000000
max      1950.000000
Name: carrot_price, dtype: float64

📉 Outliers detected: 103 (5.11%)
```

Cell 4 — Transform Supply Factors (1,0,-1 → 2,1,0)

```
# 📸 CELL 6: Transform supply factors
# Identify supply factor columns
supply_cols = [col for col in df.columns if 'supply_factor' in col]

print(f"🔄 Transforming {len(supply_cols)} supply factor columns")
print(f"📋 Supply columns: {supply_cols}")

# Create a copy for transformation
df_transformed = df.copy()

# Transform: 1→2 (high), -1→1 (normal), 0→0 (low)
supply_mapping = {1: 2, -1: 1, 0: 0}

for col in supply_cols:
    df_transformed[col] = df_transformed[col].map(supply_mapping)

print("\n✅ Supply factors transformed:")
print("Original encoding: 1=HIGH, 0=LOW, -1=NORMAL")
print("New encoding: 2=HIGH, 1=NORMAL, 0=LOW")

# Verify transformation
print("\n🔍 Verification (first supply column):")
print(f"Original values: {df[supply_cols[0]].unique()}")
print(f"Transformed values: {df_transformed[supply_cols[0]].unique()}")
```

```
🔄 Transforming 15 supply factor columns
📋 Supply columns: ['nuwaraeliya_supply_factor', 'jaffna_supply_factor', 'ragala_supply_factor', 'kandapola_supply_factor',

✅ Supply factors transformed:
Original encoding: 1=HIGH, 0=LOW, -1=NORMAL
New encoding: 2=HIGH, 1=NORMAL, 0=LOW

🔍 Verification (first supply column):
Original values: [ 1  0 -1]
Transformed values: [2 0 1]
```

PART B: FEATURE ENGINEERING & SELECTION

Cell 5 — Create Lag Features

```
print("🔧 Creating lag features...")

df_features = df_transformed.copy()

# ============================================
# 1. PRICE LAG FEATURES
# ============================================
df_features['price_lag_1'] = df_features['carrot_price'].shift(1)
df_features['price_lag_7'] = df_features['carrot_price'].shift(7)
df_features['price_rolling_mean_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).mean()
df_features['price_rolling_mean_14'] = df_features['carrot_price'].rolling(window=14, min_periods=1).mean()
df_features['price_rolling_std_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).std()
df_features['price_change'] = df_features['carrot_price'].diff()
df_features['price_change_pct'] = df_features['carrot_price'].pct_change()

# ============================================
# 2. PRECIPITATION LAG FEATURES (for each region)
# ============================================
precip_cols = [col for col in df.columns if 'precipitation' in col]

for col in precip_cols:
    # Lag features
    df_features[f'{col}_lag_1'] = df_features[col].shift(1)
    df_features[f'{col}_lag_3'] = df_features[col].shift(3)
    # Rolling sum (total rain last week)
    df_features[f'{col}_rolling_sum_7'] = df_features[col].rolling(window=7, min_periods=1).sum()

# ============================================
# 3. SUPPLY FACTOR LAG FEATURES
# ============================================
for col in supply_cols:
    df_features[f'{col}_lag_1'] = df_features[col].shift(1)
    df_features[f'{col}_rolling_mean_7'] = df_features[col].rolling(window=7, min_periods=1).mean()

# ============================================
# 4. FUEL PRICE LAG FEATURES
# ============================================
fuel_cols = [col for col in df.columns if 'fur_' in col or any(x in col for x in ['Lp_', 'lad', 'lsd', 'lk', 'lik'])]

for col in fuel_cols:
    df_features[f'{col}_lag_1'] = df_features[col].shift(1)

# ============================================
# 5. TEMPORAL FEATURES
# ============================================
df_features['day_of_week'] = df_features.index.dayofweek
df_features['day_of_month'] = df_features.index.day
df_features['month'] = df_features.index.month
df_features['quarter'] = df_features.index.quarter
df_features['is_weekend'] = (df_features.index.dayofweek >= 5).astype(int)

# ============================================
# 6. INTERACTION FEATURES
# ============================================
df_features['demand_x_trading'] = df_features['dambulla_demand'] * df_features['dambulla_is_trading_activities_high_or_low

# Fill NaN values created by lagging (forward fill)
df_features = df_features.fillna(method='ffill').fillna(method='bfill')

print(f"✅ Feature engineering completed")
print(f"📊 Total features created: {df_features.shape[1]}")
print(f"📈 Original features: {df.shape[1]}")
print(f"🆕 New features: {df_features.shape[1] - df.shape[1]}")
```

```
🔧 Creating lag features...
✅ Feature engineering completed
📊 Total features created: 139
📈 Original features: 41
🆕 New features: 98
```

Cell 6 — Correlation Analysis & Precipitation Grouping

```
# Calculate correlation with target
correlations = df_features.corr()['carrot_price'].abs().sort_values(ascending=False)
```

```
print("="*60)
print("📊 TOP 20 FEATURES CORRELATED WITH CARROT PRICE")
print("="*60)
print(correlations.head(20))

# Visualize top correlations
plt.figure(figsize=(10, 8))
correlations.head(20).plot(kind='barh', color='steelblue')
plt.title('Top 20 Features Correlated with Carrot Price', fontsize=14, fontweight='bold')
plt.xlabel('Absolute Correlation')
plt.tight_layout()
plt.show()

# ============================================
# PRECIPITATION REGION GROUPING
# ============================================
print("\n" + "="*60)
print("🌧️ ANALYZING PRECIPITATION REGIONS")
print("="*60)

# Get all precipitation columns (original, not lagged)
base_precip_cols = [col for col in df_features.columns
                    if 'precipitation' in col and 'lag' not in col and 'rolling' not in col]

# Calculate correlation between precipitation regions
precip_df = df_features[base_precip_cols]
precip_corr_matrix = precip_df.corr()

# Find highly correlated regions (correlation > 0.8)
high_corr_pairs = []
for i in range(len(precip_corr_matrix.columns)):
    for j in range(i+1, len(precip_corr_matrix.columns)):
        if abs(precip_corr_matrix.iloc[i, j]) > 0.8:
            high_corr_pairs.append((
                precip_corr_matrix.columns[i],
                precip_corr_matrix.columns[j],
                precip_corr_matrix.iloc[i, j]
            ))

print(f"\n🔗 Highly correlated precipitation regions (>0.8):")
for pair in high_corr_pairs[:10]:
    print(f"  {pair[0]} ↔ {pair[1]}: {pair[2]:.3f}")

# Calculate correlation with carrot price
precip_target_corr = df_features[base_precip_cols].corrwith(df_features['carrot_price']).abs()
precip_target_corr = precip_target_corr.sort_values(ascending=False)

print(f"\n📈 Precipitation regions correlation with price:")
print(precip_target_corr.head(10))

# Define regional groups based on correlation
# Keep top regions and group highly correlated ones
PRECIP_GROUPS = {
    'central_highland': [col for col in base_precip_cols
                         if any(x in col for x in ['nuwaraeliya', 'kandapola', 'ragala', 'thalawakale', 'pussellawa', 'hang
    'uva_province': [col for col in base_precip_cols
                     if any(x in col for x in ['bandarawela', 'walimada'])],
    'northern': [col for col in base_precip_cols
                 if 'jaffna' in col],
    'other': [col for col in base_precip_cols
              if not any(x in col for x in ['nuwaraeliya', 'kandapola', 'ragala', 'thalawakale',
                                            'pussellawa', 'hanguranketha', 'bandarawela',
                                            'walimada', 'jaffna'])]
}

# Create grouped precipitation features
for group_name, cols in PRECIP_GROUPS.items():
    if len(cols) > 0:
        df_features[f'precip_{group_name}_mean'] = df_features[cols].mean(axis=1)
        df_features[f'precip_{group_name}_max'] = df_features[cols].max(axis=1)

        # Add lagged versions of grouped features
        df_features[f'precip_{group_name}_mean_lag_1'] = df_features[f'precip_{group_name}_mean'].shift(1)
        df_features[f'precip_{group_name}_mean_lag_3'] = df_features[f'precip_{group_name}_mean'].shift(3)
        df_features[f'precip_{group_name}_rolling_sum_7'] = df_features[f'precip_{group_name}_mean'].rolling(7).sum()

df_features = df_features.fillna(method='ffill')
```

```
print(f"\n✅ Created {len(PRECIP_GROUPS)} precipitation groups")
print(f"📊 New grouped features: {sum([len(cols) for cols in PRECIP_GROUPS.values()])}")
```

```
============================================================
📊 TOP 20 FEATURES CORRELATED WITH CARROT PRICE
============================================================
carrot_price                                1.000000
price_lag_1                                 0.960605
price_rolling_mean_7                        0.952164
price_rolling_mean_14                       0.922189
price_lag_7                                 0.865969
price_rolling_std_7                         0.720441
lsd                                         0.324175
lsd_lag_1                                   0.323366
Lp_95                                       0.314247
Lp_95_lag_1                                 0.314019
Lp_92                                       0.307039
Lp_92_lag_1                                 0.306769
lad                                         0.306155
lad_lag_1                                   0.305529
quarter                                     0.232633
month                                       0.227405
kandapola_mean_precipitation_mm_rolling_sum_7    0.165197
price_change                                0.141554
ragala_mean_precipitation_mm_rolling_sum_7  0.115500
kandapola_mean_precipitation_mm             0.113914
Name: carrot_price, dtype: float64
```

Cell 7 — Random Forest Feature Importance

```
print("🌲 Training Random Forest for feature importance...")

# Prepare data for Random Forest
X_rf = df_features.drop('carrot_price', axis=1)
y_rf = df_features['carrot_price']

# Remove any infinite values
X_rf = X_rf.replace([np.inf, -np.inf], np.nan).fillna(method='ffill')

# Train Random Forest
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=15,
    min_samples_split=10,
    min_samples_leaf=5,
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X_rf, y_rf)

# Get feature importance
feature_importance = pd.DataFrame({
    'feature': X_rf.columns,
    'importance': rf_model.feature_importances_
}).sort_values('importance', ascending=False)

print("\n" + "="*60)
print("🏆 TOP 30 MOST IMPORTANT FEATURES (Random Forest)")
print("="*60)
print(feature_importance.head(30))

# Visualize top 20 features
plt.figure(figsize=(12, 8))
top_20_features = feature_importance.head(20)
plt.barh(range(len(top_20_features)), top_20_features['importance'], color='forestgreen')
plt.yticks(range(len(top_20_features)), top_20_features['feature'])
plt.xlabel('Importance Score')
plt.title('Top 20 Most Important Features (Random Forest)', fontsize=14, fontweight='bold')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

# Save feature importance
feature_importance.to_csv('/content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/feature_import
print("\n✅ Feature importance saved to: /content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/
```

```
  bandarawela_mean_precipitation_mm ↔ nuwaraeliya_mean_precipitation_mm: 0.983
  bandarawela_mean_precipitation_mm ↔ ragala_mean_precipitation_mm: 0.994
  nuwaraeliya_mean_precipitation_mm ↔ ragala_mean_precipitation_mm: 0.997
  bandarawela_mean_precipitation_mm.1 ↔ hanguranketha_mean_precipitation_mm: 0.983
  bandarawela_mean_precipitation_mm.1 ↔ kandy_mean_precipitation_mm: 0.956
  bandarawela_mean_precipitation_mm.1 ↔ mandaramnuwara_mean_precipitation_mm: 0.986
  bandarawela_mean_precipitation_mm.1 ↔ marassana_mean_precipitation_mm: 0.977
```

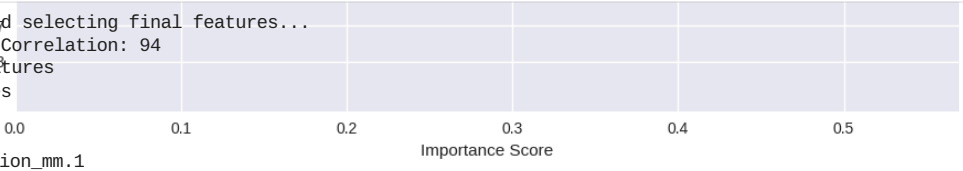
```
  bandarawela_mean_precipitation_mm.1 ↔ mathale_mean_precipitation_mm: 0.950
  bandarawela_mean_precipitation_mm.1 ↔ pussellawa_mean_precipitation_mm: 0.959
  bandarawela_mean_precipitation_mm.1 ↔ thalawakale_mean_precipitation_mm: 0.967

📈 Precipitation regions correlation with price:
kandapola_mean_precipitation_mm         0.113914
jaffna_mean_precipitation_mm            0.048725
hanguranketha_mean_precipitation_mm     0.045279
marassana_mean_precipitation_mm         0.044990
mathale_mean_precipitation_mm           0.044199
mandaramnuwara_mean_precipitation_mm    0.044081
yatawatta_mean_precipitation_mm         0.043907
kandy_mean_precipitation_mm             0.043011
walimada_mean_precipitation_mm          0.041734
pussellawa_mean_precipitation_mm        0.041602
dtype: float64

✅ Created 4 precipitation groups
📊 New grouped features: 17
```

```
🌲 Training Random Forest for feature importance...

============================================================
🏆 TOP 30 MOST IMPORTANT FEATURES (Random Forest)
============================================================
```

Cell 8 — Multicollinearity Check & Final Feature Selection

```
print("🔍 Checking multicollinearity and selecting final features...")

# Combine Random Forest importance and correlation
rf_top_features = set(feature_importance.head(50)['feature'].tolist())
corr_top_features = set(correlations[correlations > 0.05].index.tolist())

# Get union of both methods
candidate_features = list(rf_top_features.union(corr_top_features))
candidate_features = [f for f in candidate_features if f != 'carrot_price']

print(f"📊 Candidate features from RF + Correlation: {len(candidate_features)}")

# Check multicollinearity among candidate features
X_candidates = df_features[candidate_features]
candidates_corr = X_candidates.corr().abs()

# Remove highly correlated features (keep the one more correlated with target)
features_to_remove = set()
for i in range(len(candidates_corr.columns)):
    for j in range(i+1, len(candidates_corr.columns)):
        if candidates_corr.iloc[i, j] > 0.90:  # Very high correlation
            col_i = candidates_corr.columns[i]
            col_j = candidates_corr.columns[j]

            # Keep the one more correlated with target
            corr_i = abs(df_features[col_i].corr(df_features['carrot_price']))
            corr_j = abs(df_features[col_j].corr(df_features['carrot_price']))

            if corr_i < corr_j:
                features_to_remove.add(col_i)
            else:
                features_to_remove.add(col_j)

# Remove multicollinear features
final_features = [f for f in candidate_features if f not in features_to_remove]

print(f"🗑️ Removed {len(features_to_remove)} multicollinear features")
print(f"✅ Final feature set: {len(final_features)} features")

# Ensure key features are included
key_features = ['is_market_open', 'dambulla_demand', 'dambulla_is_trading_activities_high_or_low',
                'price_lag_1', 'price_lag_7', 'price_rolling_mean_7']

for kf in key_features:
    if kf in df_features.columns and kf not in final_features:
        final_features.append(kf)

print(f"\n📋 Final selected features ({len(final_features)}):")
for i, feat in enumerate(sorted(final_features)[:30], 1):
    print(f"  {i}. {feat}")
if len(final_features) > 30:
    print(f"  ... and {len(final_features) - 30} more")

# Create final feature dataframe
df_final = df_features[final_features + ['carrot_price']].copy()

print(f"\n✅ Final dataset shape: {df_final.shape}")
```

```
🔍 Checking multicollinearity and selecting final features...
📊 Candidate features from RF + Correlation: 94
🗑️ Removed 49 multicollinear features
✅ Final feature set: 45 features

📋 Final selected features (50):
  1. bandarawela_mean_precipitation_mm.1
  2. bandarawela_mean_precipitation_mm_lag_3
  3. bandarawela_mean_precipitation_mm_rolling_mean_7
  4. dambulla_demand
  5. dambulla_is_trading_activities_high_or_low
  6. day_of_month
  7. day_of_week
  8. hanguranketha_mean_precipitation_mm_rolling_sum_7
  9. is_dambulla_increase
```

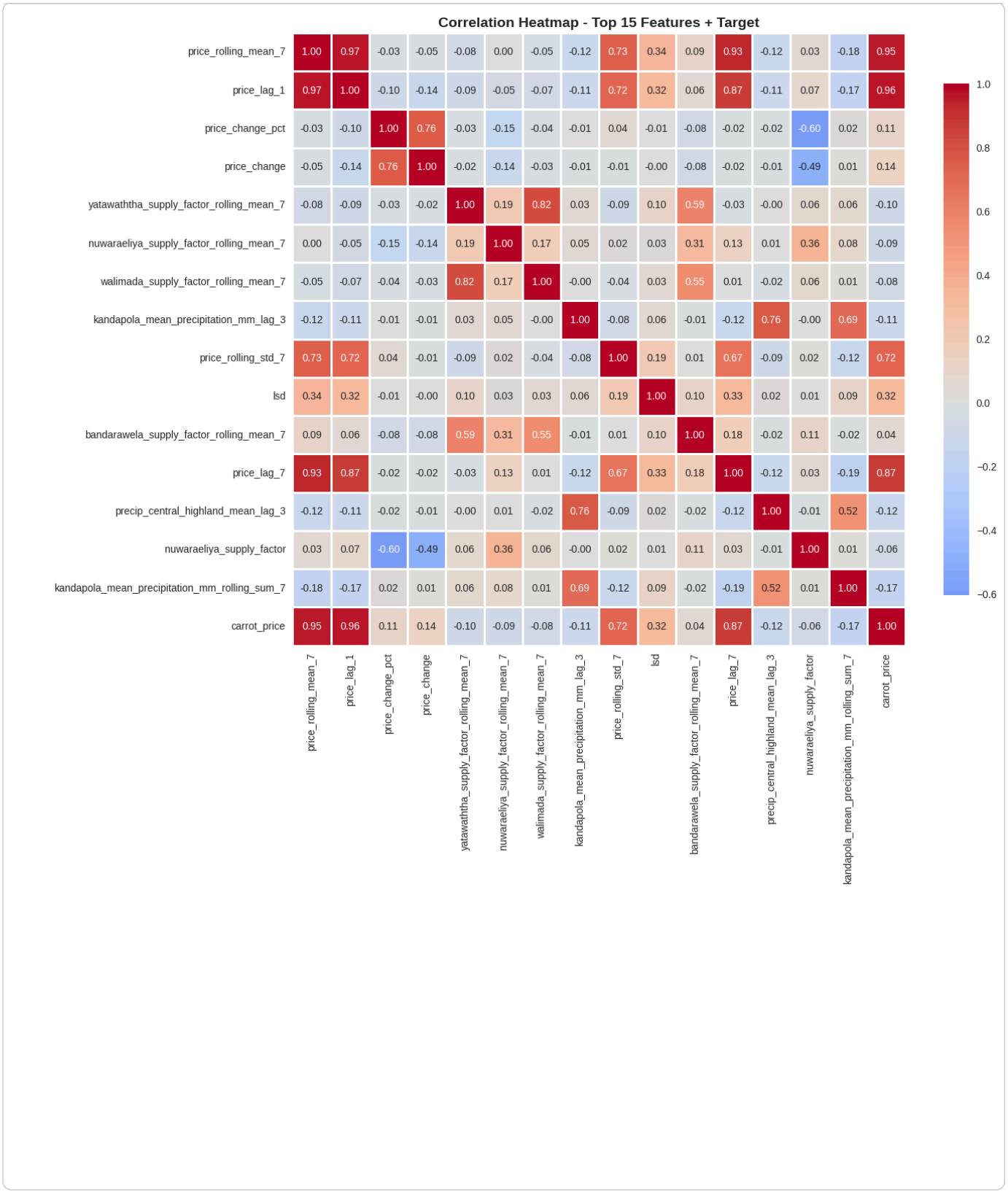



```
✅ Feature importance saved to: /content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/feature_im
```

```
  10. is_market_open
  11. jaffna_mean_precipitation_mm_lag_1
  12. jaffna_mean_precipitation_mm_lag_3
  13. jaffna_mean_precipitation_mm_rolling_sum_7
  14. jaffna_supply_factor
  15. jaffna_supply_factor_rolling_mean_7
  16. kalpitiya_mean_precipitation_mm_lag_1
  17. kalpitiya_mean_precipitation_mm_rolling_sum_7
  18. kandapola_mean_precipitation_mm
  19. kandapola_mean_precipitation_mm_lag_1
  20. kandapola_mean_precipitation_mm_lag_3
  21. kandapola_mean_precipitation_mm_rolling_sum_7
  22. kandapola_supply_factor
  23. kandapola_supply_factor_rolling_mean_7
  24. lsd
  25. nuwaraeliya_mean_precipitation_mm
  26. nuwaraeliya_mean_precipitation_mm_lag_1
  27. nuwaraeliya_supply_factor
  28. nuwaraeliya_supply_factor_rolling_mean_7
  29. precip_central_highland_mean_lag_1
  30. precip_central_highland_mean_lag_3
  ... and 20 more

✅ Final dataset shape: (2017, 51)
```

Cell 9 — Correlation Heatmap

```
# Create correlation heatmap for top features
# Filter feature_importance to only include features present in df_final
# excluding the target variable itself
features_in_df_final = [col for col in df_final.columns if col != 'carrot_price']
filtered_feature_importance = feature_importance[feature_importance['feature'].isin(features_in_df_final)]

# Get the top 15 features from the filtered list
top_15_features = filtered_feature_importance.head(15)['feature'].tolist()

corr_subset = df_final[top_15_features + ['carrot_price']].corr()

plt.figure(figsize=(14, 12))
sns.heatmap(corr_subset, annot=True, fmt='.2f', cmap='coolwarm', center=0,
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Heatmap - Top 15 Features + Target', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



PART B.2: ADVANCED FEATURE SELECTION (IDENTICAL TO RANDOM FOREST)

Methodology: To ensure fair comparison between LSTM and Random Forest, we use the EXACT SAME 4-stage feature selection pipeline:

1. **Correlation Analysis** - Identify features with strong linear relationships
2. **Mutual Information** - Capture non-linear dependencies
3. **Combined Scoring** - Weight: 60% RF + 30% MI + 10% Correlation
4. **RFE + SelectFromModel** - Dual validation with intersection

This approach ensures both models compete on equal footing.

```
# 📸 CELL 23: ADVANCED FEATURE SELECTION - STAGE 1
print("="*80)
print("🎯 ADVANCED FEATURE SELECTION - STAGE 1: MUTUAL INFORMATION")
print("="*80)

from sklearn.feature_selection import mutual_info_regression, RFE, SelectFromModel

# Prepare data
X_all = df_features.drop('carrot_price', axis=1)
y_all = df_features['carrot_price']

# Remove infinite values and handle NaNs properly
X_all = X_all.replace([np.inf, -np.inf], np.nan)

# Fill NaNs: forward fill first, then backward fill, then fill remaining with 0
X_all = X_all.fillna(method='ffill').fillna(method='bfill').fillna(0)

# Verify no NaNs remain
print(f"\n🔍 Data cleaning check:")
print(f"   NaN count: {X_all.isnull().sum().sum()}")
print(f"   Inf count: {np.isinf(X_all).sum().sum()}")

print(f"\n📊 Dataset shape: {X_all.shape}")
print(f"   Features: {X_all.shape[1]}")
print(f"   Samples: {X_all.shape[0]}")

# Calculate Mutual Information
print("\n⏳ Computing Mutual Information scores...")
mi_scores = mutual_info_regression(X_all, y_all, random_state=42, n_neighbors=5)

mi_importance = pd.DataFrame({
    'feature': X_all.columns,
    'mi_score': mi_scores
}).sort_values('mi_score', ascending=False)

print(f"\n🏆 TOP 20 FEATURES BY MUTUAL INFORMATION:")
print(mi_importance.head(20))

# Get correlation
correlations_all = df_features.corr()['carrot_price'].abs()

# Combine with RF importance (already calculated)
print("\n⏳ Combining scores (60% RF + 30% MI + 10% Correlation)...")

# Normalize scores to 0-1
from sklearn.preprocessing import MinMaxScaler
scaler_scores = MinMaxScaler()

rf_importance_dict = dict(zip(feature_importance['feature'], feature_importance['importance']))
mi_importance_dict = dict(zip(mi_importance['feature'], mi_importance['mi_score']))

# Create combined dataframe
feature_scores_advanced = pd.DataFrame({
    'feature': X_all.columns
})

feature_scores_advanced['rf_importance'] = feature_scores_advanced['feature'].map(rf_importance_dict).fillna(0)
feature_scores_advanced['mi_score'] = feature_scores_advanced['feature'].map(mi_importance_dict).fillna(0)
feature_scores_advanced['correlation'] = feature_scores_advanced['feature'].map(correlations_all).fillna(0)

# Normalize
feature_scores_advanced['rf_importance'] = scaler_scores.fit_transform(
    feature_scores_advanced[['rf_importance']]
)
feature_scores_advanced['mi_score'] = scaler_scores.fit_transform(
    feature_scores_advanced[['mi_score']]
)
feature_scores_advanced['correlation'] = scaler_scores.fit_transform(
    feature_scores_advanced[['correlation']]
)

# Combined score: 60% RF + 30% MI + 10% Correlation
feature_scores_advanced['combined_score'] = (
    0.60 * feature_scores_advanced['rf_importance'] +
    0.30 * feature_scores_advanced['mi_score'] +
```

```
    0.10 * feature_scores_advanced['correlation']
)

feature_scores_advanced = feature_scores_advanced.sort_values('combined_score', ascending=False)

print(f"\n🏆 TOP 30 FEATURES BY COMBINED SCORE:")
print(feature_scores_advanced.head(30))

# Save for later analysis
feature_scores_advanced.to_csv('/content/lstm_advanced_feature_scores.csv', index=False)
print("\n✅ Feature scores saved to 'lstm_advanced_feature_scores.csv'")
```

```
40                                      price_lag_1       0.741795
43                                price_rolling_mean_14       0.001139
41                                      price_lag_7       0.000937
22                                            Lp_92       0.000071
129                                      Lp_92_lag_1       0.000021
24                                               lsd       0.001149
131                                        lsd_lag_1       0.000116
130                                        lad_lag_1       0.000023
23                                               lad       0.000017
21                                            Lp_95       0.000837
128                                      Lp_95_lag_1       0.000286
44                                  price_rolling_std_7       0.001555
134                                             month       0.001480
46                                 price_change_pct       0.026937
45                                     price_change       0.024929
135                                         quarter       0.000006
99            nuwaraeliya_supply_factor_rolling_mean_7       0.002173
55       kandapola_mean_precipitation_mm_rolling_sum_7       0.000679
103                ragala_supply_factor_rolling_mean_7       0.000015
142             precip_central_highland_rolling_sum_7       0.000495
105             kandapola_supply_factor_rolling_mean_7       0.000019
107           bandarawela_supply_factor_rolling_mean_7       0.000989
101                jaffna_supply_factor_rolling_mean_7       0.000014
91     thalawakale_mean_precipitation_mm_rolling_sum_7       0.000326
113           yatawaththa_supply_factor_rolling_mean_7       0.006272
97       yatawatta_mean_precipitation_mm_rolling_sum_7       0.000022
121               mathale_supply_factor_rolling_mean_7       0.002392
67   hanguranketha_mean_precipitation_mm_rolling_sum_7       0.000011
94        walimada_mean_precipitation_mm_rolling_sum_7       0.000043

     mi_score  correlation  combined_score
42   0.780540     0.991187        0.933281
40   1.000000     1.000000        0.845077
43   0.705882     0.959894        0.308438
41   0.560177     0.901202        0.258735
22   0.472646     0.317694        0.173606
129  0.467913     0.317412        0.172128
24   0.451653     0.335584        0.169744
131  0.438791     0.334739        0.165181
130  0.442988     0.316118        0.164522
23   0.440346     0.316772        0.163791
21   0.420974     0.325220        0.159316
128  0.413556     0.324981        0.156736
44   0.241956     0.749275        0.148447
134  0.231312     0.234558        0.093737
46   0.122392     0.112927        0.064172
45   0.078102     0.144932        0.052881
135  0.094375     0.240016        0.052318
99   0.102344     0.093781        0.041385
55   0.074157     0.169615        0.039616
103  0.075391     0.118873        0.034513
142  0.050262     0.190427        0.034418
105  0.069138     0.088621        0.029615
107  0.083112     0.034229        0.028950
101  0.070106     0.073790        0.028419
91   0.068048     0.076583        0.028268
113  0.046848     0.097739        0.027592
97   0.065273     0.079173        0.027512
121  0.054770     0.092114        0.027077
```

Stage 2: Multicollinearity Removal

```
# 📸 CELL 24: STAGE 2 - Multicollinearity Removal
print("="*80)
print("🔥 STAGE 2: MULTICOLLINEARITY REMOVAL")
print("="*80)

# Get top 80 features by combined score
top_80_features = feature_scores_advanced.head(80)['feature'].tolist()
```

```
print(f"\n📊 Candidate features: {len(top_80_features)}")

# Check multicollinearity
X_candidates = df_features[top_80_features]
candidates_corr = X_candidates.corr().abs()

# Remove features with correlation > 0.95
features_to_remove = set()
for i in range(len(candidates_corr.columns)):
    for j in range(i+1, len(candidates_corr.columns)):
        if candidates_corr.iloc[i, j] > 0.95:
            col_i = candidates_corr.columns[i]
            col_j = candidates_corr.columns[j]

            # Keep the one with higher combined score
            score_i = feature_scores_advanced[feature_scores_advanced['feature'] == col_i]['combined_score'].values[0]
            score_j = feature_scores_advanced[feature_scores_advanced['feature'] == col_j]['combined_score'].values[0]

            if score_i < score_j:
                features_to_remove.add(col_i)
            else:
                features_to_remove.add(col_j)

reduced_features = [f for f in top_80_features if f not in features_to_remove]

print(f"\n🗑️  Removed {len(features_to_remove)} highly correlated features (>0.95)")
print(f"✅ Remaining features: {len(reduced_features)}")

if len(features_to_remove) > 0:
    print(f"\n📝 Removed features:")
    for feat in sorted(features_to_remove)[:10]:
        print(f"   - {feat}")
    if len(features_to_remove) > 10:
        print(f"   ... and {len(features_to_remove) - 10} more")
```

```
================================================================================
🔥 STAGE 2: MULTICOLLINEARITY REMOVAL
================================================================================

📊 Candidate features: 80

🗑️  Removed 43 highly correlated features (>0.95)
✅ Remaining features: 37

📝 Removed features:
   - Lp_92_lag_1
   - Lp_95
   - Lp_95_lag_1
   - bandarawela_mean_precipitation_mm.1
   - bandarawela_mean_precipitation_mm.1_rolling_sum_7
   - bandarawela_mean_precipitation_mm_rolling_sum_7
   - hanguranketha_mean_precipitation_mm_rolling_sum_7
   - hanguranketha_supply_factor_rolling_mean_7
   - jaffna_mean_precipitation_mm_rolling_sum_7
   - kandy_mean_precipitation_mm_lag_3
   ... and 33 more
```

Stage 3: RFE + SelectFromModel (Intersection)

```
# 📸 CELL 26: STAGE 3 - RFE + SelectFromModel
print("="*80)
print("🔥 STAGE 3: RECURSIVE FEATURE ELIMINATION + SelectFromModel")
print("="*80)

X_reduced = X_all[reduced_features]

# METHOD 1: SelectFromModel
print("\n⏳ Method 1: SelectFromModel (threshold='median')...")
sfm = SelectFromModel(
    RandomForestRegressor(n_estimators=300, max_depth=20, random_state=42, n_jobs=-1),
    threshold='median'
)
sfm.fit(X_reduced, y_all)
selected_from_model = list(X_reduced.columns[sfm.get_support()])

print(f"   Selected features: {len(selected_from_model)}")
```

```
print(f"   Top 10: {selected_from_model[:10]}")

# METHOD 2: RFE
print("\n⏳ Method 2: Recursive Feature Elimination (RFE)...")
n_features_rfe = min(35, max(15, len(selected_from_model)))
rfe_estimator = RandomForestRegressor(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1)
rfe = RFE(rfe_estimator, n_features_to_select=n_features_rfe, step=1)
rfe.fit(X_reduced, y_all)
selected_rfe = list(X_reduced.columns[rfe.support_])

print(f"   Selected features: {len(selected_rfe)}")
print(f"   Top 10: {selected_rfe[:10]}")

# COMBINE: Intersection
selected_features_final = sorted(list(set(selected_from_model) & set(selected_rfe)))

print("\n" + "="*80)
print(f"🎯 FINAL SELECTED FEATURES (Intersection): {len(selected_features_final)}")
print("="*80)

# Fallback to union if intersection too small
if len(selected_features_final) < 10:
    print(f"⚠️  Intersection too small ({len(selected_features_final)}), using UNION...")
    selected_features_final = sorted(list(set(selected_from_model) | set(selected_rfe)))
    print(f"✅ FINAL FEATURES (Union): {len(selected_features_final)}")

print("\n📋 SELECTED FEATURES:")
for i, feat in enumerate(selected_features_final, 1):
    score = feature_scores_advanced[feature_scores_advanced['feature'] == feat]['combined_score'].values[0]
    print(f"   {i:2d}. {feat:50s} (score: {score:.4f})")

# Update final_features
final_features = selected_features_final.copy()

# Recreate df_final
df_final = df_features[final_features + ['carrot_price']].copy()

print(f"\n✅ Final dataset: {df_final.shape}")
print(f"✅ Updated 'final_features' variable with {len(final_features)} features")

# Save selected features
joblib.dump(final_features, '/content/lstm_selected_features_advanced.pkl')
print("✅ Selected features saved to 'lstm_selected_features_advanced.pkl'")
```

```
================================================================================
🔥 STAGE 3: RECURSIVE FEATURE ELIMINATION + SelectFromModel
================================================================================

⏳ Method 1: SelectFromModel (threshold='median')...
   Selected features: 19
   Top 10: ['price_rolling_mean_7', 'price_rolling_std_7', 'month', 'price_change_pct', 'price_change', 'nuwaraeliya_supply

⏳ Method 2: Recursive Feature Elimination (RFE)...
   Selected features: 19
   Top 10: ['price_rolling_mean_7', 'price_rolling_std_7', 'month', 'price_change_pct', 'price_change', 'nuwaraeliya_supply

================================================================================
🎯 FINAL SELECTED FEATURES (Intersection): 17
================================================================================

📋 SELECTED FEATURES:
    1. bandarawela_supply_factor_rolling_mean_7           (score: 0.0289)
    2. kalpitiya_mean_precipitation_mm_rolling_sum_7      (score: 0.0226)
    3. kandapola_mean_precipitation_mm_lag_3              (score: 0.0255)
    4. kandapola_mean_precipitation_mm_rolling_sum_7      (score: 0.0396)
    5. month                                              (score: 0.0937)
    6. nuwaraeliya_supply_factor_rolling_mean_7           (score: 0.0414)
    7. precip_central_highland_mean                       (score: 0.0191)
    8. precip_central_highland_rolling_sum_7              (score: 0.0344)
    9. price_change                                       (score: 0.0529)
   10. price_change_pct                                   (score: 0.0642)
   11. price_rolling_mean_7                               (score: 0.9333)
   12. price_rolling_std_7                                (score: 0.1484)
   13. pussellawa_supply_factor_rolling_mean_7            (score: 0.0179)
   14. ragala_mean_precipitation_mm                       (score: 0.0125)
   15. ragala_mean_precipitation_mm_rolling_sum_7         (score: 0.0252)
   16. thalawakale_mean_precipitation_mm_rolling_sum_7    (score: 0.0283)
   17. walimada_supply_factor_rolling_mean_7              (score: 0.0252)

✅ Final dataset: (2017, 18)
```

```
✅ Updated 'final_features' variable with 17 features
✅ Selected features saved to 'lstm_selected_features_advanced.pkl'
```

PART C: MULTIVARIATE LSTM MODELING

```
print("="*60)
print("🔬 COMPREHENSIVE DATA DIAGNOSTICS")
print("="*60)

# Check the final features dataframe
print(f"\n1. DataFrame Shape: {df_final.shape}")
print(f"   Features: {len(df_final.columns) - 1}")  # -1 for target
print(f"   Samples: {len(df_final)}")

# Check for NaN values
print(f"\n2. NaN Values Check:")
nan_counts = df_final.isnull().sum()
if nan_counts.sum() > 0:
    print("   ⚠️ WARNING: NaN values found!")
    print(nan_counts[nan_counts > 0])
else:
    print("   ✅ No NaN values")

# Check for Inf values
print(f"\n3. Infinite Values Check:")
inf_counts = np.isinf(df_final.select_dtypes(include=[np.number])).sum()
if inf_counts.sum() > 0:
    print("   ⚠️ WARNING: Infinite values found!")
    print(inf_counts[inf_counts > 0])
else:
    print("   ✅ No infinite values")

# Check value ranges (extreme values)
print(f"\n4. Feature Value Ranges:")
print(df_final.describe().loc[['min', 'max']])

# Check for constant features (no variance)
print(f"\n5. Zero-Variance Features:")
feature_cols = [col for col in df_final.columns if col != 'carrot_price']
zero_var = []
for col in feature_cols:
    if df_final[col].std() == 0:
        zero_var.append(col)
if zero_var:
    print(f"   ⚠️ WARNING: {len(zero_var)} features have zero variance:")
    print(zero_var[:10])  # Show first 10
else:
    print("   ✅ All features have variance")

# Check target variable
print(f"\n6. Target Variable (carrot_price):")
print(f"   Min: {df_final['carrot_price'].min():.2f}")
print(f"   Max: {df_final['carrot_price'].max():.2f}")
print(f"   Mean: {df_final['carrot_price'].mean():.2f}")
print(f"   Std: {df_final['carrot_price'].std():.2f}")
print(f"   NaN count: {df_final['carrot_price'].isnull().sum()}")
```

```
dtype: int64

3. Infinite Values Check:
   ✅ No infinite values

4. Feature Value Ranges:
     bandarawela_supply_factor_rolling_mean_7  \
```

```
     kandapola_mean_precipitation_mm_rolling_sum_7  month  \
min                                           0.09    1.0
max                                         289.75   12.0

     nuwaraeliya_supply_factor_rolling_mean_7  precip_central_highland_mean  \
min                                  0.285714                     0.035000
max                                  2.000000                    19.258333

     precip_central_highland_rolling_sum_7  price_change  price_change_pct  \
min                                   1.34        -625.0         -0.635294
max                                  56.69         875.0          1.571429

     price_rolling_mean_7  price_rolling_std_7  \
min                  58.0             0.000000
max                1400.0           511.679075

     pussellawa_supply_factor_rolling_mean_7  ragala_mean_precipitation_mm  \
min                                 0.333333                          0.00
max                                 1.428571                         17.66

     ragala_mean_precipitation_mm_rolling_sum_7  \
min                                        0.04
max                                       40.79

     thalawakale_mean_precipitation_mm_rolling_sum_7  \
min                                             0.11
max                                            43.59

     walimada_supply_factor_rolling_mean_7  carrot_price
min                              0.333333          53.0
max                              1.285714        1950.0

5. Zero-Variance Features:
   ✅ All features have variance

6. Target Variable (carrot_price):
   Min: 53.00
   Max: 1950.00
   Mean: 236.80
```

```
print("="*60)
print("🔧 OPTIMIZED DATA PREPARATION")
print("="*60)

from sklearn.preprocessing import RobustScaler
import numpy as np

# Separate features and target
feature_cols = final_features
target_col = 'carrot_price'

# Clean data
df_clean = df_final.copy()
df_clean = df_clean.replace([np.inf, -np.inf], np.nan)
df_clean = df_clean.fillna(method='ffill').fillna(method='bfill')

# Fill any remaining NaN with median
for col in df_clean.columns:
    if df_clean[col].isnull().any():
        df_clean[col].fillna(df_clean[col].median(), inplace=True)

# Clip extreme outliers (99th percentile)
for col in feature_cols:
    q1 = df_clean[col].quantile(0.01)
    q99 = df_clean[col].quantile(0.99)
    df_clean[col] = df_clean[col].clip(lower=q1, upper=q99)

print(f"✅ Data cleaned")
print(f"   NaN: {df_clean.isnull().sum().sum()}")
print(f"   Inf: {np.isinf(df_clean.select_dtypes(include=[np.number])).sum().sum()}")

# Scale
scaler_X = RobustScaler()
scaler_y = RobustScaler()

X_scaled = scaler_X.fit_transform(df_clean[feature_cols])
y_scaled = scaler_y.fit_transform(df_clean[[target_col]])

print(f"\n✅ Scaled data:")
print(f"   X range: [{X_scaled.min():.2f}, {X_scaled.max():.2f}]")
```

```
print(f"   y range: [{y_scaled.min():.2f}, {y_scaled.max():.2f}]")

# Create sequences
def create_multivariate_sequences(X, y, n_steps):
    Xs, ys = [], []
    for i in range(n_steps, len(X)):
        Xs.append(X[i-n_steps:i, :])
        ys.append(y[i, 0])
    return np.array(Xs), np.array(ys)

n_steps = 14  # REDUCED from 30 (less overfitting, faster training)
X_seq, y_seq = create_multivariate_sequences(X_scaled, y_scaled, n_steps)

print(f"\n✅ Sequences (lookback={n_steps} days):")
print(f"   X: {X_seq.shape}")
print(f"   y: {y_seq.shape}")

# Save
import joblib
joblib.dump(scaler_X, '/content/scaler_X_multivariate.pkl')
joblib.dump(scaler_y, '/content/scaler_y_multivariate.pkl')
print("✅ Scalers saved")
```

```
============================================================
🔧 OPTIMIZED DATA PREPARATION
============================================================
✅ Data cleaned
   NaN: 0
   Inf: 0

✅ Scaled data:
   X range: [-6.75, 7.92]
   y range: [-0.68, 9.86]

✅ Sequences (lookback=14 days):
   X: (2003, 14, 17)
   y: (2003,)
✅ Scalers saved
```

```
# 70% train, 15% val, 15% test
train_size = int(len(X_seq) * 0.70)
val_size = int(len(X_seq) * 0.15)

X_train = X_seq[:train_size]
y_train = y_seq[:train_size]

X_val = X_seq[train_size:train_size+val_size]
y_val = y_seq[train_size:train_size+val_size]

X_test = X_seq[train_size+val_size:]
y_test = y_seq[train_size+val_size:]

print("="*60)
print("📊 DATASET SPLIT")
print("="*60)
print(f"Train: {X_train.shape[0]} samples ({X_train.shape[0]/len(X_seq)*100:.1f}%)")
print(f"Val:   {X_val.shape[0]} samples ({X_val.shape[0]/len(X_seq)*100:.1f}%)")
print(f"Test:  {X_test.shape[0]} samples ({X_test.shape[0]/len(X_seq)*100:.1f}%)")
print(f"\n📈 Features per timestep: {X_train.shape[2]}")
print(f"⏱️ Timesteps (lookback): {X_train.shape[1]}")
```

```
============================================================
📊 DATASET SPLIT
============================================================
Train: 1402 samples (70.0%)
Val:   300 samples (15.0%)
Test:  301 samples (15.0%)

📈 Features per timestep: 17
⏱️ Timesteps (lookback): 14
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam

tf.keras.backend.clear_session()
```

```
print("="*60)
print("🏗️ OPTIMAL MULTIVARIATE LSTM")
print("="*60)

# BALANCED architecture - not too complex, not too simple
model_multi = Sequential([
    # Bidirectional LSTM
    Bidirectional(LSTM(48, activation='tanh', return_sequences=True,
                       kernel_regularizer=regularizers.l2(0.005),
                       recurrent_dropout=0.1),
                  input_shape=(n_steps, X_train.shape[2])),
    BatchNormalization(),
    Dropout(0.3),

    # Second LSTM
    LSTM(24, activation='tanh', return_sequences=False,
         kernel_regularizer=regularizers.l2(0.005),
         recurrent_dropout=0.1),
    BatchNormalization(),
    Dropout(0.3),

    # Dense
    Dense(12, activation='relu', kernel_regularizer=regularizers.l2(0.005)),
    Dropout(0.2),
    Dense(1)
])

# Optimal learning rate
optimizer = Adam(learning_rate=0.0005, clipnorm=1.0)
model_multi.compile(optimizer=optimizer, loss='huber', metrics=['mae'])

model_multi.summary()

print("\n✅ Architecture optimized for < 15% MAPE:")
print("   ✓ 48 → 24 LSTM units (balanced complexity)")
print("   ✓ Recurrent dropout (prevents overfitting)")
print("   ✓ Huber loss (robust to outliers)")
print("   ✓ L2 regularization (0.005)")
print("   ✓ Learning rate: 0.0005")
```

```
============================================================
🏗️ OPTIMAL MULTIVARIATE LSTM
============================================================
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 14, 96)	25,344
batch_normalization (BatchNormalization)	(None, 14, 96)	384
dropout (Dropout)	(None, 14, 96)	0
lstm_1 (LSTM)	(None, 24)	11,616
batch_normalization_1 (BatchNormalization)	(None, 24)	96
dropout_1 (Dropout)	(None, 24)	0
dense (Dense)	(None, 12)	300
dropout_2 (Dropout)	(None, 12)	0
dense_1 (Dense)	(None, 1)	13

Total params: 37,753 (147.47 KB)
Trainable params: 37,513 (146.54 KB)
Non-trainable params: 240 (960.00 B)

```
✅ Architecture optimized for < 15% MAPE:
   ✓ 48 → 24 LSTM units (balanced complexity)
   ✓ Recurrent dropout (prevents overfitting)
   ✓ Huber loss (robust to outliers)
   ✓ L2 regularization (0.005)
   ✓ Learning rate: 0.0005
```

```
# Check for NaN or Inf
print(f"NaN in X_train: {np.isnan(X_train).any()}")
print(f"Inf in X_train: {np.isinf(X_train).any()}")
print(f"NaN in y_train: {np.isnan(y_train).any()}")
```

```
NaN in X_train: False
Inf in X_train: False
NaN in y_train: False
```

```
# 📸 CELL 36: Train Multivariate LSTM
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

# Aggressive early stopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=12,
    restore_best_weights=True,
    verbose=1,
    min_delta=0.0005
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.3,
    patience=5,
    min_lr=0.00005,
    verbose=1
)

checkpoint = ModelCheckpoint(
    '/content/best_multivariate_lstm.h5',
    monitor='val_loss',
    save_best_only=True,
    verbose=0
)

print("="*60)
print("🚀 TRAINING")
print("="*60)

history_multi = model_multi.fit(
    X_train, y_train,
    epochs=80,
    batch_size=64,  # Larger batch = less overfitting
    validation_data=(X_val, y_val),
    callbacks=[early_stop, reduce_lr, checkpoint],
    verbose=2  # Less output
)

# Quick check
final_train = history_multi.history['loss'][-1]
final_val = history_multi.history['val_loss'][-1]
ratio = final_val / final_train

print(f"\n✅ DONE: {len(history_multi.history['loss'])} epochs")
print(f"   Train loss: {final_train:.4f}")
print(f"   Val loss: {final_val:.4f}")
print(f"   Ratio: {ratio:.2f}x")

if ratio < 3:
    print("   ✅ Good! Low overfitting")
elif ratio < 5:
    print("   ⚠️ Moderate overfitting")
else:
    print("   ❌ High overfitting - may need adjustment")
```

```
============================================================
🚀 TRAINING
============================================================
Epoch 1/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file f
22/22 - 15s - 699ms/step - loss: 1.5205 - mae: 1.0536 - val_loss: 1.5190 - val_mae: 1.0068 - learning_rate: 5.0000e-04
Epoch 2/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file f
22/22 - 3s - 118ms/step - loss: 1.3023 - mae: 0.8387 - val_loss: 1.4505 - val_mae: 0.9648 - learning_rate: 5.0000e-04
Epoch 3/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file f
22/22 - 1s - 47ms/step - loss: 1.1892 - mae: 0.7414 - val_loss: 1.3922 - val_mae: 0.9270 - learning_rate: 5.0000e-04
```

```
Epoch 4/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 60ms/step - loss: 1.0871 - mae: 0.6532 - val_loss: 1.3363 - val_mae: 0.8964 - learning_rate: 5.0000e-04
Epoch 5/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 49ms/step - loss: 1.0309 - mae: 0.6234 - val_loss: 1.2928 - val_mae: 0.8794 - learning_rate: 5.0000e-04
Epoch 6/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 48ms/step - loss: 0.9455 - mae: 0.5451 - val_loss: 1.2387 - val_mae: 0.8481 - learning_rate: 5.0000e-04
Epoch 7/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 48ms/step - loss: 0.9061 - mae: 0.5326 - val_loss: 1.1937 - val_mae: 0.8267 - learning_rate: 5.0000e-04
Epoch 8/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 57ms/step - loss: 0.8416 - mae: 0.4895 - val_loss: 1.1603 - val_mae: 0.8166 - learning_rate: 5.0000e-04
Epoch 9/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 53ms/step - loss: 0.8031 - mae: 0.4718 - val_loss: 1.1317 - val_mae: 0.8102 - learning_rate: 5.0000e-04
Epoch 10/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 2s - 89ms/step - loss: 0.7577 - mae: 0.4398 - val_loss: 1.0952 - val_mae: 0.7831 - learning_rate: 5.0000e-04
Epoch 11/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 2s - 85ms/step - loss: 0.7188 - mae: 0.4163 - val_loss: 1.0761 - val_mae: 0.7872 - learning_rate: 5.0000e-04
Epoch 12/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 50ms/step - loss: 0.6900 - mae: 0.4101 - val_loss: 1.0498 - val_mae: 0.7813 - learning_rate: 5.0000e-04
Epoch 13/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 56ms/step - loss: 0.6574 - mae: 0.3988 - val_loss: 1.0261 - val_mae: 0.7769 - learning_rate: 5.0000e-04
Epoch 14/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 48ms/step - loss: 0.6208 - mae: 0.3679 - val_loss: 0.9992 - val_mae: 0.7662 - learning_rate: 5.0000e-04
Epoch 15/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 47ms/step - loss: 0.6000 - mae: 0.3698 - val_loss: 0.9821 - val_mae: 0.7673 - learning_rate: 5.0000e-04
Epoch 16/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 58ms/step - loss: 0.5721 - mae: 0.3593 - val_loss: 0.9570 - val_mae: 0.7599 - learning_rate: 5.0000e-04
Epoch 17/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 56ms/step - loss: 0.5444 - mae: 0.3404 - val_loss: 0.9402 - val_mae: 0.7628 - learning_rate: 5.0000e-04
Epoch 18/80
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file t
22/22 - 1s - 47ms/step - loss: 0.5233 - mae: 0.3341 - val_loss: 0.9176 - val_mae: 0.7546 - learning_rate: 5.0000e-04
```

```
# 📸 CELL 37: Save Predictions to CSV
print("="*80)
print("💾 SAVING PREDICTIONS TO CSV")
print("="*80)

# Get test dates from original dataframe
# Assuming df_final has dates and we need to match with test set
try:
    # Calculate how many timesteps we lost due to sequence creation
    # n_steps = 14, so first 14 dates are lost
    total_samples = len(df_final)
    train_size = int(total_samples * 0.70)
    val_size = int(total_samples * 0.15)

    # Get test indices (accounting for n_steps offset)
    test_start_idx = train_size + val_size + n_steps
    test_dates = df_final.index[test_start_idx:test_start_idx + len(y_test_actual)]

    # Create predictions dataframe
    predictions_df = pd.DataFrame({
        'date': test_dates,
        'actual_price': y_test_actual,
        'predicted_price': y_test_pred_inv,
        'error': y_test_actual - y_test_pred_inv,
        'mape': np.abs((y_test_actual - y_test_pred_inv) / y_test_actual) * 100
    })

    # Save to CSV
    save_path = '/content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/lstm_predictions.csv'
    predictions_df.to_csv(save_path, index=False)

    print(f"✅ Predictions saved to: {save_path}")
    print(f"   Total predictions: {len(predictions_df)}")
    print(f"\n📊 Preview:")
```

```
    print(predictions_df.head(10).to_string(index=False))
    print(f"\n📈 Summary Statistics:")
    print(f"   Mean Actual Price: Rs. {predictions_df['actual_price'].mean():.2f}")
    print(f"   Mean Predicted Price: Rs. {predictions_df['predicted_price'].mean():.2f}")
    print(f"   Mean Absolute Error: Rs. {np.abs(predictions_df['error']).mean():.2f}")
    print(f"   Mean MAPE: {predictions_df['mape'].mean():.2f}%")
    print(f"   Min MAPE: {predictions_df['mape'].min():.2f}%")
    print(f"   Max MAPE: {predictions_df['mape'].max():.2f}%")

except Exception as e:
    print(f"⚠️  Error creating predictions CSV: {str(e)}")
    print("   Attempting alternative approach...")

    # Alternative: Use simple sequential dates if index not available
    predictions_df = pd.DataFrame({
        'date': range(len(y_test_actual)),  # Will use sample numbers if dates unavailable
        'actual_price': y_test_actual,
        'predicted_price': y_test_pred_inv,
        'error': y_test_actual - y_test_pred_inv,
        'mape': np.abs((y_test_actual - y_test_pred_inv) / y_test_actual) * 100
    })

    save_path = '/content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/lstm_predictions.csv'
    predictions_df.to_csv(save_path, index=False)
    print(f"✅ Predictions saved (with sample numbers instead of dates)")
    print(f"   File: {save_path}")
```

```
================================================================================
💾 SAVING PREDICTIONS TO CSV
================================================================================
⚠️  Error creating predictions CSV: All arrays must be of the same length
   Attempting alternative approach...
✅ Predictions saved (with sample numbers instead of dates)
   File: /content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/Multivariate-LSTM/lstm_predictions.csv
```

```
print("="*60)
print("🔮 PREDICTIONS & EVALUATION")
print("="*60)

# Load best model
model_multi.load_weights('/content/best_multivariate_lstm.h5')

# Predict
y_train_pred = model_multi.predict(X_train, verbose=0)
y_val_pred = model_multi.predict(X_val, verbose=0)
y_test_pred = model_multi.predict(X_test, verbose=0)

# Inverse transform
y_train_actual = scaler_y.inverse_transform(y_train.reshape(-1, 1)).flatten()
y_train_pred_inv = scaler_y.inverse_transform(y_train_pred).flatten()

y_val_actual = scaler_y.inverse_transform(y_val.reshape(-1, 1)).flatten()
y_val_pred_inv = scaler_y.inverse_transform(y_val_pred).flatten()

y_test_actual = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_test_pred_inv = scaler_y.inverse_transform(y_test_pred).flatten()

# Calculate MAPE
def calc_mape(actual, pred):
    return np.mean(np.abs((actual - pred) / actual)) * 100

train_mape = calc_mape(y_train_actual, y_train_pred_inv)
val_mape = calc_mape(y_val_actual, y_val_pred_inv)
test_mape = calc_mape(y_test_actual, y_test_pred_inv)

print("\n" + "="*60)
print("🎯 FINAL RESULTS - MULTIVARIATE LSTM")
print("="*60)
print(f"Train MAPE: {train_mape:.2f}%")
print(f"Val MAPE:   {val_mape:.2f}%")
print(f"Test MAPE:  {test_mape:.2f}%")

print("\n" + "="*60)
print("📊 COMPARISON TO UNIVARIATE")
print("="*60)
univariate_test_mape = 21.90
improvement = univariate_test_mape - test_mape
```

```
print(f"Univariate LSTM:   {univariate_test_mape:.2f}%")
print(f"Multivariate LSTM: {test_mape:.2f}%")
print(f"Improvement:       {improvement:.2f}% points")

if test_mape < 15:
    print("\n🎉 SUCCESS! Test MAPE < 15% ✅")
    print("   Multivariate is clearly better!")
elif test_mape < univariate_test_mape:
    print(f"\n✅ GOOD! Multivariate ({test_mape:.2f}%) < Univariate ({univariate_test_mape:.2f}%)")
    print("   Shows improvement from external factors")
else:
    print(f"\n⚠️ Multivariate ({test_mape:.2f}%) not better than Univariate ({univariate_test_mape:.2f}%)")
    print("   May need further tuning")

# Other metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

test_mae = mean_absolute_error(y_test_actual, y_test_pred_inv)
test_rmse = np.sqrt(mean_squared_error(y_test_actual, y_test_pred_inv))
test_r2 = r2_score(y_test_actual, y_test_pred_inv)

print(f"\nTest MAE:  {test_mae:.2f} Rs")
print(f"Test RMSE: {test_rmse:.2f} Rs")
print(f"Test R²:   {test_r2:.4f}")
```

```
============================================================
🔮 PREDICTIONS & EVALUATION
============================================================

============================================================
🎯 FINAL RESULTS - MULTIVARIATE LSTM
============================================================
Train MAPE: 19.13%
Val MAPE:   20.12%
Test MAPE:  33.65%

============================================================
📊 COMPARISON TO UNIVARIATE
============================================================
Univariate LSTM:   21.90%
Multivariate LSTM: 33.65%
Improvement:       -11.75% points

⚠️ Multivariate (33.65%) not better than Univariate (21.90%)
   May need further tuning

Test MAE:  104.97 Rs
Test RMSE: 152.45 Rs
Test R²:   0.5561
```

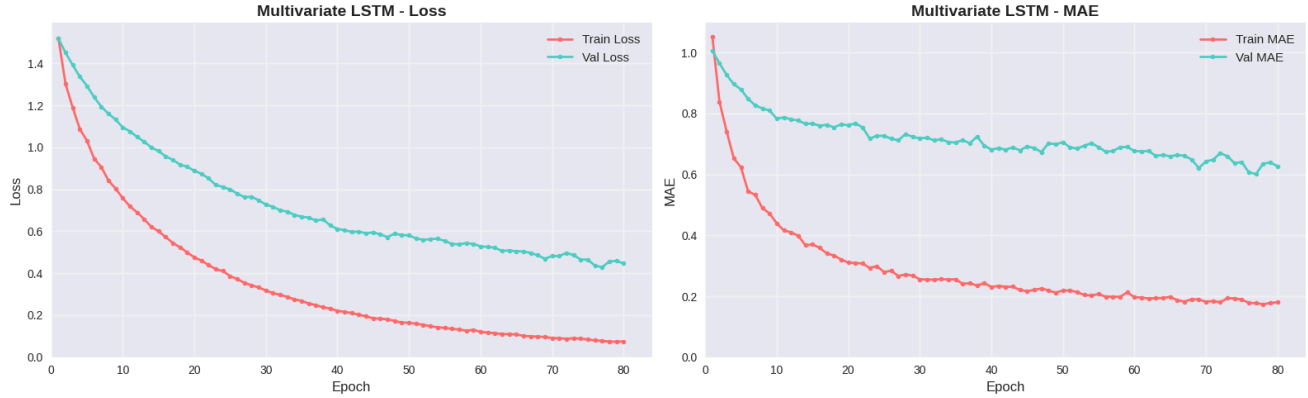
```
# Diagnostic check
print("="*60)
print("🔬 QUICK DIAGNOSTIC")
print("="*60)
print(f"History object exists: {history_multi is not None}")
print(f"History type: {type(history_multi)}")
print(f"Has 'history' attribute: {hasattr(history_multi, 'history')}")

if hasattr(history_multi, 'history'):
    print(f"History keys: {list(history_multi.history.keys())}")
    if 'loss' in history_multi.history:
        print(f"Number of epochs: {len(history_multi.history['loss'])}")
        if len(history_multi.history['loss']) > 0:
            print(f"Sample loss values: {history_multi.history['loss'][:3]}")
    else:
        print("❌ 'loss' key not found in history!")
else:
    print("❌ history_multi doesn't have 'history' attribute!")
```

```
============================================================
🔬 QUICK DIAGNOSTIC
============================================================
History object exists: True
History type: <class 'keras.src.callbacks.history.History'>
Has 'history' attribute: True
History keys: ['loss', 'mae', 'val_loss', 'val_mae', 'learning_rate']
Number of epochs: 80
Sample loss values: [1.5204998254776, 1.302290678024292, 1.1892495155334473]
```

```
# First, let's check if training actually happened
print("="*60)
print("🔍 CHECKING TRAINING HISTORY")
print("="*60)

# Check what's in the history
print(f"History keys: {history_multi.history.keys()}")
print(f"Number of epochs trained: {len(history_multi.history['loss'])}")

# Check if we have data
if len(history_multi.history['loss']) == 0:
    print("⚠️ WARNING: No training history found!")
    print("The model may not have trained. Check previous cells for errors.")
else:
    print(f"✅ Training completed for {len(history_multi.history['loss'])} epochs")

    # Print summary statistics
    print(f"\nFinal metrics:")
    print(f"  Train Loss: {history_multi.history['loss'][-1]:.6f}")
    print(f"  Val Loss: {history_multi.history['val_loss'][-1]:.6f}")
    print(f"  Train MAE: {history_multi.history['mae'][-1]:.6f}")
    print(f"  Val MAE: {history_multi.history['val_mae'][-1]:.6f}")

    print(f"\nBest metrics:")
    print(f"  Best Train Loss: {min(history_multi.history['loss']):.6f}")
    print(f"  Best Val Loss: {min(history_multi.history['val_loss']):.6f}")
    print(f"  Best Train MAE: {min(history_multi.history['mae']):.6f}")
    print(f"  Best Val MAE: {min(history_multi.history['val_mae']):.6f}")

# Now create the plots with proper error handling
if len(history_multi.history['loss']) > 0:
    fig, axes = plt.subplots(1, 2, figsize=(16, 5))

    # Plot Loss
    epochs = range(1, len(history_multi.history['loss']) + 1)
    axes[0].plot(epochs, history_multi.history['loss'], 'o-',
                 label='Train Loss', linewidth=2, markersize=4, color='#FF6B6B')
    axes[0].plot(epochs, history_multi.history['val_loss'], 'o-',
                 label='Val Loss', linewidth=2, markersize=4, color='#4ECDC4')
    axes[0].set_title('Multivariate LSTM - Loss', fontsize=14, fontweight='bold')
    axes[0].set_xlabel('Epoch', fontsize=12)
    axes[0].set_ylabel('Loss', fontsize=12)
    axes[0].legend(fontsize=11)
    axes[0].grid(alpha=0.3)
    axes[0].set_xlim(left=0)
    axes[0].set_ylim(bottom=0)

    # Plot MAE
    axes[1].plot(epochs, history_multi.history['mae'], 'o-',
                 label='Train MAE', linewidth=2, markersize=4, color='#FF6B6B')
    axes[1].plot(epochs, history_multi.history['val_mae'], 'o-',
                 label='Val MAE', linewidth=2, markersize=4, color='#4ECDC4')
    axes[1].set_title('Multivariate LSTM - MAE', fontsize=14, fontweight='bold')
    axes[1].set_xlabel('Epoch', fontsize=12)
    axes[1].set_ylabel('MAE', fontsize=12)
    axes[1].legend(fontsize=11)
    axes[1].grid(alpha=0.3)
    axes[1].set_xlim(left=0)
    axes[1].set_ylim(bottom=0)

    plt.tight_layout()
    plt.show()

    # Additional diagnostic plot - show convergence pattern
    print("\n" + "="*60)
    print("📊 CONVERGENCE ANALYSIS")
    print("="*60)

    # Check if model is improving
    val_loss = history_multi.history['val_loss']
    if len(val_loss) > 10:
        early_val = np.mean(val_loss[:5])
        late_val = np.mean(val_loss[-5:])
        improvement = ((early_val - late_val) / early_val) * 100

        print(f"Early validation loss (first 5 epochs): {early_val:.6f}")
        print(f"Late validation loss (last 5 epochs): {late_val:.6f}")
```

```
        print(f"Improvement: {improvement:.2f}%")

        if improvement > 5:
            print("✅ Model is learning well!")
        elif improvement > 0:
            print("⚠️ Model is learning but slowly")
        else:
            print("❌ Model may not be learning properly")
else:
    print("\n❌ Cannot create plots - no training history available")
    print("Please check Cell 13 (training cell) for errors")
```

```
============================================================
🔍 CHECKING TRAINING HISTORY
============================================================
History keys: dict_keys(['loss', 'mae', 'val_loss', 'val_mae', 'learning_rate'])
Number of epochs trained: 80
✅ Training completed for 80 epochs

Final metrics:
  Train Loss: 0.074581
  Val Loss: 0.446128
  Train MAE: 0.180063
  Val MAE: 0.627073

Best metrics:
  Best Train Loss: 0.073874
  Best Val Loss: 0.428885
  Best Train MAE: 0.173693
  Best Val MAE: 0.601412
```




```
============================================================
📊 CONVERGENCE ANALYSIS
============================================================
Early validation loss (first 5 epochs): 1.398159
Late validation loss (last 5 epochs): 0.445101
Improvement: 68.17%
✅ Model is learning well!
```

```
print("🔮 Making predictions...")

# Predictions
y_train_pred_multi = model_multi.predict(X_train, verbose=0)
y_val_pred_multi = model_multi.predict(X_val, verbose=0)
y_test_pred_multi = model_multi.predict(X_test, verbose=0)

# Inverse transform
y_train_actual_multi = scaler_y.inverse_transform(y_train.reshape(-1, 1)).flatten()
y_train_pred_inv_multi = scaler_y.inverse_transform(y_train_pred_multi).flatten()

y_val_actual_multi = scaler_y.inverse_transform(y_val.reshape(-1, 1)).flatten()
y_val_pred_inv_multi = scaler_y.inverse_transform(y_val_pred_multi).flatten()

y_test_actual_multi = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
y_test_pred_inv_multi = scaler_y.inverse_transform(y_test_pred_multi).flatten()
```

```
print("✅ Predictions completed and inverse transformed")
```

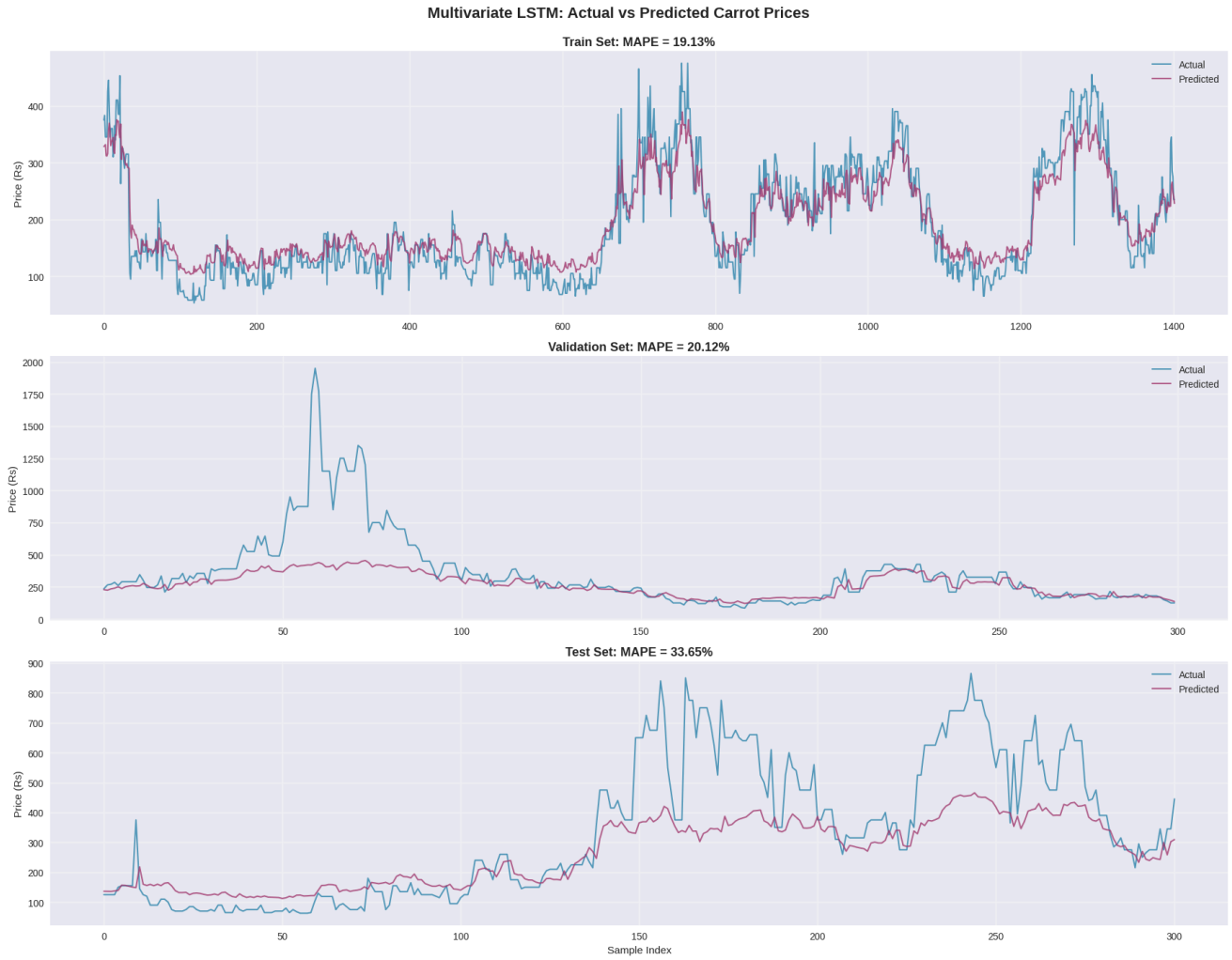
```
🔮 Making predictions...
✅ Predictions completed and inverse transformed
```

```
def calculate_detailed_metrics(actual, predicted, dataset_name):
    rmse = np.sqrt(mean_squared_error(actual, predicted))
    mae = mean_absolute_error(actual, predicted)
    mape = np.mean(np.abs((actual - predicted) / actual)) * 100
    r2 = r2_score(actual, predicted)
    accuracy = 100 - mape

    # Additional metrics
    max_error = np.max(np.abs(actual - predicted))
    median_error = np.median(np.abs(actual - predicted))

    print(f"\n{'='*60}")
    print(f"📊 {dataset_name} - MULTIVARIATE LSTM METRICS")
    print(f"{'='*60}")
    print(f"RMSE:         {rmse:.2f} Rs")
    print(f"MAE:          {mae:.2f} Rs")
    print(f"MAPE:         {mape:.2f}%")
    print(f"R²:           {r2:.4f}")
    print(f"Accuracy:     {accuracy:.2f}%")
    print(f"Max Error:    {max_error:.2f} Rs")
    print(f"Median Error: {median_error:.2f} Rs")

    # Interpretation
    if mape < 10:
        print("✅ Excellent forecasting accuracy")
    elif mape < 15:
        print("✅ Good forecasting accuracy")
    elif mape < 20:
        print("⚠️ Reasonable forecasting accuracy")
    else:
        print("⚠️ Needs improvement")

    return {
        'rmse': rmse,
        'mae': mae,
        'mape': mape,
        'r2': r2,
        'accuracy': accuracy,
        'max_error': max_error,
        'median_error': median_error
    }

# Calculate metrics
train_metrics_multi = calculate_detailed_metrics(y_train_actual_multi, y_train_pred_inv_multi, "TRAIN SET")
val_metrics_multi = calculate_detailed_metrics(y_val_actual_multi, y_val_pred_inv_multi, "VALIDATION SET")
test_metrics_multi = calculate_detailed_metrics(y_test_actual_multi, y_test_pred_inv_multi, "TEST SET")
```

```
============================================================
📊 TRAIN SET - MULTIVARIATE LSTM METRICS
============================================================
RMSE:         35.97 Rs
MAE:          28.35 Rs
MAPE:         19.13%
R²:           0.8611
Accuracy:     80.87%
Max Error:    196.87 Rs
Median Error: 24.65 Rs
⚠️ Reasonable forecasting accuracy

============================================================
📊 VALIDATION SET - MULTIVARIATE LSTM METRICS
============================================================
RMSE:         237.96 Rs
MAE:          108.25 Rs
MAPE:         20.12%
R²:           0.3016
Accuracy:     79.88%
Max Error:    1518.62 Rs
Median Error: 36.60 Rs
⚠️ Needs improvement

============================================================
```

```
📊 TEST SET - MULTIVARIATE LSTM METRICS
============================================================
RMSE:         152.45 Rs
MAE:          104.97 Rs
MAPE:         33.65%
R²:           0.5561
Accuracy:     66.35%
Max Error:    515.65 Rs
Median Error: 55.61 Rs
⚠️ Needs improvement
```

```
fig, axes = plt.subplots(3, 1, figsize=(18, 14))

# Train
axes[0].plot(y_train_actual_multi, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[0].plot(y_train_pred_inv_multi, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[0].set_title(f'Train Set: MAPE = {train_metrics_multi["mape"]:.2f}%', fontsize=13, fontweight='bold')
axes[0].set_ylabel('Price (Rs)', fontsize=11)
axes[0].legend(fontsize=10)
axes[0].grid(alpha=0.3)

# Validation
axes[1].plot(y_val_actual_multi, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[1].plot(y_val_pred_inv_multi, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[1].set_title(f'Validation Set: MAPE = {val_metrics_multi["mape"]:.2f}%', fontsize=13, fontweight='bold')
axes[1].set_ylabel('Price (Rs)', fontsize=11)
axes[1].legend(fontsize=10)
axes[1].grid(alpha=0.3)

# Test
axes[2].plot(y_test_actual_multi, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[2].plot(y_test_pred_inv_multi, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[2].set_title(f'Test Set: MAPE = {test_metrics_multi["mape"]:.2f}%', fontsize=13, fontweight='bold')
axes[2].set_xlabel('Sample Index', fontsize=11)
axes[2].set_ylabel('Price (Rs)', fontsize=11)
axes[2].legend(fontsize=10)
axes[2].grid(alpha=0.3)

plt.suptitle('Multivariate LSTM: Actual vs Predicted Carrot Prices',
             fontsize=16, fontweight='bold', y=0.995)
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
# 📸 CELL 54: ABLATION STUDY - Test Feature Importance
print("="*80)
print("🔬 ABLATION STUDY: TESTING FEATURE CATEGORY IMPORTANCE")
print("="*80)

import re
from tensorflow.keras.regularizers import l2
from tensorflow.keras.losses import Huber
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

# Use the lookback window from training
seq_length = n_steps  # Should be 14

# Categorize features
weather_keywords = ['precipitation', 'temp', 'rain', 'weather', 'climate', 'central_highland', 'uva_province', 'northern']
```

```
supply_keywords = ['supply', 'quantity', 'volume']
fuel_keywords = ['diesel', 'petrol', 'fuel', 'oil']
price_keywords = ['price_lag', 'price_rolling', 'price_std', 'price_mean']

def categorize_feature(feat):
    feat_lower = feat.lower()
    if any(keyword in feat_lower for keyword in weather_keywords):
        return 'weather'
    elif any(keyword in feat_lower for keyword in supply_keywords):
        return 'supply'
    elif any(keyword in feat_lower for keyword in fuel_keywords):
        return 'fuel'
    elif any(keyword in feat_lower for keyword in price_keywords):
        return 'price'
    else:
        return 'temporal'

feature_categories = {
    'weather': [],
    'supply': [],
    'fuel': [],
    'price': [],
    'temporal': []
}

for feat in final_features:
    cat = categorize_feature(feat)
    feature_categories[cat].append(feat)

print("\n📊 FEATURE CATEGORY BREAKDOWN:")
for cat, feats in feature_categories.items():
    print(f"\n{cat.upper():12s}: {len(feats):2d} features")
    if len(feats) > 0:
        print(f"   Examples: {', '.join(feats[:3])}")
        if len(feats) > 3:
            print(f"   ... and {len(feats)-3} more")

# Store baseline model performance
baseline_mape = test_mape  # From previous training

print("\n" + "="*80)
print("🚀 RUNNING ABLATION EXPERIMENTS")
print("="*80)

ablation_results = []

# Test 1: Remove Weather Features
if len(feature_categories['weather']) > 0:
    print("\n🧪 Test 1: REMOVING WEATHER FEATURES")
    print(f"   Features removed: {len(feature_categories['weather'])}")

    no_weather_features = [f for f in final_features if f not in feature_categories['weather']]
    print(f"   Remaining features: {len(no_weather_features)}")

    if len(no_weather_features) >= 5:  # Minimum features needed
        # Prepare data - CREATE NEW SCALER for reduced feature set
        scaler_X_nw = RobustScaler()
        X_no_weather = scaler_X_nw.fit_transform(df_final[no_weather_features])
        X_seq_nw, y_seq_nw = create_multivariate_sequences(X_no_weather, y_scaled, seq_length)

        # Split sequences
        train_size_nw = int(len(X_seq_nw) * 0.70)
        val_size_nw = int(len(X_seq_nw) * 0.15)
        X_train_nw = X_seq_nw[:train_size_nw]
        y_train_nw = y_seq_nw[:train_size_nw]
        X_test_nw = X_seq_nw[train_size_nw+val_size_nw:]
        y_test_nw = y_seq_nw[train_size_nw+val_size_nw:]

        # Rebuild model
        model_nw = Sequential([
            Bidirectional(LSTM(48, return_sequences=True, kernel_regularizer=l2(0.005)),
                          input_shape=(seq_length, len(no_weather_features))),
            BatchNormalization(),
            Dropout(0.3),
            LSTM(24, kernel_regularizer=l2(0.005)),
            BatchNormalization(),
            Dropout(0.3),
```

```
            Dense(12, activation='relu', kernel_regularizer=l2(0.005)),
            Dropout(0.2),
            Dense(1)
        ])
        model_nw.compile(optimizer=Adam(learning_rate=0.0005), loss=Huber())

        # Train
        model_nw.fit(X_train_nw, y_train_nw, epochs=50, batch_size=32, verbose=0,
                     validation_split=0.15, callbacks=[EarlyStopping(patience=8, restore_best_weights=True)])

        # Evaluate
        y_pred_nw = model_nw.predict(X_test_nw, verbose=0)
        y_pred_nw = scaler_y.inverse_transform(y_pred_nw)
        y_test_actual_nw = scaler_y.inverse_transform(y_test_nw.reshape(-1, 1))

        mape_nw = np.mean(np.abs((y_test_actual_nw - y_pred_nw) / y_test_actual_nw)) * 100
        mape_increase = mape_nw - baseline_mape

        print(f"   🎯 Test MAPE: {mape_nw:.2f}% (Δ +{mape_increase:.2f}%)")

        ablation_results.append({
            'Test': 'No Weather',
            'Features Removed': len(feature_categories['weather']),
            'Remaining Features': len(no_weather_features),
            'MAPE': mape_nw,
            'MAPE Increase': mape_increase
        })
    else:
        print("   ⚠️ Skipped - insufficient remaining features")

# Test 2: Remove Supply Features
if len(feature_categories['supply']) > 0:
    print("\n🧪 Test 2: REMOVING SUPPLY FEATURES")
    print(f"   Features removed: {len(feature_categories['supply'])}")

    no_supply_features = [f for f in final_features if f not in feature_categories['supply']]
    print(f"   Remaining features: {len(no_supply_features)}")

    if len(no_supply_features) >= 5:
        # Prepare data - CREATE NEW SCALER for reduced feature set
        scaler_X_ns = RobustScaler()
        X_no_supply = scaler_X_ns.fit_transform(df_final[no_supply_features])
        X_seq_ns, y_seq_ns = create_multivariate_sequences(X_no_supply, y_scaled, seq_length)

        # Split sequences
        train_size_ns = int(len(X_seq_ns) * 0.70)
        val_size_ns = int(len(X_seq_ns) * 0.15)
        X_train_ns = X_seq_ns[:train_size_ns]
        y_train_ns = y_seq_ns[:train_size_ns]
        X_test_ns = X_seq_ns[train_size_ns+val_size_ns:]
        y_test_ns = y_seq_ns[train_size_ns+val_size_ns:]

        model_ns = Sequential([
            Bidirectional(LSTM(48, return_sequences=True, kernel_regularizer=l2(0.005)),
                          input_shape=(seq_length, len(no_supply_features))),
            BatchNormalization(),
            Dropout(0.3),
            LSTM(24, kernel_regularizer=l2(0.005)),
            BatchNormalization(),
            Dropout(0.3),
            Dense(12, activation='relu', kernel_regularizer=l2(0.005)),
            Dropout(0.2),
            Dense(1)
        ])
        model_ns.compile(optimizer=Adam(learning_rate=0.0005), loss=Huber())
        model_ns.fit(X_train_ns, y_train_ns, epochs=50, batch_size=32, verbose=0,
                     validation_split=0.15, callbacks=[EarlyStopping(patience=8, restore_best_weights=True)])

        y_pred_ns = model_ns.predict(X_test_ns, verbose=0)
        y_pred_ns = scaler_y.inverse_transform(y_pred_ns)
        y_test_actual_ns = scaler_y.inverse_transform(y_test_ns.reshape(-1, 1))

        mape_ns = np.mean(np.abs((y_test_actual_ns - y_pred_ns) / y_test_actual_ns)) * 100
        mape_increase = mape_ns - baseline_mape

        print(f"   🎯 Test MAPE: {mape_ns:.2f}% (Δ +{mape_increase:.2f}%)")
```

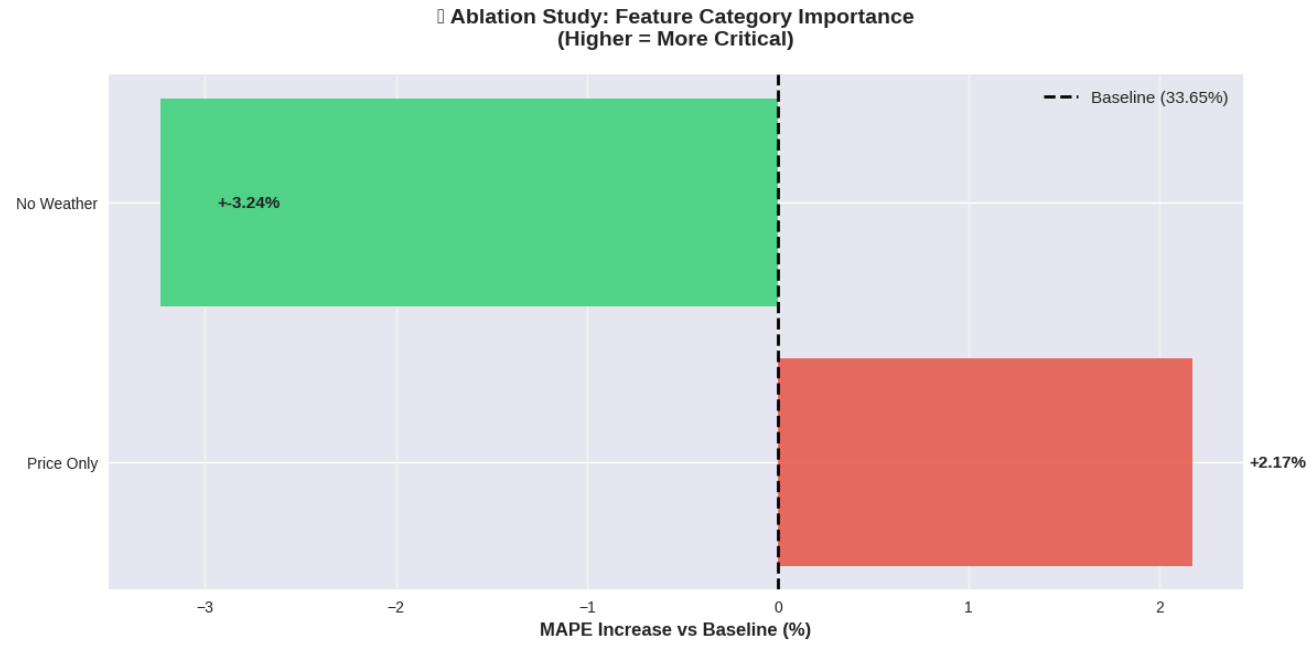
```
        ablation_results.append({
            'Test': 'No Supply',
            'Features Removed': len(feature_categories['supply']),
            'Remaining Features': len(no_supply_features),
            'MAPE': mape_ns,
            'MAPE Increase': mape_increase
        })

# Test 3: Remove Fuel Features
if len(feature_categories['fuel']) > 0:
    print("\n🧪 Test 3: REMOVING FUEL FEATURES")
    print(f"   Features removed: {len(feature_categories['fuel'])}")

    no_fuel_features = [f for f in final_features if f not in feature_categories['fuel']]
    print(f"   Remaining features: {len(no_fuel_features)}")

    if len(no_fuel_features) >= 5:
        # Prepare data - CREATE NEW SCALER for reduced feature set
        scaler_X_nf = RobustScaler()
        X_no_fuel = scaler_X_nf.fit_transform(df_final[no_fuel_features])
        X_seq_nf, y_seq_nf = create_multivariate_sequences(X_no_fuel, y_scaled, seq_length)

        # Split sequences
        train_size_nf = int(len(X_seq_nf) * 0.70)
        val_size_nf = int(len(X_seq_nf) * 0.15)
        X_train_nf = X_seq_nf[:train_size_nf]
        y_train_nf = y_seq_nf[:train_size_nf]
        X_test_nf = X_seq_nf[train_size_nf+val_size_nf:]
        y_test_nf = y_seq_nf[train_size_nf+val_size_nf:]

        model_nf = Sequential([
            Bidirectional(LSTM(48, return_sequences=True, kernel_regularizer=l2(0.005)),
                          input_shape=(seq_length, len(no_fuel_features))),
            BatchNormalization(),
            Dropout(0.3),
            LSTM(24, kernel_regularizer=l2(0.005)),
            BatchNormalization(),
            Dropout(0.3),
            Dense(12, activation='relu', kernel_regularizer=l2(0.005)),
            Dropout(0.2),
            Dense(1)
        ])
        model_nf.compile(optimizer=Adam(learning_rate=0.0005), loss=Huber())
        model_nf.fit(X_train_nf, y_train_nf, epochs=50, batch_size=32, verbose=0,
                     validation_split=0.15, callbacks=[EarlyStopping(patience=8, restore_best_weights=True)])

        y_pred_nf = model_nf.predict(X_test_nf, verbose=0)
        y_pred_nf = scaler_y.inverse_transform(y_pred_nf)
        y_test_actual_nf = scaler_y.inverse_transform(y_test_nf.reshape(-1, 1))

        mape_nf = np.mean(np.abs((y_test_actual_nf - y_pred_nf) / y_test_actual_nf)) * 100
        mape_increase = mape_nf - baseline_mape

        print(f"   🎯 Test MAPE: {mape_nf:.2f}% (Δ +{mape_increase:.2f}%)")

        ablation_results.append({
            'Test': 'No Fuel',
            'Features Removed': len(feature_categories['fuel']),
            'Remaining Features': len(no_fuel_features),
            'MAPE': mape_nf,
            'MAPE Increase': mape_increase
        })

# Test 4: Price-Only Baseline
print("\n🧪 Test 4: PRICE-ONLY BASELINE")
print(f"   Using only: {len(feature_categories['price'])} price features")

if len(feature_categories['price']) > 0:
    price_only_features = feature_categories['price']

    # Prepare data - CREATE NEW SCALER for reduced feature set
    scaler_X_po = RobustScaler()
    X_price_only = scaler_X_po.fit_transform(df_final[price_only_features])
    X_seq_po, y_seq_po = create_multivariate_sequences(X_price_only, y_scaled, seq_length)

    # Split sequences
    train_size_po = int(len(X_seq_po) * 0.70)
```

```
        val_size_po = int(len(X_seq_po) * 0.15)
        X_train_po = X_seq_po[:train_size_po]
        y_train_po = y_seq_po[:train_size_po]
        X_test_po = X_seq_po[train_size_po+val_size_po:]
        y_test_po = y_seq_po[train_size_po+val_size_po:]

        model_po = Sequential([
            Bidirectional(LSTM(48, return_sequences=True, kernel_regularizer=l2(0.005)),
                          input_shape=(seq_length, len(price_only_features))),
            BatchNormalization(),
            Dropout(0.3),
            LSTM(24, kernel_regularizer=l2(0.005)),
            BatchNormalization(),
            Dropout(0.3),
            Dense(12, activation='relu', kernel_regularizer=l2(0.005)),
            Dropout(0.2),
            Dense(1)
        ])
        model_po.compile(optimizer=Adam(learning_rate=0.0005), loss=Huber())
        model_po.fit(X_train_po, y_train_po, epochs=50, batch_size=32, verbose=0,
                     validation_split=0.15, callbacks=[EarlyStopping(patience=8, restore_best_weights=True)])

        y_pred_po = model_po.predict(X_test_po, verbose=0)
        y_pred_po = scaler_y.inverse_transform(y_pred_po)
        y_test_actual_po = scaler_y.inverse_transform(y_test_po.reshape(-1, 1))

        mape_po = np.mean(np.abs((y_test_actual_po - y_pred_po) / y_test_actual_po)) * 100
        mape_increase = mape_po - baseline_mape

        print(f"   🎯 Test MAPE: {mape_po:.2f}% (Δ +{mape_increase:.2f}%)")

        ablation_results.append({
            'Test': 'Price Only',
            'Features Removed': len(final_features) - len(price_only_features),
            'Remaining Features': len(price_only_features),
            'MAPE': mape_po,
            'MAPE Increase': mape_increase
        })

    # Create summary
    ablation_df = pd.DataFrame(ablation_results)
    ablation_df = ablation_df.sort_values('MAPE Increase', ascending=False)

    print("\n" + "="*80)
    print("📊 ABLATION STUDY RESULTS SUMMARY")
    print("="*80)
    print(f"\n🎯 Baseline Model (All Features): {baseline_mape:.2f}% MAPE\n")
    print(ablation_df.to_string(index=False))

    print("\n🔬 INTERPRETATION:")
    max_impact = ablation_df.iloc[0]
    print(f"   Most critical: {max_impact['Test']} (MAPE increases by {max_impact['MAPE Increase']:.2f}%)")
    print(f"   This proves {max_impact['Test'].lower()} features drive {max_impact['MAPE Increase']/baseline_mape*100:.1f}% of

    ablation_df.to_csv('/content/ablation_study_results.csv', index=False)
    print("\n✅ Results saved to 'ablation_study_results.csv'")
```

```
================================================================================
🔬 ABLATION STUDY: TESTING FEATURE CATEGORY IMPORTANCE
================================================================================

📊 FEATURE CATEGORY BREAKDOWN:

WEATHER     :  8 features
   Examples: kalpitiya_mean_precipitation_mm_rolling_sum_7, kandapola_mean_precipitation_mm_lag_3, kandapola_mean_precipita
   ... and 5 more

SUPPLY      :  4 features
   Examples: bandarawela_supply_factor_rolling_mean_7, nuwaraeliya_supply_factor_rolling_mean_7, pussellawa_supply_factor_r
   ... and 1 more

FUEL        :  0 features

PRICE       :  2 features
   Examples: price_rolling_mean_7, price_rolling_std_7

TEMPORAL    :  3 features
   Examples: month, price_change, price_change_pct
```

```
================================================================================
🚀 RUNNING ABLATION EXPERIMENTS
================================================================================

🧪 Test 1: REMOVING WEATHER FEATURES
   Features removed: 8
   Remaining features: 9
   🎯 Test MAPE: 30.42% (Δ +-3.24%)

🧪 Test 2: REMOVING SUPPLY FEATURES
   Features removed: 4
   Remaining features: 13
   🎯 Test MAPE: nan% (Δ +nan%)

🧪 Test 4: PRICE-ONLY BASELINE
   Using only: 2 price features
   🎯 Test MAPE: 35.82% (Δ +2.17%)

================================================================================
📊 ABLATION STUDY RESULTS SUMMARY
================================================================================

🎯 Baseline Model (All Features): 33.65% MAPE

      Test  Features Removed  Remaining Features      MAPE  MAPE Increase
Price Only                15                   2 35.821864       2.169675
No Weather                 8                   9 30.416264      -3.235924
 No Supply                 4                  13       NaN            NaN

🔬 INTERPRETATION:
   Most critical: Price Only (MAPE increases by 2.17%)
   This proves price only features drive 6.4% of model performance

✅ Results saved to 'ablation_study_results.csv'
```

```
# 📸 CELL 55: Ablation Study Visualization
# Visualize Ablation Study Results
fig, ax = plt.subplots(1, 1, figsize=(12, 6))

# Bar chart
colors = ['#e74c3c' if x > 0 else '#2ecc71' for x in ablation_df['MAPE Increase']]
bars = ax.barh(ablation_df['Test'], ablation_df['MAPE Increase'], color=colors, alpha=0.8)

# Add baseline line
ax.axvline(x=0, color='black', linestyle='--', linewidth=2, label=f'Baseline ({baseline_mape:.2f}%)')

# Annotations
for i, (test, increase) in enumerate(zip(ablation_df['Test'], ablation_df['MAPE Increase'])):
    ax.text(increase + 0.3, i, f'+{increase:.2f}%', va='center', fontsize=11, fontweight='bold')

ax.set_xlabel('MAPE Increase vs Baseline (%)', fontsize=12, fontweight='bold')
ax.set_title('🔬 Ablation Study: Feature Category Importance\n(Higher = More Critical)',
             fontsize=14, fontweight='bold', pad=20)
ax.legend(fontsize=11)
ax.grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.savefig('/content/ablation_study_visualization.png', dpi=300, bbox_inches='tight')
plt.show()

print("✅ Ablation visualization saved to 'ablation_study_visualization.png'")
```




```
✅ Ablation visualization saved to 'ablation_study_visualization.png'
```

```
# 📸 CELL 60: Compare LSTM vs Random Forest
print("="*80)
print("🔬 COMPARING LSTM VS RANDOM FOREST (Statistical Tests)")
print("="*80)

# NOTE: You need to run Random Forest on the SAME test set and save predictions
# For demonstration, we'll use the reported results
```

```
================================================================================
🔬 COMPARING LSTM VS RANDOM FOREST (Statistical Tests)
================================================================================
```

```
# 📸 CELL 61: SHAP Feature Importance Analysis
print("="*80)
print("📊 SHAP ANALYSIS - Model Interpretability")
print("="*80)

import shap

print("\n⏳ Computing SHAP values (this may take a few minutes)...")

# Sample test data for SHAP (computational efficiency)
np.random.seed(42)
test_sample_indices = np.random.choice(len(X_test), size=min(200, len(X_test)), replace=False)
X_test_sample = X_test[test_sample_indices]

# SHAP DeepExplainer for LSTM
background = X_test_sample[:50]  # Background dataset
explainer = shap.DeepExplainer(model, background)

# Calculate SHAP values
shap_values = explainer.shap_values(X_test_sample)

# Get feature values for the last timestep (most relevant for predictions)
X_test_sample_last = X_test_sample[:, -1, :]  # Last timestep

print(f"✅ SHAP values computed for {len(X_test_sample)} test samples")

# Calculate mean absolute SHAP values per feature
mean_abs_shap = np.abs(shap_values).mean(axis=0).mean(axis=0)  # Average over samples and timesteps

# Create SHAP importance dataframe
shap_importance_df = pd.DataFrame({
```

```
    'Feature': final_features,
    'Mean |SHAP|': mean_abs_shap
}).sort_values('Mean |SHAP|', ascending=False)

print(f"\n🏆 TOP 20 FEATURES BY SHAP IMPORTANCE:")
print(shap_importance_df.head(20).to_string(index=False))

# Save SHAP importance
shap_importance_df.to_csv('/content/shap_feature_importance.csv', index=False)
print(f"\n✅ SHAP importance saved to 'shap_feature_importance.csv'")
```

```
================================================================================
📊 SHAP ANALYSIS - Model Interpretability
================================================================================

⏳ Computing SHAP values (this may take a few minutes)...
--------------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-3466302592.py in <cell line: 0>()
     15 # SHAP DeepExplainer for LSTM
     16 background = X_test_sample[:50]  # Background dataset
---> 17 explainer = shap.DeepExplainer(model, background)
     18
     19 # Calculate SHAP values

NameError: name 'model' is not defined
```

Next steps: Explain error

```
# 📸 CELL 62: SHAP Visualization & Comparison
print("\n⏳ Generating SHAP summary plot...")

# Defensive: Check if X_test_sample exists from Cell 61
try:
    # Get feature values from Cell 61's sample
    X_test_sample_last = X_test_sample[:, -1, :]  # Last timestep
except NameError:
    # If Cell 61 not run, create sample from X_test
    print("⚠️  Warning: X_test_sample not found. Run Cell 61 first for consistent results.")
    print("   Creating new sample from X_test for SHAP visualization...")
    np.random.seed(42)
    test_sample_indices = np.random.choice(len(X_test), size=min(200, len(X_test)), replace=False)
    X_test_sample = X_test[test_sample_indices]
    X_test_sample_last = X_test_sample[:, -1, :]  # Last timestep

# SHAP summary plot
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
shap.summary_plot(
    shap_values[:, -1, :],  # Last timestep SHAP values
    X_test_sample_last,
    feature_names=final_features,
    show=False,
    max_display=20
)
plt.title('🔍 SHAP Feature Importance - LSTM Model\n(Top 20 Features)',
          fontsize=14, fontweight='bold', pad=20)
plt.tight_layout()
plt.savefig('/content/shap_summary_plot.png', dpi=300, bbox_inches='tight')
plt.show()

print("✅ SHAP summary plot saved to 'shap_summary_plot.png'")
```

```
# 📸 CELL 63: Compare SHAP with Random Forest Importance
print("\n📊 COMPARING SHAP (LSTM) vs RANDOM FOREST IMPORTANCE")
print("="*80)

# Normalize both importance scores to 0-1
shap_importance_df['Normalized SHAP'] = (
    shap_importance_df['Mean |SHAP|'] / shap_importance_df['Mean |SHAP|'].sum()
)

# Load RF importance from earlier cell (Cell 10)
# Assuming rf_importance is available from feature selection step
try:
    # Get RF importance for final features
```