Crop Price Prediction Model - Data Preprocessing & Model Explanation

📌 Overview

This project is a **crop price prediction model** that estimates the price of various vegetables per kilogram based on historical data. The model uses **Random Forest Regression** and features like **vegetable type, season, month, farm size, condition of the crop, and recent disasters** to make predictions.

The implementation involves **data preprocessing, feature transformation, and training a machine learning model** to make accurate price predictions.

---

1 Data Preprocessing & Feature Engineering

Before training the model, we performed several preprocessing steps to **clean, transform, and prepare** the data.

1.1 Loading Dataset

The dataset is loaded using **Pandas**, assuming the file is named Expanded_Crop_price.csv.

```
df = pd.read_csv("/content/Expanded_Crop_price.csv")
```

Validation: We check if the Price per kg column exists. If not, an error is raised.

```
if 'Price per kg' not in df.columns:
    raise ValueError("Error: 'Price per kg' column is missing from the dataset.")
```

1.2 Handling Categorical Data

Some features in the dataset are categorical (e.g., Vegetable, Season, Vegetable Condition). These must be **converted into numerical format** using one-hot encoding before feeding them into the model.

1.2.1 Encoding the 'Month' Column

Since months are stored as text (Jan, Feb, ...), we convert them into numerical values (e.g., Jan = 1, Feb = 2, etc.).

```
month_mapping = {
    'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'may': 5, 'jun': 6,
    'jul': 7, 'aug': 8, 'sep': 9, 'oct': 10, 'nov': 11, 'dec': 12,
    'july': 7, 'sept': 9  # Handle alternate spellings
}
```

df['Month'] = df['Month'].map(month_mapping)

🔹 If there are **missing or unrecognized months**, we replace them with the most common month in the dataset.

```
if df['Month'].isnull().sum() > 0:
    df['Month'] = df['Month'].fillna(df['Month'].mode()[0])
```

1.3 Defining Features & Target Variable

The **target variable** is the price of the crop (Price per kg). All other columns are features used for prediction.

```
X = df.drop('Price per kg', axis=1)  # Features
y = df['Price per kg']  # Target variable
```

1.4 Splitting Data for Training & Testing

To evaluate the model, we split the dataset into **80% training** and **20% testing**.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1.5 Feature Selection

We classify features into **categorical** and **numerical** for separate preprocessing steps.

Categorical Features

- Vegetable
- Season
- Vegetable condition
- Deasaster Happen in last 3month

Numerical Features

- Month
- Farm size *(if present in the dataset)*

```
categorical_features = ['Vegetable', 'Season', 'Vegetable condition', 'Deasaster Happen in last 3month']
```

```
numerical_features = ['Month', 'Farm size'] if 'Farm size' in X_train.columns else ['Month']
```

- **Validation**: If any numerical features are missing, an error is raised.

```
missing_numerical_features = [feature for feature in numerical_features if feature not in X_train.columns]

if missing_numerical_features:

    raise ValueError(f"Error: Missing numerical features: {missing_numerical_features}")
```

1.6 Feature Transformation using ColumnTransformer

We apply **Standard Scaling** to numerical features and **One-Hot Encoding** to categorical features.

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer


preprocessor = ColumnTransformer(

    transformers=[

        ('num', StandardScaler(), numerical_features),

        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)

    ])
```

- **Why Standard Scaling?**

It **normalizes numerical values** (e.g., Month, Farm size) to improve model performance.

- **Why One-Hot Encoding?**

It **converts categorical values** (e.g., Vegetable, Season) into machine-readable numerical format.

```
X_train = preprocessor.fit_transform(X_train)

X_test = preprocessor.transform(X_test)
```

---

2 Training the Machine Learning Model

We use **Random Forest Regression**, a powerful model for predicting continuous values.

```
from sklearn.ensemble import RandomForestRegressor


model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train, y_train)


print("✅ Model training completed successfully!")
```

🔹 **Why Random Forest?**

- It is **robust to outliers** and missing values.
- It **performs well on structured/tabular data**.
- It can **handle both numerical and categorical data** efficiently.

---

3 Interactive User Input for Predictions

To **predict crop prices**, the user provides inputs through a **menu-based interactive system**.

3.1 Fetching Unique Values for Dropdowns

To ensure valid user input, we **dynamically fetch available values** for categorical features.

```
vegetable_options = df['Vegetable'].unique().tolist()

season_options = df['Season'].unique().tolist()

condition_options = df['Vegetable condition'].unique().tolist()

disaster_options = df['Deasaster Happen in last 3month'].unique().tolist()
```

3.2 User Inputs

The user selects options via an **interactive menu**:

```
vegetable = input(f"🌱 Select Vegetable {vegetable_options}: ").strip()

season = input(f"📅 Select Season {season_options}: ").strip()

condition = input(f"🥦 Select Vegetable Condition {condition_options}: ").strip()
```

disaster = input(f"🌍 Any Disaster in Last 3 Months {disaster_options}: ").strip()

3.3 Handling Month Input

The user enters a month, which is mapped to a numeric value:

month_name = input("📅 Enter Month (e.g., jan, feb, mar, apr): ").strip().lower()

month = month_mapping.get(month_name, None)

if month is None:

    print(f"⚠️ Invalid month '{month_name}', defaulting to January.")

    month = 1

---

4 Making Predictions

4.1 Creating an Input DataFrame

A **new DataFrame** is created with user inputs:

input_data = pd.DataFrame({

    'Vegetable': [vegetable],

    'Season': [season],

    'Vegetable condition': [condition],

    'Deasaster Happen in last 3month': [disaster],

    'Month': [month]

})

4.2 Transforming Input Data

We **apply the same preprocessing pipeline** to transform the input:

input_data = preprocessor.transform(input_data)

4.3 Predicting Crop Price

The trained **Random Forest model** predicts the crop price:

predicted_price = model.predict(input_data)[0]

print(f"\n💰 Predicted Crop Price: ₹{predicted_price:.2f} per kg\n")

---

5 Summary

✅ **Data Preprocessing**

- Handled missing data
- Encoded categorical features
- Normalized numerical values

✅ **Model Training**

- Used **Random Forest Regression**
- Achieved efficient price predictions

✅ **User Interaction**

- **Dropdown-based input system**
- **Dynamic error handling**
- **Live price prediction**

🎯 **This system helps farmers & traders predict crop prices based on real-world conditions!** 🚀