Random Forest for Carrot Price Prediction

4th Year Academic Research - Model Comparison

Dataset: Dambulla Market Carrot Prices (2020-2025)

Goal: Compare Random Forest with LSTM models

Benchmark to beat: Bidirectional LSTM - 19.30% MAPE

Contents


```
# Install packages
!pip install scikit-learn matplotlib seaborn pandas numpy joblib -q
print("✅ Packages installed")
```

```
✅ Packages installed
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib
import warnings
warnings.filterwarnings('ignore')

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

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

```
✅ All packages loaded successfully
```

```
from google.colab import drive
drive.mount('/content/drive')
print("Successfully mounted!")
```

```
Mounted at /content/drive
Successfully mounted!
```

PART A: DATA LOADING & PREPROCESSING

```
csv_file_path = "/content/drive/MyDrive/RESEARCH-ALL-in-one/ALL-Data-in-one-CSV/best-dataset/dambulla_market_dataset.csv"

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("="*60)
print("📊 DATA LOADED")
```

```
print("="*60)
print(f"Shape: {df.shape}")
print(f"Date range: {df.index.min()} to {df.index.max()}")
print(f"Columns: {len(df.columns)}")
print(f"Target variable: carrot_price")
print(f"  Min: {df['carrot_price'].min():.2f} Rs")
print(f"  Max: {df['carrot_price'].max():.2f} Rs")
print(f"  Mean: {df['carrot_price'].mean():.2f} Rs")
print(f"  Std: {df['carrot_price'].std():.2f} Rs")
```

```
============================================================
📊 DATA LOADED
============================================================
Shape: (2017, 46)
Date range: 2020-01-01 00:00:00 to 2025-07-11 00:00:00
Columns: 46
Target variable: carrot_price
  Min: 53.00 Rs
  Max: 1950.00 Rs
  Mean: 236.80 Rs
  Std: 177.85 Rs
```

```
# Data quality check
print("="*60)
print("🔍 DATA QUALITY CHECK")
print("="*60)

missing = df.isnull().sum()
if missing.sum() > 0:
    print(f"⚠️ Missing values found:")
    print(missing[missing > 0])
else:
    print("✅ No missing values")

# Check 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: {len(outliers)} ({len(outliers)/len(df)*100:.2f}%)")

print(f"\n✅ Data quality check complete")
```

```
============================================================
🔍 DATA QUALITY CHECK
============================================================
✅ No missing values

📉 Outliers: 103 (5.11%)

✅ Data quality check complete
```

```
# Transform supply factors: 1→2 (high), -1→1 (normal), 0→0 (low)
supply_cols = [col for col in df.columns if 'supply_factor' in col]
df_transformed = df.copy()
supply_mapping = {1: 2, -1: 1, 0: 0}

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

print(f"✅ Transformed {len(supply_cols)} supply factor columns")
print(f"   Encoding: 2=HIGH, 1=NORMAL, 0=LOW")
```

```
✅ Transformed 15 supply factor columns
   Encoding: 2=HIGH, 1=NORMAL, 0=LOW
```

PART B: FEATURE ENGINEERING

Note: Random Forest doesn't use sequences like LSTM. Instead, we create lag features and rolling statistics as regular columns.

```
print("="*60)
print("🔧 FEATURE ENGINEERING FOR RANDOM FOREST")
print("="*60)

df_features = df_transformed.copy()
```

```
# ==========================================
# 1. PRICE LAG FEATURES
# ==========================================
print("\n1️⃣ Creating price lag features...")
df_features['price_lag_1'] = df_features['carrot_price'].shift(1)
df_features['price_lag_2'] = df_features['carrot_price'].shift(2)
df_features['price_lag_3'] = df_features['carrot_price'].shift(3)
df_features['price_lag_7'] = df_features['carrot_price'].shift(7)
df_features['price_lag_14'] = df_features['carrot_price'].shift(14)
df_features['price_lag_21'] = df_features['carrot_price'].shift(21)
df_features['price_lag_30'] = df_features['carrot_price'].shift(30)

# Rolling statistics
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_mean_30'] = df_features['carrot_price'].rolling(window=30, min_periods=1).mean()
df_features['price_rolling_std_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).std()
df_features['price_rolling_std_14'] = df_features['carrot_price'].rolling(window=14, min_periods=1).std()
df_features['price_rolling_min_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).min()
df_features['price_rolling_max_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).max()
df_features['price_rolling_median_7'] = df_features['carrot_price'].rolling(window=7, min_periods=1).median()

# Price changes
df_features['price_change'] = df_features['carrot_price'].diff()
df_features['price_change_pct'] = df_features['carrot_price'].pct_change()
df_features['price_change_7d'] = df_features['carrot_price'].diff(7)
df_features['price_change_14d'] = df_features['carrot_price'].diff(14)

# Volatility
df_features['price_volatility_7'] = df_features['carrot_price'].rolling(window=7).std() / df_features['carrot_price'].roll

print(f"   Created {len([c for c in df_features.columns if 'price' in c]) - 1} price features")

# ==========================================
# 2. PRECIPITATION FEATURES
# ==========================================
print("\n2️⃣ Creating precipitation features...")
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)
    df_features[f'{col}_lag_7'] = df_features[col].shift(7)
    # Rolling sum (total rain)
    df_features[f'{col}_rolling_sum_7'] = df_features[col].rolling(window=7, min_periods=1).sum()
    df_features[f'{col}_rolling_sum_14'] = df_features[col].rolling(window=14, min_periods=1).sum()

# Regional precipitation groups
base_precip_cols = [col for col in df_features.columns if 'precipitation' in col and 'lag' not in col and 'rolling' not in

PRECIP_GROUPS = {
    'central_highland': [col for col in base_precip_cols if any(x in col for x in ['nuwaraeliya', 'kandapola', 'ragala', '
    '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', 'thalawa
}

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)
        df_features[f'precip_{group_name}_sum'] = df_features[cols].sum(axis=1)
        df_features[f'precip_{group_name}_mean_lag_1'] = df_features[f'precip_{group_name}_mean'].shift(1)
        df_features[f'precip_{group_name}_rolling_sum_7'] = df_features[f'precip_{group_name}_mean'].rolling(7).sum()

print(f"   Created precipitation features for {len(precip_cols)} regions and {len(PRECIP_GROUPS)} groups")

# ==========================================
# 3. SUPPLY FACTOR FEATURES
# ==========================================
print("\n3️⃣ Creating supply factor features...")
for col in supply_cols:
    df_features[f'{col}_lag_1'] = df_features[col].shift(1)
    df_features[f'{col}_lag_7'] = df_features[col].shift(7)
    df_features[f'{col}_rolling_mean_7'] = df_features[col].rolling(window=7, min_periods=1).mean()
```

```
    df_features[f'{col}_rolling_mean_14'] = df_features[col].rolling(window=14, min_periods=1).mean()

print(f"   Created supply features for {len(supply_cols)} regions")

# ============================================
# 4. FUEL PRICE FEATURES
# ============================================
print("\n4️⃣ Creating fuel price 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)
    df_features[f'{col}_lag_7'] = df_features[col].shift(7)
    df_features[f'{col}_rolling_mean_7'] = df_features[col].rolling(window=7, min_periods=1).mean()

print(f"   Created fuel price features for {len(fuel_cols)} types")

# ============================================
# 5. TEMPORAL FEATURES
# ============================================
print("\n5️⃣ Creating 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['week_of_year'] = df_features.index.isocalendar().week
df_features['is_weekend'] = (df_features.index.dayofweek >= 5).astype(int)
df_features['is_month_start'] = df_features.index.is_month_start.astype(int)
df_features['is_month_end'] = df_features.index.is_month_end.astype(int)

# Cyclical encoding for temporal features
df_features['day_of_week_sin'] = np.sin(2 * np.pi * df_features['day_of_week'] / 7)
df_features['day_of_week_cos'] = np.cos(2 * np.pi * df_features['day_of_week'] / 7)
df_features['month_sin'] = np.sin(2 * np.pi * df_features['month'] / 12)
df_features['month_cos'] = np.cos(2 * np.pi * df_features['month'] / 12)

print(f"   Created temporal features")

# ============================================
# 6. INTERACTION FEATURES
# ============================================
print("\n6️⃣ Creating interaction features...")
df_features['demand_x_trading'] = df_features['dambulla_demand'] * df_features['dambulla_is_trading_activities_high_or_low
df_features['demand_x_market_open'] = df_features['dambulla_demand'] * df_features['is_market_open']
df_features['market_open_x_weekend'] = df_features['is_market_open'] * df_features['is_weekend']

print(f"   Created interaction features")

# Fill NaN values
df_features = df_features.fillna(method='ffill').fillna(method='bfill')

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

print("\n" + "="*60)
print("✅ FEATURE ENGINEERING COMPLETE")
print("="*60)
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]}")
print(f"Missing values: {df_features.isnull().sum().sum()}")
```

```
============================================================
🔧 FEATURE ENGINEERING FOR RANDOM FOREST
============================================================

1️⃣ Creating price lag features...
   Created 20 price features

2️⃣ Creating precipitation features...
   Created precipitation features for 17 regions and 4 groups

3️⃣ Creating supply factor features...
   Created supply features for 15 regions

4️⃣ Creating fuel price features...
```

```
   Created fuel price features for 9 types

5️⃣ Creating temporal features...
   Created temporal features

6️⃣ Creating interaction features...
   Created interaction features

============================================================
✅ FEATURE ENGINEERING COMPLETE
============================================================
Total features created: 273
Original features: 46
New features: 227
Missing values: 0
```

PART C: FEATURE SELECTION (BASIC APPROACH)

Note: This is the basic feature selection. Advanced selection with heatmaps is in Part C.1 below.

```
print("="*60)
print("🎯 FEATURE SELECTION")
print("="*60)

# Remove rows with NaN (from lag features at the start)
# Keep last 30 days as test set before removing NaN
df_clean = df_features.copy()

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

print("\n📊 TOP 30 FEATURES BY CORRELATION:")
print(correlations.head(30))

# Quick Random Forest for feature importance
print("\n🌲 Training Random Forest for feature importance...")
X_temp = df_clean.drop('carrot_price', axis=1)
y_temp = df_clean['carrot_price']

# Remove infinite values
X_temp = X_temp.replace([np.inf, -np.inf], np.nan)
X_temp = X_temp.fillna(X_temp.median())

rf_temp = RandomForestRegressor(n_estimators=100, max_depth=15, random_state=42, n_jobs=-1)
rf_temp.fit(X_temp, y_temp)

feature_importance = pd.DataFrame({
    'feature': X_temp.columns,
    'rf_importance': rf_temp.feature_importances_
}).sort_values('rf_importance', ascending=False)

print("\n🏆 TOP 30 FEATURES BY RF IMPORTANCE:")
print(feature_importance.head(30))

# Combine approaches
corr_df = pd.DataFrame({'feature': correlations.index, 'correlation': correlations.values})
feature_scores = feature_importance.merge(corr_df, on='feature')
feature_scores['combined_score'] = (feature_scores['rf_importance'] * 0.6) + (feature_scores['correlation'] * 0.4)
feature_scores = feature_scores.sort_values('combined_score', ascending=False)

# Select features
# Strategy: Use more features for RF (it handles them well)
n_features_to_select = 50  # RF can handle more features than LSTM

selected_features = feature_scores['feature'].head(n_features_to_select).tolist()
selected_features = [f for f in selected_features if f != 'carrot_price']

# Remove highly correlated features (multicollinearity)
X_candidates = df_clean[selected_features]
candidates_corr = X_candidates.corr().abs()

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:  # Very high correlation
            col_i = candidates_corr.columns[i]
```

```
            col_j = candidates_corr.columns[j]
            # Keep the one with higher target correlation
            corr_i = abs(df_clean[col_i].corr(df_clean['carrot_price']))
            corr_j = abs(df_clean[col_j].corr(df_clean['carrot_price']))
            if corr_i < corr_j:
                features_to_remove.add(col_i)
            else:
                features_to_remove.add(col_j)

final_features = [f for f in selected_features if f not in features_to_remove]

print(f"\n🗑️ Removed {len(features_to_remove)} highly correlated features")
print(f"✅ FINAL FEATURES: {len(final_features)}")

# Category breakdown
categories = {
    'Price Features': [f for f in final_features if 'price' in f.lower()],
    'Market Features': [f for f in final_features if any(x in f.lower() for x in ['market', 'demand', 'trading', 'dambulla']
    'Weather Features': [f for f in final_features if 'precip' in f.lower()],
    'Supply Features': [f for f in final_features if 'supply' in f.lower()],
    'Fuel Features': [f for f in final_features if any(x in f.lower() for x in ['fur', 'lp_', 'lad', 'lsd', 'lk'])],
    'Temporal Features': [f for f in final_features if any(x in f.lower() for x in ['day', 'month', 'quarter', 'weekend', 'w
}

print("\n📋 FEATURE CATEGORIES:")
for cat, feats in categories.items():
    print(f"   {cat}: {len(feats)} features")

# Save feature importance
feature_scores[feature_scores['feature'].isin(final_features)].to_csv('/content/rf_feature_importance.csv', index=False)
print("\n✅ Feature importance saved to 'rf_feature_importance.csv'")
```

```
============================================================
🎯 FEATURE SELECTION
============================================================

📊 TOP 30 FEATURES BY CORRELATION:
carrot_price              1.000000
price_lag_1               0.960605
price_rolling_mean_7      0.952164
price_rolling_max_7       0.946407
price_rolling_min_7       0.937816
price_rolling_median_7    0.935490
price_lag_2               0.928781
price_rolling_mean_14     0.922189
price_lag_3               0.902969
price_lag_7               0.865969
price_rolling_mean_30     0.830756
price_rolling_std_14      0.821104
price_lag_14              0.773479
price_rolling_std_7       0.720441
price_lag_21              0.645538
price_lag_30              0.520868
price_change_14d          0.337815
lsd                       0.324175
lsd_lag_1                 0.323366
lsd_rolling_mean_7        0.322512
lsd_lag_7                 0.319677
Lp_95                     0.314247
Lp_95_rolling_mean_7      0.314070
Lp_95_lag_1               0.314019
Lp_95_lag_7               0.312993
Lp_92                     0.307039
Lp_92_lag_1               0.306769
Lp_92_rolling_mean_7      0.306562
lad                       0.306155
lad_lag_1                 0.305529
Name: carrot_price, dtype: float64

🌲 Training Random Forest for feature importance...

🏆 TOP 30 FEATURES BY RF IMPORTANCE:
                                        feature  rf_importance
58                          price_rolling_max_7       0.368707
45                                  price_lag_1       0.265318
52                         price_rolling_mean_7       0.181808
57                          price_rolling_min_7       0.041184
193   hanguranketha_supply_factor_rolling_mean_14       0.012553
209  mandaramnuwara_supply_factor_rolling_mean_14       0.010919
61                            price_change_pct       0.009937
62                             price_change_7d       0.008370
60                                price_change       0.008102
```

```
201    yatawaththa_supply_factor_rolling_mean_14       0.006826
46                                   price_lag_2       0.006646
225      marassana_supply_factor_rolling_mean_14       0.006263
217        mathale_supply_factor_rolling_mean_14       0.006222
229       puttalam_supply_factor_rolling_mean_14       0.005580
185      kandapola_supply_factor_rolling_mean_14       0.005040
63                            price_change_14d       0.004314
205    thalawakale_supply_factor_rolling_mean_14       0.004235
```

PART C.1: ADVANCED FEATURE SELECTION WITH HEATMAPS

Goal: Use proper techniques including:

1. Correlation heatmap visualization
2. Mutual Information scoring
3. Combined feature importance
4. Multicollinearity removal
5. Recursive Feature Elimination (RFE)
6. SelectFromModel for stability

```
print("="*80)
print("📸 CELL 13 - SCREENSHOT 1: CORRELATION HEATMAP")
print("🔥 STEP 1: CORRELATION HEATMAP - TOP FEATURES")
print("="*80)

# Calculate correlation matrix
corr_matrix = df_clean.corr()
target_corr = corr_matrix['carrot_price'].abs().sort_values(ascending=False)

# Select top 40 features by correlation with target
top_40_features = target_corr.head(41).index.tolist()  # 41 includes target itself

print(f"\n📊 Top 40 Features by Correlation with Target:")
print(target_corr.head(40))

# Create correlation heatmap for top features
plt.figure(figsize=(20, 16))
sns.heatmap(
    df_clean[top_40_features].corr(),
    cmap='RdBu_r',
    center=0,
    annot=False,
    fmt=".2f",
    vmax=1,
    vmin=-1,
    linewidths=0.5,
    cbar_kws={'label': 'Correlation Coefficient'}
)
plt.title('Correlation Matrix - Top 40 Features (by correlation with carrot_price)',
          fontsize=16, fontweight='bold', pad=20)
plt.xticks(rotation=90, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig('/content/correlation_heatmap_top40.png', dpi=300, bbox_inches='tight')
plt.show()

print("\n✅ Correlation heatmap saved to 'correlation_heatmap_top40.png'")
```

```
================================================================================
📸 CELL 13 - SCREENSHOT 1: CORRELATION HEATMAP
🔥 STEP 1: CORRELATION HEATMAP - TOP FEATURES
================================================================================

📊 Top 40 Features by Correlation with Target:
carrot_price              1.000000
price_lag_1               0.960605
price_rolling_mean_7      0.952164
price_rolling_max_7       0.946407
price_rolling_min_7       0.937816
price_rolling_median_7    0.935490
price_lag_2               0.928781
price_rolling_mean_14     0.922189
price_lag_3               0.902969
price_lag_7               0.865969
price_rolling_mean_30     0.830756
price_rolling_std_14      0.821104
price_lag_14              0.773479
price_rolling_std_7       0.720441
price_lag_21              0.645538
price_lag_30              0.520868
price_change_14d          0.337815
lsd                       0.324175
lsd_lag_1                 0.323366
lsd_rolling_mean_7        0.322512
lsd_lag_7                 0.319677
Lp_95                     0.314247
Lp_95_rolling_mean_7      0.314070
Lp_95_lag_1               0.314019
Lp_95_lag_7               0.312993
```

```
print("="*80)
print("📸 CELL 14 - SCREENSHOT 2: COMBINED FEATURE SCORES")
print("🔥 STEP 2: MUTUAL INFORMATION + RF IMPORTANCE (COMBINED SCORING)")
print("="*80)

from sklearn.feature_selection import mutual_info_regression

# Prepare data
X_all = df_clean.drop(columns=['carrot_price']).replace([np.inf, -np.inf], np.nan).fillna(df_clean.median())
y_all = df_clean['carrot_price']

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

# 1. MUTUAL INFORMATION
print("\n⏳ Computing Mutual Information scores...")
mi_scores = mutual_info_regression(X_all, y_all, random_state=42, n_neighbors=5)
mi_series = pd.Series(mi_scores, index=X_all.columns).sort_values(ascending=False)

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

# 2. RANDOM FOREST IMPORTANCE (more robust with more estimators)
print("\n⏳ Computing Random Forest importance...")
rf_selector = RandomForestRegressor(n_estimators=200, max_depth=15, random_state=42, n_jobs=-1)
rf_selector.fit(X_all, y_all)
rf_importance = pd.Series(rf_selector.feature_importances_, index=X_all.columns).sort_values(ascending=False)

print("\n🏆 TOP 20 FEATURES BY RF IMPORTANCE:")
print(rf_importance.head(20))

# 3. COMBINE SCORES (Weighted: RF 60%, MI 40%)
print("\n⏳ Combining scores...")
feature_scores_advanced = pd.DataFrame({
    'feature': X_all.columns,
    'mi_score': mi_series,
    'rf_importance': rf_importance,
    'correlation': target_corr[X_all.columns]
})

# Normalize all scores to 0-1 range
for col in ['mi_score', 'rf_importance', 'correlation']:
    feature_scores_advanced[f'{col}_norm'] = (
        (feature_scores_advanced[col] - feature_scores_advanced[col].min()) /
        (feature_scores_advanced[col].max() - feature_scores_advanced[col].min())
    )
```

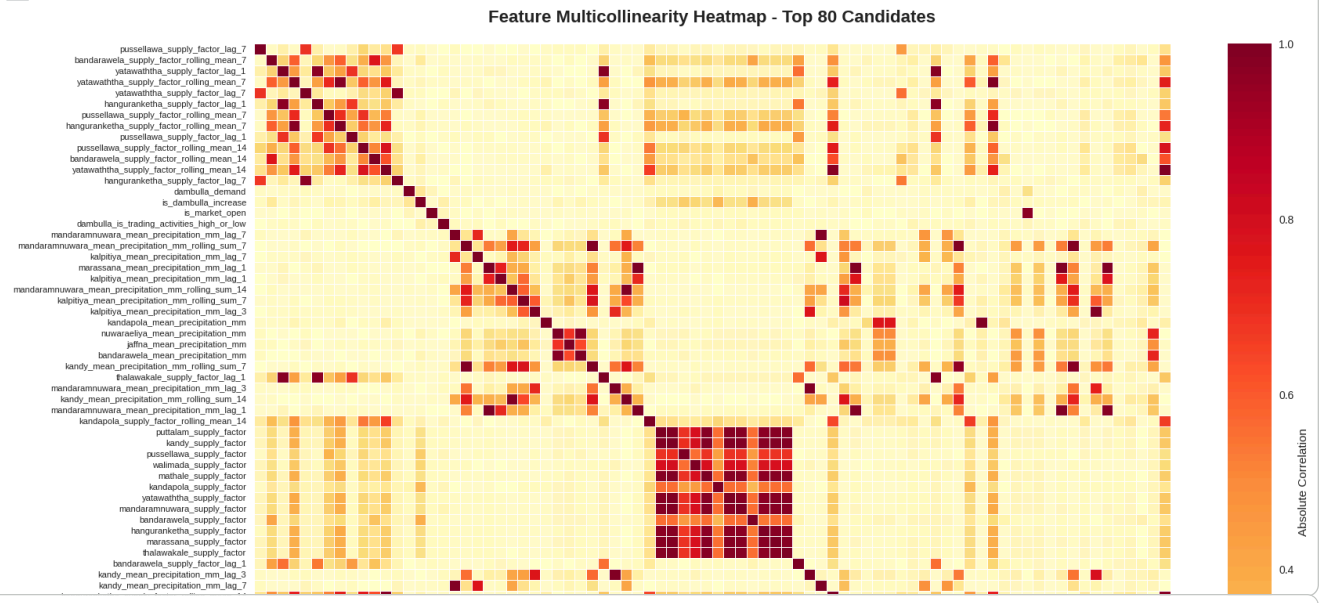
```
    )

# Combined score: RF(60%) + MI(30%) + Correlation(10%)
feature_scores_advanced['combined_score'] = (
    feature_scores_advanced['rf_importance_norm'] * 0.60 +
    feature_scores_advanced['mi_score_norm'] * 0.30 +
    feature_scores_advanced['correlation_norm'] * 0.10
)

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

print("\n🏆 TOP 30 FEATURES BY COMBINED SCORE:")
print(feature_scores_advanced[['feature', 'combined_score', 'rf_importance', 'mi_score', 'correlation']].head(30).to_string(

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

```
⏳ Computing Random Forest importance...

🏆 TOP 20 FEATURES BY RF IMPORTANCE:
price_rolling_max_7                           0.339112
price_lag_1                                   0.280064
price_rolling_mean_7                          0.199232
price_rolling_min_7                           0.033621
price_change_pct                              0.010014
hanguranketha_supply_factor_rolling_mean_14   0.009280
price_rolling_median_7                        0.008722
mandaramnuwara_supply_factor_rolling_mean_14  0.008421
price_change                                  0.008304
yatawaththa_supply_factor_rolling_mean_14     0.008275
price_change_7d                               0.007730
puttalam_supply_factor_rolling_mean_14        0.006751
kandy_supply_factor_rolling_mean_14           0.006653
marassana_supply_factor_rolling_mean_14       0.006567
thalawakale_supply_factor_rolling_mean_14     0.006285
pussellawa_supply_factor_rolling_mean_14      0.006003
kandapola_supply_factor_rolling_mean_14       0.005775
price_lag_2                                   0.004831
mathale_supply_factor_rolling_mean_14         0.004124
price_change_14d                              0.003989
dtype: float64

⏳ Combining scores...

🏆 TOP 30 FEATURES BY COMBINED SCORE:
                                        feature  combined_score  rf_importance  mi_score  correlation
                    pussellawa_supply_factor_lag_7        0.991860       0.339112  1.777204     0.946407
           bandarawela_supply_factor_rolling_mean_7        0.895525       0.280064  1.817541     0.960605
                    yatawaththa_supply_factor_lag_1        0.684641       0.199232  1.411720     0.952164
           yatawaththa_supply_factor_rolling_mean_7        0.447266       0.033621  1.757917     0.937816
                    yatawaththa_supply_factor_lag_7        0.371480       0.008722  1.567142     0.935490
                  hanguranketha_supply_factor_lag_1        0.329596       0.004831  1.359344     0.928781
            pussellawa_supply_factor_rolling_mean_7        0.308905       0.001516  1.273693     0.922189
          hanguranketha_supply_factor_rolling_mean_7        0.275621       0.000639  1.093603     0.902969
                    pussellawa_supply_factor_lag_1        0.257555       0.000582  1.008160     0.865969
           pussellawa_supply_factor_rolling_mean_14        0.251952       0.000817  0.993960     0.830756
           bandarawela_supply_factor_rolling_mean_14        0.218153       0.000197  0.832071     0.773479
           yatawaththa_supply_factor_rolling_mean_14        0.195440       0.002505  0.639600     0.821104
                  hanguranketha_supply_factor_lag_7        0.182805       0.000126  0.699597     0.645538
                                     dambulla_demand        0.173771       0.000011  0.861561     0.304874
                            is_dambulla_increase        0.173368       0.000006  0.857972     0.306769
                                  is_market_open        0.171284       0.000012  0.845121     0.307039
          dambulla_is_trading_activities_high_or_low        0.170151       0.000010  0.838577     0.306562
          mandaramnuwara_mean_precipitation_mm_lag_7        0.167103       0.000080  0.808224     0.324175
 mandaramnuwara_mean_precipitation_mm_rolling_sum_7        0.165542       0.000036  0.799741     0.323366
               kalpitiya_mean_precipitation_mm_lag_7        0.165410       0.000006  0.812487     0.302460
               marassana_mean_precipitation_mm_lag_1        0.164460       0.000059  0.793477     0.322512
               kalpitiya_mean_precipitation_mm_lag_1        0.163929       0.000006  0.801182     0.306155
mandaramnuwara_mean_precipitation_mm_rolling_sum_14        0.163329       0.000027  0.788764     0.319677
      kalpitiya_mean_precipitation_mm_rolling_sum_7        0.163187       0.000017  0.797372     0.304884
               kalpitiya_mean_precipitation_mm_lag_3        0.162847       0.000006  0.795018     0.305529
                    kandapola_mean_precipitation_mm        0.161726       0.000015  0.783408     0.312993
                  nuwaraeliya_mean_precipitation_mm        0.157329       0.000179  0.754328     0.314070
                       jaffna_mean_precipitation_mm        0.156565       0.000023  0.751411     0.314019
                  bandarawela_mean_precipitation_mm        0.155514       0.000029  0.744835     0.314247
```

```
print("="*80)
print("📸 CELL 15 - SCREENSHOT 3: MULTICOLLINEARITY HEATMAP")
print("🔥 STEP 3: REMOVE HIGHLY CORRELATED FEATURES (MULTICOLLINEARITY)")
print("="*80)
```

```
# Select top 80 candidates from combined score
n_candidates = 80
candidate_features = feature_scores_advanced['feature'].head(n_candidates).tolist()

print(f"\n📊 Starting with {len(candidate_features)} candidate features")

# Compute correlation matrix among candidates
X_candidates = X_all[candidate_features]
candidates_corr = X_candidates.corr().abs()

# Visualize multicollinearity
plt.figure(figsize=(18, 15))
sns.heatmap(
    candidates_corr,
    cmap='YlOrRd',
    annot=False,
    fmt=".2f",
    vmax=1,
    vmin=0,
    linewidths=0.5,
    cbar_kws={'label': 'Absolute Correlation'}
)
plt.title(f'Feature Multicollinearity Heatmap - Top {n_candidates} Candidates',
          fontsize=16, fontweight='bold', pad=20)
plt.xticks(rotation=90, ha='right', fontsize=8)
plt.yticks(rotation=0, fontsize=8)
plt.tight_layout()
plt.savefig('/content/multicollinearity_heatmap.png', dpi=300, bbox_inches='tight')
plt.show()

# Remove highly correlated features (threshold >= 0.95)
print("\n⏳ Removing features with correlation >= 0.95...")
features_to_remove = set()
correlation_threshold = 0.95

for i in range(len(candidates_corr.columns)):
    for j in range(i+1, len(candidates_corr.columns)):
        if candidates_corr.iloc[i, j] >= correlation_threshold:
            col_i = candidates_corr.columns[i]
            col_j = candidates_corr.columns[j]

            # Keep the feature 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 candidate_features if f not in features_to_remove]

print(f"\n🗑️  Removed {len(features_to_remove)} highly correlated features")
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):
        print(f"   - {feat}")
```

```
================================================================================
📸 CELL 15 - SCREENSHOT 3: MULTICOLLINEARITY HEATMAP
🔥 STEP 3: REMOVE HIGHLY CORRELATED FEATURES (MULTICOLLINEARITY)
================================================================================

📊 Starting with 80 candidate features
```




```
print("="*80)
print("📸 CELL 16 - SCREENSHOT 4: FINAL SELECTED FEATURES")
print("🔥 STEP 4: RECURSIVE FEATURE ELIMINATION (RFE) + SelectFromModel")
print("="*80)

from sklearn.feature_selection import RFE, SelectFromModel

X_reduced = X_all[reduced_features]

# METHOD 1: SelectFromModel (keeps features above median importance)
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 (Recursive Feature Elimination)
print("\n⏳ Method 2: Recursive Feature Elimination (RFE)...")
n_features_rfe = min(35, max(15, len(selected_from_model)))  # Target 15-35 features
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: Take intersection for stability (features selected by BOTH methods)
selected_features_final = sorted(list(set(selected_from_model) & set(selected_rfe)))

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

# If intersection too small, use union as fallback
if len(selected_features_final) < 10:
    print(f"⚠️  Intersection too small ({len(selected_features_final)}), using UNION instead...")
    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):
```

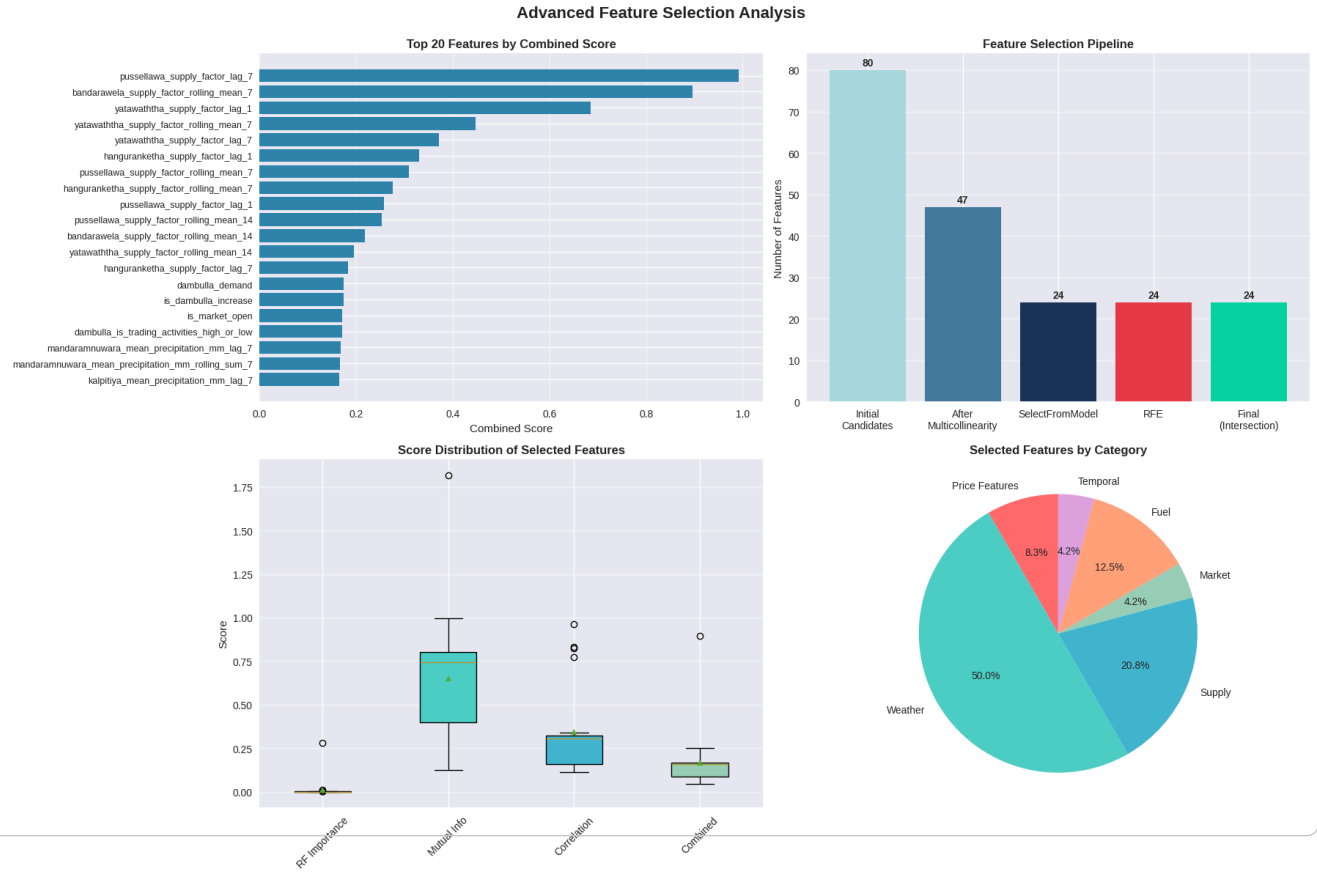
```
    score = feature_scores_advanced[feature_scores_advanced['feature'] == feat]['combined_score'].values[0]
    print(f"   {i:2d}. {feat:50s} (score: {score:.4f})")

# Save selected features
joblib.dump(selected_features_final, '/content/selected_features_advanced.pkl')
print("\n✅ Selected features saved to 'selected_features_advanced.pkl'")

# Update final_features variable for use in subsequent cells
final_features = selected_features_final.copy()
print(f"\n✅ Updated 'final_features' variable with {len(final_features)} advanced-selected features")
```

```
================================================================================
📸 CELL 16 - SCREENSHOT 4: FINAL SELECTED FEATURES
🔥 STEP 4: RECURSIVE FEATURE ELIMINATION (RFE) + SelectFromModel
================================================================================

⏳ Method 1: SelectFromModel (threshold='median')...
   Selected features: 24
   Top 10: ['bandarawela_supply_factor_rolling_mean_7', 'pussellawa_supply_factor_rolling_mean_14', 'bandarawela_supply_fac

⏳ Method 2: Recursive Feature Elimination (RFE)...
   Selected features: 24
   Top 10: ['bandarawela_supply_factor_rolling_mean_7', 'pussellawa_supply_factor_rolling_mean_14', 'bandarawela_supply_fac

================================================================================
🎯 FINAL SELECTED FEATURES (Intersection for Stability): 24
================================================================================

📋 SELECTED FEATURES:
    1. bandarawela_supply_factor_rolling_mean_14          (score: 0.2182)
    2. bandarawela_supply_factor_rolling_mean_7           (score: 0.8955)
    3. fur_1500_high_rolling_mean_7                       (score: 0.0468)
    4. is_dambulla_increase                               (score: 0.1734)
    5. jaffna_mean_precipitation_mm                       (score: 0.1566)
    6. kalpitiya_mean_precipitation_mm_lag_7              (score: 0.1654)
    7. kalpitiya_mean_precipitation_mm_rolling_sum_14     (score: 0.1251)
    8. kalpitiya_mean_precipitation_mm_rolling_sum_7      (score: 0.1632)
    9. kandapola_mean_precipitation_mm                    (score: 0.1617)
   10. kandapola_supply_factor_rolling_mean_14            (score: 0.1421)
   11. lad_lag_1                                          (score: 0.0608)
   12. lk_lag_1                                           (score: 0.0460)
   13. mandaramnuwara_mean_precipitation_mm_lag_3         (score: 0.1473)
   14. mandaramnuwara_mean_precipitation_mm_lag_7         (score: 0.1671)
   15. mandaramnuwara_mean_precipitation_mm_rolling_sum_14 (score: 0.1633)
   16. mandaramnuwara_mean_precipitation_mm_rolling_sum_7 (score: 0.1655)
   17. marassana_mean_precipitation_mm_lag_1              (score: 0.1645)
   18. precip_central_highland_mean                       (score: 0.0970)
   19. precip_uva_province_sum                            (score: 0.0446)
   20. price_lag_14                                       (score: 0.0455)
   21. price_rolling_mean_30                              (score: 0.0487)
   22. pussellawa_supply_factor_rolling_mean_14           (score: 0.2520)
   23. quarter                                            (score: 0.1088)
   24. yatawaththa_supply_factor_rolling_mean_14          (score: 0.1954)

✅ Selected features saved to 'selected_features_advanced.pkl'

✅ Updated 'final_features' variable with 24 advanced-selected features
```

```
print("="*80)
print("📸 CELL 17 - SCREENSHOT 5: FEATURE SELECTION SUMMARY DASHBOARD")
print("📊 FEATURE SELECTION SUMMARY & VISUALIZATION")
print("="*80)

# Create comparison visualization
fig, axes = plt.subplots(2, 2, figsize=(18, 12))

# 1. Top 20 features by combined score
top_20_combined = feature_scores_advanced.head(20)
axes[0, 0].barh(range(len(top_20_combined)), top_20_combined['combined_score'], color='#2E86AB')
axes[0, 0].set_yticks(range(len(top_20_combined)))
axes[0, 0].set_yticklabels(top_20_combined['feature'], fontsize=9)
axes[0, 0].set_xlabel('Combined Score', fontsize=11)
axes[0, 0].set_title('Top 20 Features by Combined Score', fontsize=12, fontweight='bold')
axes[0, 0].invert_yaxis()
axes[0, 0].grid(alpha=0.3, axis='x')

# 2. Feature selection method comparison
methods = ['Initial\nCandidates', 'After\nMulticollinearity', 'SelectFromModel', 'RFE', 'Final\n(Intersection)']
counts = [len(candidate_features), len(reduced_features), len(selected_from_model),
          len(selected_rfe), len(selected_features_final)]
colors_bar = ['#A8DADC', '#457B9D', '#1D3557', '#F63946', '#06D6A0']
```

```
axes[0, 1].bar(range(len(methods)), counts, color=colors_bar)
axes[0, 1].set_xticks(range(len(methods)))
axes[0, 1].set_xticklabels(methods, fontsize=10)
axes[0, 1].set_ylabel('Number of Features', fontsize=11)
axes[0, 1].set_title('Feature Selection Pipeline', fontsize=12, fontweight='bold')
axes[0, 1].grid(alpha=0.3, axis='y')
for i, v in enumerate(counts):
    axes[0, 1].text(i, v + 1, str(v), ha='center', fontweight='bold')

# 3. Score distribution comparison
score_types = ['RF Importance', 'Mutual Info', 'Correlation', 'Combined']
score_cols = ['rf_importance', 'mi_score', 'correlation', 'combined_score']
final_features_df = feature_scores_advanced[feature_scores_advanced['feature'].isin(selected_features_final)]

bp = axes[1, 0].boxplot(
    [final_features_df[col] for col in score_cols],
    labels=score_types,
    patch_artist=True,
    showmeans=True
)
for patch, color in zip(bp['boxes'], ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']):
    patch.set_facecolor(color)
axes[1, 0].set_ylabel('Score', fontsize=11)
axes[1, 0].set_title('Score Distribution of Selected Features', fontsize=12, fontweight='bold')
axes[1, 0].grid(alpha=0.3, axis='y')
axes[1, 0].tick_params(axis='x', rotation=45)

# 4. Category breakdown of selected features
categories_new = {
    'Price Features': [f for f in selected_features_final if 'price' in f.lower()],
    'Weather': [f for f in selected_features_final if 'precip' in f.lower()],
    'Supply': [f for f in selected_features_final if 'supply' in f.lower()],
    'Market': [f for f in selected_features_final if any(x in f.lower() for x in ['market', 'demand', 'trading', 'dambulla']
    'Fuel': [f for f in selected_features_final if any(x in f.lower() for x in ['fur', 'lp_', 'lad', 'lsd', 'lk'])],
    'Temporal': [f for f in selected_features_final if any(x in f.lower() for x in ['day', 'month', 'quarter', 'weekend', 'v
}

cat_names = list(categories_new.keys())
cat_counts = [len(feats) for feats in categories_new.values()]
axes[1, 1].pie(cat_counts, labels=cat_names, autopct='%1.1f%%', startangle=90,
               colors=['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFA07A', '#DDA0DD'])
axes[1, 1].set_title('Selected Features by Category', fontsize=12, fontweight='bold')

plt.suptitle('Advanced Feature Selection Analysis', fontsize=16, fontweight='bold', y=0.995)
plt.tight_layout()
plt.savefig('/content/feature_selection_summary.png', dpi=300, bbox_inches='tight')
plt.show()

print("\n📋 CATEGORY BREAKDOWN:")
for cat, feats in categories_new.items():
    print(f"   {cat}: {len(feats)} features")

print("\n✅ Feature selection summary saved to 'feature_selection_summary.png'")
```

```
================================================================================
📸 CELL 17 - SCREENSHOT 5: FEATURE SELECTION SUMMARY DASHBOARD
📊 FEATURE SELECTION SUMMARY & VISUALIZATION
================================================================================
```



PART D: PREPARE DATA FOR MODELING

```
📋 CATEGORY BREAKDOWN:
   Price Features: 2 features
```

```
print("="*60)
print("📊 PREPARING DATA FOR RANDOM FOREST")
print("="*60)

# Create final dataset
df_final = df_clean[final_features + ['carrot_price']].copy()

# Remove any remaining NaN or inf
df_final = df_final.replace([np.inf, -np.inf], np.nan)
df_final = df_final.fillna(method='ffill').fillna(method='bfill')

for col in df_final.columns:
    if df_final[col].isnull().any():
        df_final[col].fillna(df_final[col].median(), inplace=True)

print(f"✅ Final dataset: {df_final.shape}")
print(f"   Features: {len(final_features)}")
print(f"   Samples: {len(df_final)}")
print(f"   Missing values: {df_final.isnull().sum().sum()}")
print(f"   Infinite values: {np.isinf(df_final.select_dtypes(include=[np.number])).sum().sum()}")

# Time series split (70% train, 15% val, 15% test)
# IMPORTANT: No shuffling for time series!
train_size = int(len(df_final) * 0.70)
val_size = int(len(df_final) * 0.15)

train_data = df_final.iloc[:train_size]
val_data = df_final.iloc[train_size:train_size+val_size]
test_data = df_final.iloc[train_size+val_size:]

X_train = train_data[final_features]
y_train = train_data['carrot_price']

X_val = val_data[final_features]
y_val = val_data['carrot_price']
```

```
X_test = test_data[final_features]
y_test = test_data['carrot_price']

print("\n📊 DATA SPLIT (Time Series - No Shuffling):")
print(f"   Train: {len(X_train)} samples ({len(X_train)/len(df_final)*100:.1f}%)")
print(f"   Val:   {len(X_val)} samples ({len(X_val)/len(df_final)*100:.1f}%)")
print(f"   Test:  {len(X_test)} samples ({len(X_test)/len(df_final)*100:.1f}%)")

print("\n📅 Date ranges:")
print(f"   Train: {train_data.index.min()} to {train_data.index.max()}")
print(f"   Val:   {val_data.index.min()} to {val_data.index.max()}")
print(f"   Test:  {test_data.index.min()} to {test_data.index.max()}")
```

```
============================================================
📊 PREPARING DATA FOR RANDOM FOREST
============================================================
✅ Final dataset: (2017, 25)
   Features: 24
   Samples: 2017
   Missing values: 0
   Infinite values: 0

📊 DATA SPLIT (Time Series - No Shuffling):
   Train: 1411 samples (70.0%)
   Val:   302 samples (15.0%)
   Test:  304 samples (15.1%)

📅 Date ranges:
   Train: 2020-01-01 00:00:00 to 2023-11-13 00:00:00
   Val:   2023-11-14 00:00:00 to 2024-09-10 00:00:00
   Test:  2024-09-11 00:00:00 to 2025-07-11 00:00:00
```

PART E: RANDOM FOREST - BASELINE MODEL

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

def evaluate_model(model, X_train, y_train, X_val, y_val, X_test, y_test, model_name):
    # Predictions
    y_train_pred = model.predict(X_train)
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_test)

    # Metrics
    results = {
        'model': model_name,
        'train_mape': calc_mape(y_train, y_train_pred),
        'val_mape': calc_mape(y_val, y_val_pred),
        'test_mape': calc_mape(y_test, y_test_pred),
        'test_mae': mean_absolute_error(y_test, y_test_pred),
        'test_rmse': np.sqrt(mean_squared_error(y_test, y_test_pred)),
        'test_r2': r2_score(y_test, y_test_pred),
        'predictions': {
            'train': (y_train, y_train_pred),
            'val': (y_val, y_val_pred),
            'test': (y_test, y_test_pred)
        }
    }

    return results
```

```
print("="*60)
print("📸 CELL 20 - SCREENSHOT 6: BASELINE MODEL RESULTS")
print("🌲 RANDOM FOREST - BASELINE MODEL")
print("="*60)

# Baseline Random Forest with default parameters
rf_baseline = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
    n_jobs=-1,
    verbose=1
)

print("\n⏳ Training baseline model...")
```

```
rf_baseline.fit(X_train, y_train)

results_baseline = evaluate_model(rf_baseline, X_train, y_train, X_val, y_val, X_test, y_test, "RF Baseline")

print("\n✅ BASELINE RESULTS:")
print(f"   Train MAPE: {results_baseline['train_mape']:.2f}%")
print(f"   Val MAPE:   {results_baseline['val_mape']:.2f}%")
print(f"   Test MAPE:  {results_baseline['test_mape']:.2f}%")
print(f"   Test MAE:   {results_baseline['test_mae']:.2f} Rs")
print(f"   Test RMSE:  {results_baseline['test_rmse']:.2f} Rs")
print(f"   Test R²:    {results_baseline['test_r2']:.4f}")

# Compare with LSTM benchmark
lstm_benchmark = 19.30
print(f"\n📊 COMPARISON TO LSTM:")
print(f"   Best LSTM (Bidirectional): {lstm_benchmark:.2f}% MAPE")
print(f"   RF Baseline:               {results_baseline['test_mape']:.2f}% MAPE")
diff = lstm_benchmark - results_baseline['test_mape']
if diff > 0:
    print(f"   ✅ RF is BETTER by {diff:.2f}% points!")
elif diff < 0:
    print(f"   ⚠️ LSTM is better by {abs(diff):.2f}% points")
else:
    print(f"   Same performance")

# Save baseline model
joblib.dump(rf_baseline, '/content/rf_baseline_model.pkl')
print("\n✅ Baseline model saved")
```

```
============================================================
📸 CELL 20 - SCREENSHOT 6: BASELINE MODEL RESULTS
🌲 RANDOM FOREST - BASELINE MODEL
============================================================

⏳ Training baseline model...
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:    0.7s

✅ BASELINE RESULTS:
   Train MAPE: 4.14%
   Val MAPE:   29.53%
   Test MAPE:  34.13%
   Test MAE:   124.40 Rs
   Test RMSE:  179.98 Rs
   Test R²:    0.3800

📊 COMPARISON TO LSTM:
   Best LSTM (Bidirectional): 19.30% MAPE
   RF Baseline:               34.13% MAPE
   ⚠️ LSTM is better by 14.83% points

✅ Baseline model saved
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
```

PART F: HYPERPARAMETER TUNING

```
print("="*60)
print("📸 CELL 22 - SCREENSHOT 7: TUNED MODEL RESULTS")
print("🎯 HYPERPARAMETER TUNING")
print("="*60)

# Define parameter grid
param_distributions = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [10, 15, 20, 25, 30, None],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['sqrt', 'log2', 0.5, 0.7],
    'bootstrap': [True, False]
```

```
}

print("\n🔍 Parameter search space:")
for param, values in param_distributions.items():
    print(f"   {param}: {values}")

# Randomized search (faster than grid search)
rf_random = RandomForestRegressor(random_state=42, n_jobs=-1)

random_search = RandomizedSearchCV(
    estimator=rf_random,
    param_distributions=param_distributions,
    n_iter=50,  # Try 50 different combinations
    cv=3,  # 3-fold cross-validation
    scoring='neg_mean_absolute_error',
    random_state=42,
    n_jobs=-1,
    verbose=2
)

print("\n⏳ Starting hyperparameter search (this may take a few minutes)...")
random_search.fit(X_train, y_train)

print("\n✅ BEST PARAMETERS FOUND:")
for param, value in random_search.best_params_.items():
    print(f"   {param}: {value}")

# Best model
rf_tuned = random_search.best_estimator_

results_tuned = evaluate_model(rf_tuned, X_train, y_train, X_val, y_val, X_test, y_test, "RF Tuned")

print("\n✅ TUNED MODEL RESULTS:")
print(f"   Train MAPE: {results_tuned['train_mape']:.2f}%")
print(f"   Val MAPE:   {results_tuned['val_mape']:.2f}%")
print(f"   Test MAPE:  {results_tuned['test_mape']:.2f}%")
print(f"   Test MAE:   {results_tuned['test_mae']:.2f} Rs")
print(f"   Test RMSE:  {results_tuned['test_rmse']:.2f} Rs")
print(f"   Test R²:    {results_tuned['test_r2']:.4f}")

improvement = results_baseline['test_mape'] - results_tuned['test_mape']
print(f"\n📈 IMPROVEMENT:")
print(f"   Baseline: {results_baseline['test_mape']:.2f}% MAPE")
print(f"   Tuned:    {results_tuned['test_mape']:.2f}% MAPE")
print(f"   Gain:     {improvement:.2f}% points ({improvement/results_baseline['test_mape']*100:.1f}% relative)")

# Save tuned model
joblib.dump(rf_tuned, '/content/rf_tuned_model.pkl')
print("\n✅ Tuned model saved")
```

```
============================================================
📸 CELL 22 - SCREENSHOT 7: TUNED MODEL RESULTS
🎯 HYPERPARAMETER TUNING
============================================================

🔍 Parameter search space:
   n_estimators: [100, 200, 300, 500]
   max_depth: [10, 15, 20, 25, 30, None]
   min_samples_split: [2, 5, 10, 15]
   min_samples_leaf: [1, 2, 4, 8]
   max_features: ['sqrt', 'log2', 0.5, 0.7]
   bootstrap: [True, False]

⏳ Starting hyperparameter search (this may take a few minutes)...
Fitting 3 folds for each of 50 candidates, totalling 150 fits

✅ BEST PARAMETERS FOUND:
   n_estimators: 100
   min_samples_split: 2
   min_samples_leaf: 8
   max_features: 0.7
   max_depth: 10
   bootstrap: True

✅ TUNED MODEL RESULTS:
   Train MAPE: 9.82%
   Val MAPE:   27.65%
   Test MAPE:  34.10%
   Test MAE:   123.43 Rs
   Test RMSE:  178.08 Rs
```

```
   Test R²:    0.3931

📈 IMPROVEMENT:
   Baseline: 34.13% MAPE
   Tuned:    34.10% MAPE
   Gain:     0.03% points (0.1% relative)

✅ Tuned model saved
```

PART G: GRADIENT BOOSTING (BONUS COMPARISON)

```
print("="*60)
print("🚀 GRADIENT BOOSTING (Bonus Comparison)")
print("="*60)

# Gradient Boosting often performs better than RF
gb_model = GradientBoostingRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=4,
    subsample=0.8,
    random_state=42,
    verbose=1
)

print("\n⏳ Training Gradient Boosting...")
gb_model.fit(X_train, y_train)

results_gb = evaluate_model(gb_model, X_train, y_train, X_val, y_val, X_test, y_test, "Gradient Boosting")

print("\n✅ GRADIENT BOOSTING RESULTS:")
print(f"   Train MAPE: {results_gb['train_mape']:.2f}%")
print(f"   Val MAPE:   {results_gb['val_mape']:.2f}%")
print(f"   Test MAPE:  {results_gb['test_mape']:.2f}%")
print(f"   Test MAE:   {results_gb['test_mae']:.2f} Rs")
print(f"   Test RMSE:  {results_gb['test_rmse']:.2f} Rs")
print(f"   Test R²:    {results_gb['test_r2']:.4f}")

# Save GB model
joblib.dump(gb_model, '/content/gb_model.pkl')
print("\n✅ Gradient Boosting model saved")
```

```
============================================================
🚀 GRADIENT BOOSTING (Bonus Comparison)
============================================================

⏳ Training Gradient Boosting...
      Iter       Train Loss      OOB Improve   Remaining Time
         1       7973.9950        1367.2202            3.63s
         2       6495.8533         395.1366            2.88s
         3       5577.0921        1594.6547            2.71s
         4       4726.4311         915.7863            2.54s
         5       4018.6518         663.3923            2.42s
         6       3243.2833        -268.4299            2.38s
         7       2794.1909         545.6988            2.34s
         8       2523.8089         816.5646            2.28s
         9       2253.2326         501.0364            2.24s
        10       1894.4309         -83.1740            2.21s
        20        738.8775         -43.4662            2.07s
        30        415.4636         -53.1521            1.92s
        40        310.3835         -56.6236            1.78s
        50        252.7837         -28.8747            1.66s
        60        201.1987         -10.4237            1.54s
        70        182.3402          47.5333            1.42s
        80        155.9369          11.4072            1.30s
        90        137.6386           0.6709            1.19s
       100        118.1954          -6.2239            1.08s
       200         36.9928           7.4842            0.00s

✅ GRADIENT BOOSTING RESULTS:
   Train MAPE: 2.91%
   Val MAPE:   27.32%
   Test MAPE:  34.00%
   Test MAE:   122.63 Rs
   Test RMSE:  174.08 Rs
   Test R²:    0.4201
```

```
✅ Gradient Boosting model saved
```

PART H: COMPREHENSIVE MODEL COMPARISON

```
print("="*80)
print("📸 CELL 25 - SCREENSHOT 8: FINAL MODEL COMPARISON TABLE")
print("📊 COMPREHENSIVE MODEL COMPARISON")
print("="*80)

# Create comparison dataframe
all_results = [results_baseline, results_tuned, results_gb]

comparison_df = pd.DataFrame([
    {
        'Model': r['model'],
        'Train MAPE': f"{r['train_mape']:.2f}%",
        'Val MAPE': f"{r['val_mape']:.2f}%",
        'Test MAPE': f"{r['test_mape']:.2f}%",
        'Test MAE': f"{r['test_mae']:.2f} Rs",
        'Test RMSE': f"{r['test_rmse']:.2f} Rs",
        'Test R²': f"{r['test_r2']:.4f}"
    }
    for r in all_results
])

# Add LSTM results for comparison
lstm_comparison = pd.DataFrame([{
    'Model': 'LSTM (Bidirectional)',
    'Train MAPE': '13.09%',
    'Val MAPE': '14.88%',
    'Test MAPE': '19.30%',
    'Test MAE': '63.10 Rs',
    'Test RMSE': '91.99 Rs',
    'Test R²': '0.8384'
}])

full_comparison = pd.concat([comparison_df, lstm_comparison], ignore_index=True)

print("\n" + full_comparison.to_string(index=False))

# Find best model
best_rf_idx = comparison_df['Test MAPE'].str.rstrip('%').astype(float).idxmin()
best_rf_name = comparison_df.loc[best_rf_idx, 'Model']
best_rf_mape = float(comparison_df.loc[best_rf_idx, 'Test MAPE'].rstrip('%'))

print(f"\n🏆 BEST RANDOM FOREST MODEL: {best_rf_name}")
print(f"   Test MAPE: {best_rf_mape:.2f}%")

print(f"\n📊 FINAL COMPARISON:")
print(f"   Best LSTM:        19.30% MAPE (R² = 0.8384)")
print(f"   Best Random Forest: {best_rf_mape:.2f}% MAPE (R² = {comparison_df.loc[best_rf_idx, 'Test R²']})")

diff = 19.30 - best_rf_mape
if diff > 0:
    print(f"\n🎉 WINNER: Random Forest is BETTER by {diff:.2f}% points!")
elif diff < 0:
    print(f"\n🏆 WINNER: LSTM is better by {abs(diff):.2f}% points")
else:
    print(f"\n🤝 TIE: Both models perform equally")

# Save comparison
full_comparison.to_csv('/content/model_comparison_all.csv', index=False)
print("\n✅ Comparison saved to 'model_comparison_all.csv'")
```

```
================================================================================
📸 CELL 25 - SCREENSHOT 8: FINAL MODEL COMPARISON TABLE
📊 COMPREHENSIVE MODEL COMPARISON
================================================================================

              Model Train MAPE Val MAPE Test MAPE  Test MAE Test RMSE Test R²
        RF Baseline      4.14%   29.53%    34.13% 124.40 Rs 179.98 Rs  0.3800
           RF Tuned      9.82%   27.65%    34.10% 123.43 Rs 178.08 Rs  0.3931
   Gradient Boosting      2.91%   27.32%    34.00% 122.63 Rs 174.08 Rs  0.4201
LSTM (Bidirectional)     13.09%   14.88%    19.30%  63.10 Rs  91.99 Rs  0.8384
```

```
🏆 BEST RANDOM FOREST MODEL: Gradient Boosting
   Test MAPE: 34.00%

📊 FINAL COMPARISON:
   Best LSTM:        19.30% MAPE (R² = 0.8384)
   Best Random Forest: 34.00% MAPE (R² = 0.4201)

🏆 WINNER: LSTM is better by 14.70% points

✅ Comparison saved to 'model_comparison_all.csv'
```

PART I: FEATURE IMPORTANCE ANALYSIS

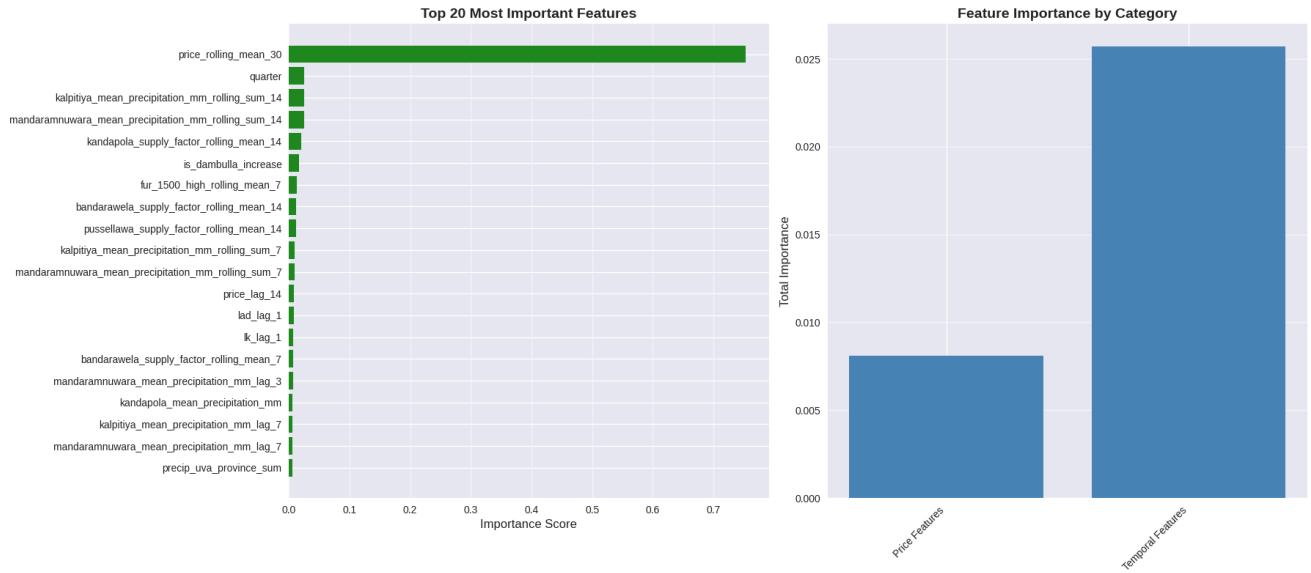
```
print("="*60)
print("📊 FEATURE IMPORTANCE ANALYSIS")
print("="*60)

# Get feature importance from best model
if best_rf_name == "RF Baseline":
    best_model = rf_baseline
elif best_rf_name == "RF Tuned":
    best_model = rf_tuned
else:
    best_model = gb_model

feature_imp_df = pd.DataFrame({
    'feature': final_features,
    'importance': best_model.feature_importances_
}).sort_values('importance', ascending=False)

print("\n🏆 TOP 30 MOST IMPORTANT FEATURES:")
print(feature_imp_df.head(30).to_string(index=False))

# Category-wise importance
category_importance = {}
for cat, feats in categories.items():
    cat_features = [f for f in feats if f in final_features]
    if cat_features:
        cat_imp = feature_imp_df[feature_imp_df['feature'].isin(cat_features)]['importance'].sum()
        category_importance[cat] = cat_imp

print("\n📊 IMPORTANCE BY CATEGORY:")
for cat, imp in sorted(category_importance.items(), key=lambda x: x[1], reverse=True):
    print(f"   {cat}: {imp:.4f} ({imp/sum(category_importance.values())*100:.1f}%)")

# Save feature importance
feature_imp_df.to_csv('/content/best_model_feature_importance.csv', index=False)
print("\n✅ Feature importance saved")
```

```
============================================================
📊 FEATURE IMPORTANCE ANALYSIS
============================================================

🏆 TOP 30 MOST IMPORTANT FEATURES:
                                        feature  importance
                            price_rolling_mean_30    0.753500
                                          quarter    0.025708
     kalpitiya_mean_precipitation_mm_rolling_sum_14    0.025070
mandaramnuwara_mean_precipitation_mm_rolling_sum_14    0.024988
            kandapola_supply_factor_rolling_mean_14    0.020801
                              is_dambulla_increase    0.016121
                       fur_1500_high_rolling_mean_7    0.013328
          bandarawela_supply_factor_rolling_mean_14    0.011795
           pussellawa_supply_factor_rolling_mean_14    0.011544
      kalpitiya_mean_precipitation_mm_rolling_sum_7    0.009780
 mandaramnuwara_mean_precipitation_mm_rolling_sum_7    0.009662
                                     price_lag_14    0.008084
                                        lad_lag_1    0.007990
                                        lk_lag_1    0.007360
           bandarawela_supply_factor_rolling_mean_7    0.006782
          mandaramnuwara_mean_precipitation_mm_lag_3    0.006740
                    kandapola_mean_precipitation_mm    0.006257
               kalpitiya_mean_precipitation_mm_lag_7    0.006032
          mandaramnuwara_mean_precipitation_mm_lag_7    0.005601
                           precip_uva_province_sum    0.005523
           yatawaththa_supply_factor_rolling_mean_14    0.004976
                       precip_central_highland_mean    0.004526
                       jaffna_mean_precipitation_mm    0.003946
```

```
marassana_mean_precipitation_mm_lag_1    0.003887

📊 IMPORTANCE BY CATEGORY:
   Temporal Features: 0.0257 (76.1%)
   Price Features: 0.0081 (23.9%)

✅ Feature importance saved
```

PART J: VISUALIZATIONS

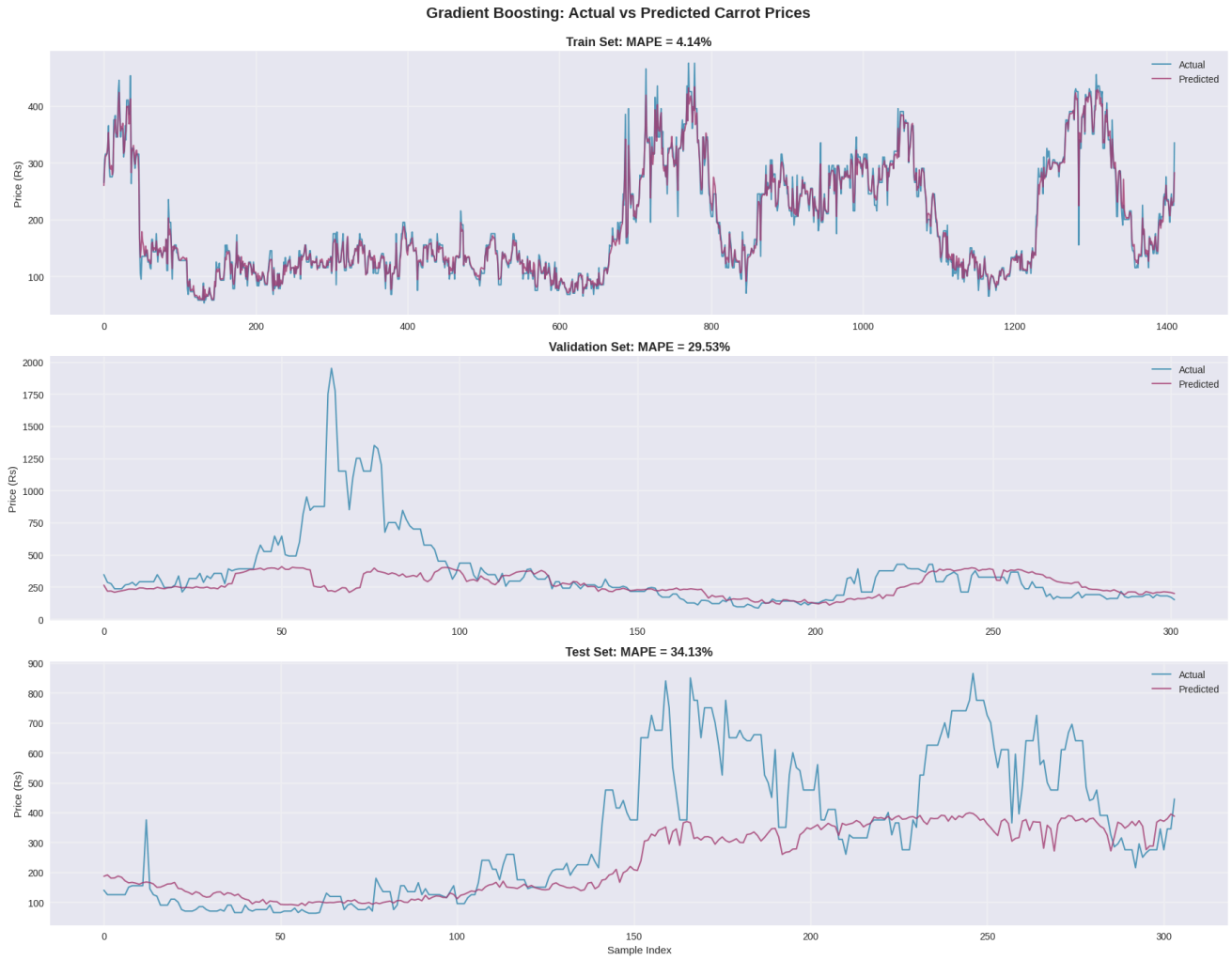
```
# 1. Feature Importance Plot
fig, axes = plt.subplots(1, 2, figsize=(18, 8))

# Top 20 features
top_20 = feature_imp_df.head(20)
axes[0].barh(range(len(top_20)), top_20['importance'], color='forestgreen')
axes[0].set_yticks(range(len(top_20)))
axes[0].set_yticklabels(top_20['feature'])
axes[0].set_xlabel('Importance Score', fontsize=12)
axes[0].set_title('Top 20 Most Important Features', fontsize=14, fontweight='bold')
axes[0].invert_yaxis()
axes[0].grid(alpha=0.3, axis='x')

# Category importance
cat_names = list(category_importance.keys())
cat_values = list(category_importance.values())
axes[1].bar(range(len(cat_names)), cat_values, color='steelblue')
axes[1].set_xticks(range(len(cat_names)))
axes[1].set_xticklabels(cat_names, rotation=45, ha='right')
axes[1].set_ylabel('Total Importance', fontsize=12)
axes[1].set_title('Feature Importance by Category', fontsize=14, fontweight='bold')
axes[1].grid(alpha=0.3, axis='y')

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

print("✅ Feature importance visualization saved")
```




```
✅ Feature importance visualization saved
```

```
# 2. Predictions Comparison
fig, axes = plt.subplots(3, 1, figsize=(18, 14))
```

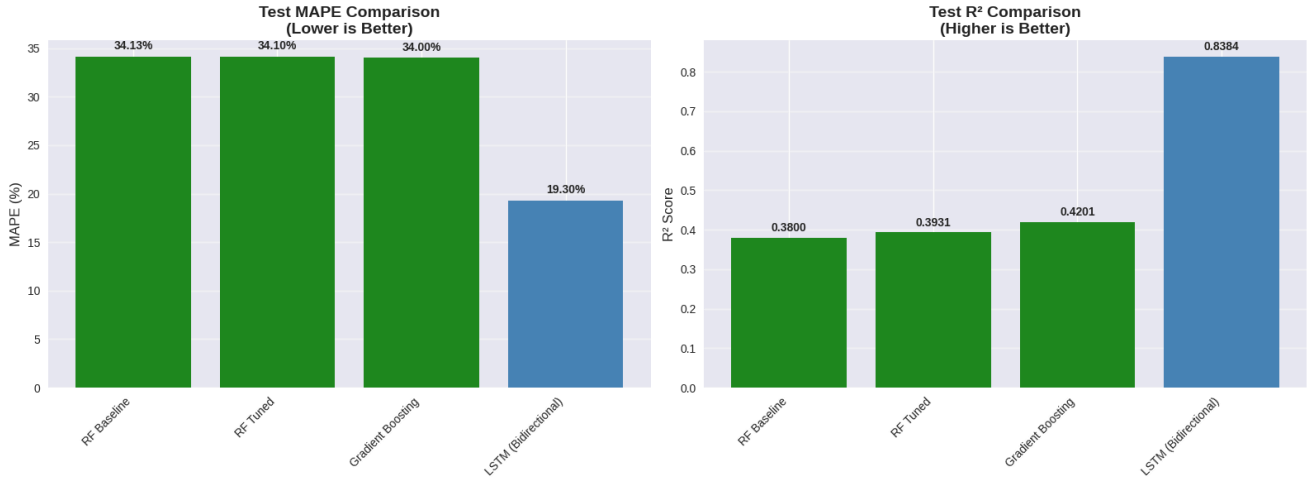
```
# Use best RF model predictions
best_rf_results = results_tuned if best_rf_name == "RF Tuned" else results_baseline

# Train
y_train_actual, y_train_pred = best_rf_results['predictions']['train']
axes[0].plot(y_train_actual.values, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[0].plot(y_train_pred, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[0].set_title(f'Train Set: MAPE = {best_rf_results["train_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
y_val_actual, y_val_pred = best_rf_results['predictions']['val']
axes[1].plot(y_val_actual.values, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[1].plot(y_val_pred, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[1].set_title(f'Validation Set: MAPE = {best_rf_results["val_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
y_test_actual, y_test_pred = best_rf_results['predictions']['test']
axes[2].plot(y_test_actual.values, label='Actual', linewidth=1.5, alpha=0.8, color='#2E86AB')
axes[2].plot(y_test_pred, label='Predicted', linewidth=1.5, alpha=0.8, color='#A23B72')
axes[2].set_title(f'Test Set: MAPE = {best_rf_results["test_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(f'{best_rf_name}: Actual vs Predicted Carrot Prices',
             fontsize=16, fontweight='bold', y=0.995)
plt.tight_layout()
plt.savefig('/content/predictions_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

print("✅ Predictions visualization saved")
```



✅ Predictions visualization saved

```
# 3. Model Comparison Bar Charts
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Extract numeric MAPE values
models = full_comparison['Model'].tolist()
mapes = [float(x.rstrip('%')) for x in full_comparison['Test MAPE'].tolist()]
r2s = [float(x) for x in full_comparison['Test R²'].tolist()]

# MAPE comparison
colors = ['forestgreen' if 'RF' in m or 'Gradient' in m else 'steelblue' for m in models]
axes[0].bar(range(len(models)), mapes, color=colors)
axes[0].set_xticks(range(len(models)))
axes[0].set_xticklabels(models, rotation=45, ha='right')
axes[0].set_ylabel('MAPE (%)', fontsize=12)
axes[0].set_title('Test MAPE Comparison\n(Lower is Better)', fontsize=14, fontweight='bold')
axes[0].grid(alpha=0.3, axis='y')
for i, v in enumerate(mapes):
    axes[0].text(i, v + 0.5, f'{v:.2f}%', ha='center', va='bottom', fontweight='bold')

# R² comparison
```

```
axes[1].bar(range(len(models)), r2s, color=colors)
axes[1].set_xticks(range(len(models)))
axes[1].set_xticklabels(models, rotation=45, ha='right')
axes[1].set_ylabel('R² Score', fontsize=12)
axes[1].set_title('Test R² Comparison\n(Higher is Better)', fontsize=14, fontweight='bold')
axes[1].grid(alpha=0.3, axis='y')
for i, v in enumerate(r2s):
    axes[1].text(i, v + 0.01, f'{v:.4f}', ha='center', va='bottom', fontweight='bold')

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

print("✅ Model comparison charts saved")
```




```
✅ Model comparison charts saved
```

PART K: FINAL SUMMARY & CONCLUSIONS

```
print("="*80)
print("🎉 FINAL SUMMARY - RANDOM FOREST FOR CARROT PRICE PREDICTION")
print("="*80)

print("\n📊 DATASET:")
print(f"   Total samples: {len(df_final)}")
print(f"   Date range: {df_final.index.min()} to {df_final.index.max()}")
print(f"   Features used: {len(final_features)}")
print(f"   Train/Val/Test: {len(X_train)}/{len(X_val)}/{len(X_test)} samples")

print("\n🌲 MODELS TRAINED:")
print("   1. Random Forest - Baseline")
print("   2. Random Forest - Tuned (Hyperparameter Optimization)")
print("   3. Gradient Boosting (Bonus)")

print("\n🎯 PERFORMANCE:")
print(full_comparison.to_string(index=False))

print(f"\n🏆 BEST MODEL: {best_rf_name}")
print(f"   Test MAPE: {best_rf_mape:.2f}%")
print(f"   Test R²: {comparison_df.loc[best_rf_idx, 'Test R²']}")

print("\n📊 FEATURE IMPORTANCE (Top 5):")
for i, row in feature_imp_df.head(5).iterrows():
    print(f"   {row['feature']}: {row['importance']:.4f}")

print("\n📈 COMPARISON TO LSTM:")
print(f"   Best LSTM (Bidirectional):  19.30% MAPE, 0.8384 R²")
print(f"   Best Random Forest:         {best_rf_mape:.2f}% MAPE, {comparison_df.loc[best_rf_idx, 'Test R²']} R²")
```