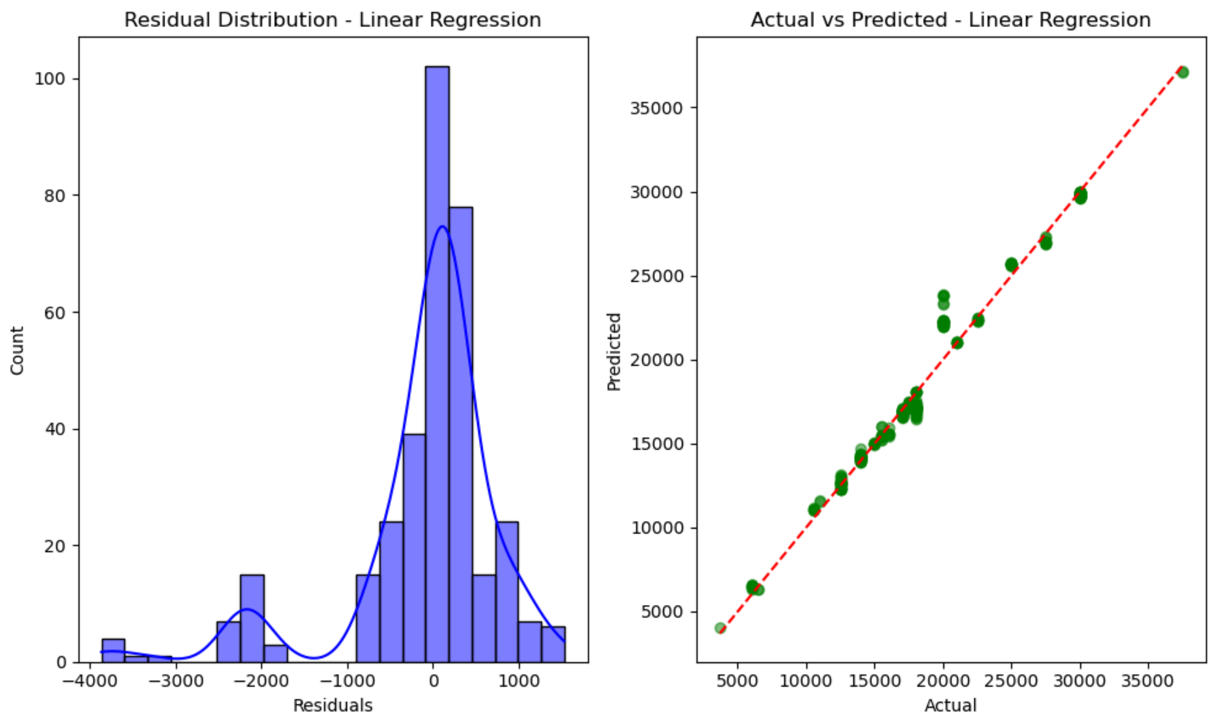
In [17]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv(r"C:\Users\LENOVO\Downloads\dataset_red_chilli.csv")

# Convert Price Date to datetime
data['Price Date'] = pd.to_datetime(data['Price Date'], errors='coerce')

# Handle rows where 'Price Date' failed to parse
data = data.dropna(subset=['Price Date'])

# Feature Engineering: Extract useful date features
data['Day'] = data['Price Date'].dt.day
data['Month'] = data['Price Date'].dt.month
data['Year'] = data['Price Date'].dt.year

# Drop original 'Price Date' as we have extracted relevant components
data.drop(['District Name', 'Market Name', 'Commodity', 'Variety', 'Grade', 'Price

# Define features (X) and target (y)
X = data.drop('Modal Price (Rs./Quintal)', axis=1)
y = data['Modal Price (Rs./Quintal)']

### 1. Detecting and Removing Outliers using IQR method ###
Q1 = X.quantile(0.25)
Q3 = X.quantile(0.75)
IQR = Q3 - Q1
# Filter out outliers
X = X[~((X < (Q1 - 1.5 * IQR)) | (X > (Q3 + 1.5 * IQR))).any(axis=1)]
y = y[X.index]  # Keep target values in sync with features

### 2. Feature Scaling ###
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

### 3. Train-Test Split ###
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran

### 4. Define models ###
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42),
    'Support Vector Regressor': SVR()
}
```

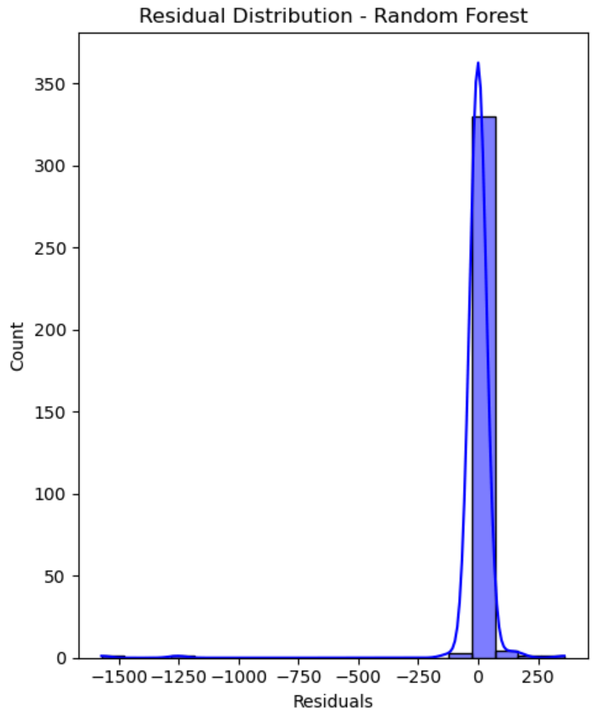
```
# Dictionary to store results
results = {}

### 5. Model Training and Evaluation ###
for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Calculate Metrics
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store the results
    results[name] = {'RMSE': rmse, 'MAE': mae, 'R²': r2}

    # Print metrics
    print(f"Model: {name}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE: {mae:.4f}")
    print(f"R²: {r2:.4f}\n")

    ### Residual Plot and Prediction vs Actual ###
    plt.figure(figsize=(10, 6))

    # Residuals Plot
    residuals = y_test - y_pred
    plt.subplot(1, 2, 1)
    sns.histplot(residuals, bins=20, kde=True, color='blue')
    plt.title(f"Residual Distribution - {name}")
    plt.xlabel("Residuals")

    # Predictions vs Actual Plot
    plt.subplot(1, 2, 2)
    plt.scatter(y_test, y_pred, alpha=0.5, color='green')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  #
    plt.title(f"Actual vs Predicted - {name}")
    plt.xlabel("Actual")
    plt.ylabel("Predicted")

    plt.tight_layout()
    plt.show()

### 6. Time vs Modal Price Plot ###
# Sort by date for time series plot
data['Price Date'] = pd.to_datetime(data['Year'].astype(str) + '-' + data['Month'].

plt.figure(figsize=(10, 6))
plt.plot(data['Price Date'], data['Modal Price (Rs./Quintal)'], marker='o', linesty
plt.title("Time vs Modal Price")
plt.xlabel("Date")
plt.ylabel("Modal Price (Rs./Quintal)")
```

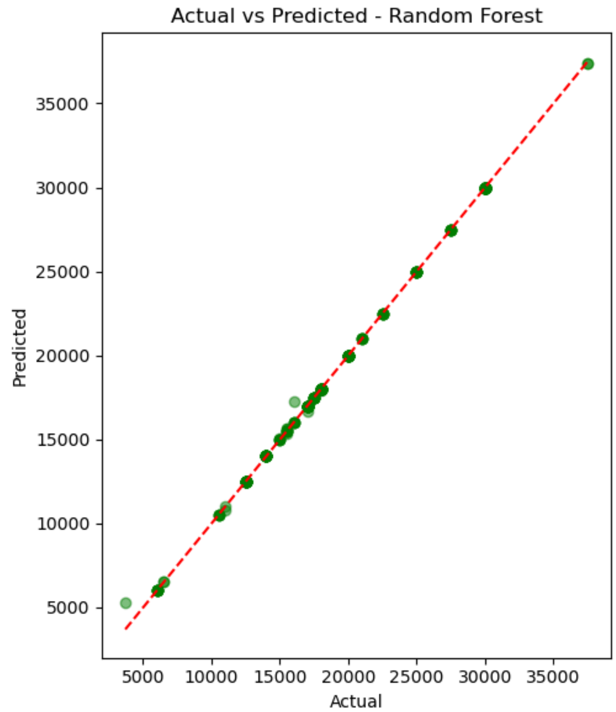
```
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_18988\3587156503.py:16: UserWarning: Co
uld not infer format, so each element will be parsed individually, falling back to `
dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  data['Price Date'] = pd.to_datetime(data['Price Date'], errors='coerce')
Model: Linear Regression
RMSE: 880.7107
MAE: 531.7584
R²: 0.9824
```

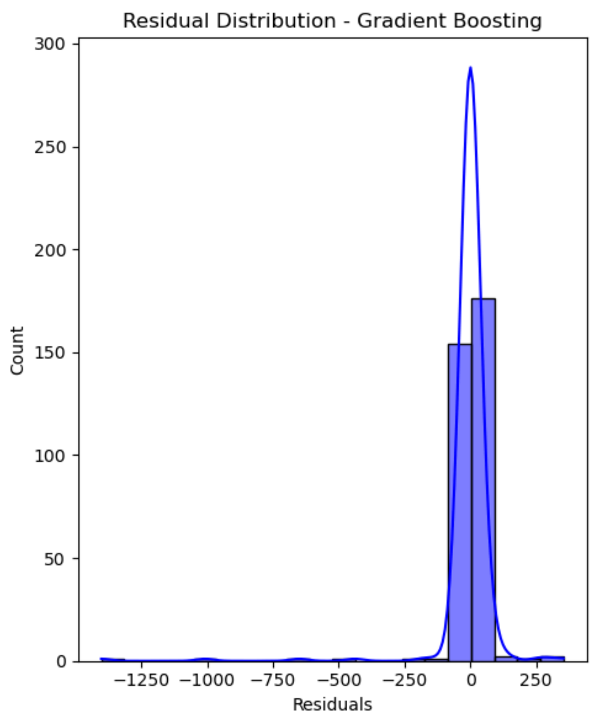


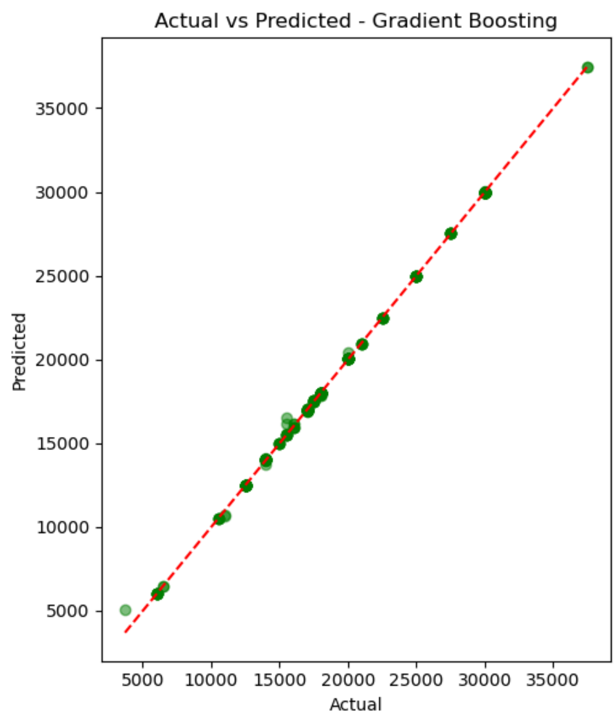

```
Model: Random Forest
RMSE: 112.4113
MAE: 12.4721
R²: 0.9997
```



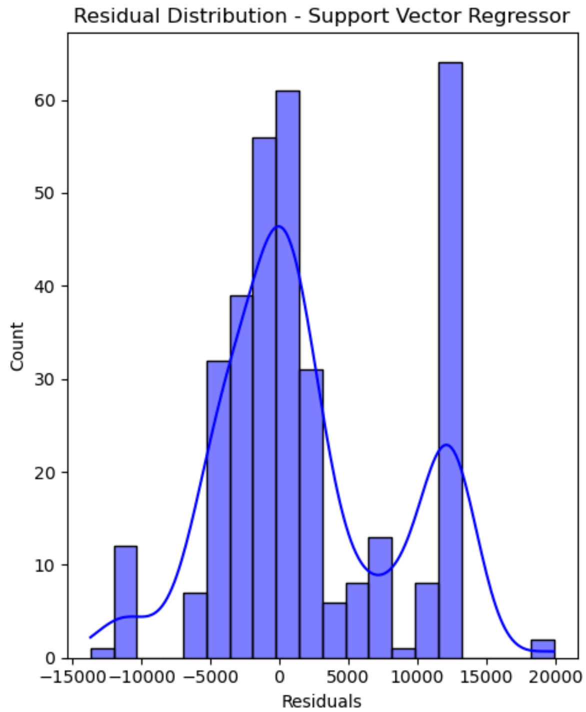


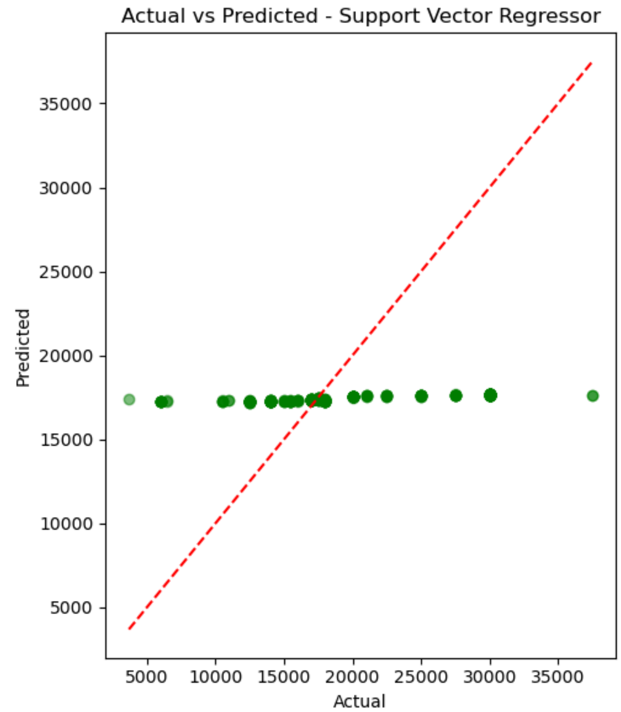

```
Model: Gradient Boosting
RMSE: 110.1928
MAE: 30.6401
R²: 0.9997
```

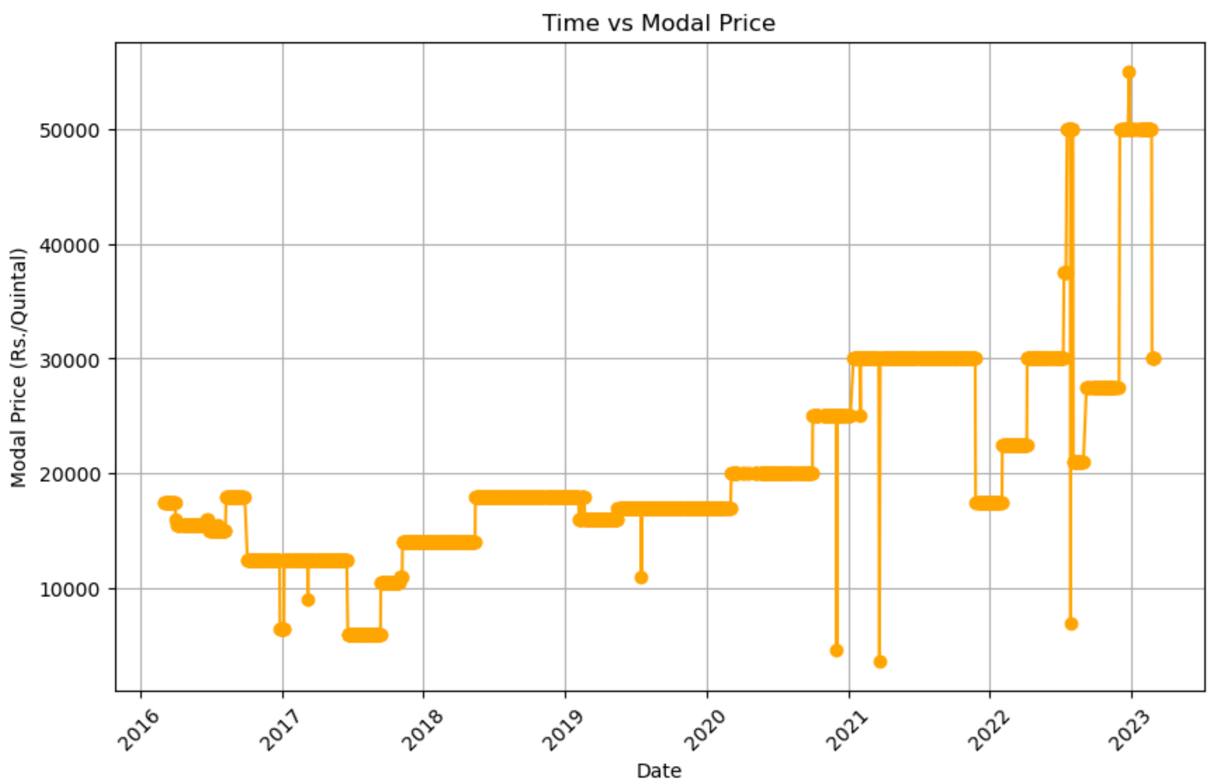





```
Model: Support Vector Regressor
RMSE: 6796.1336
MAE: 4963.1785
R²: -0.0463
```







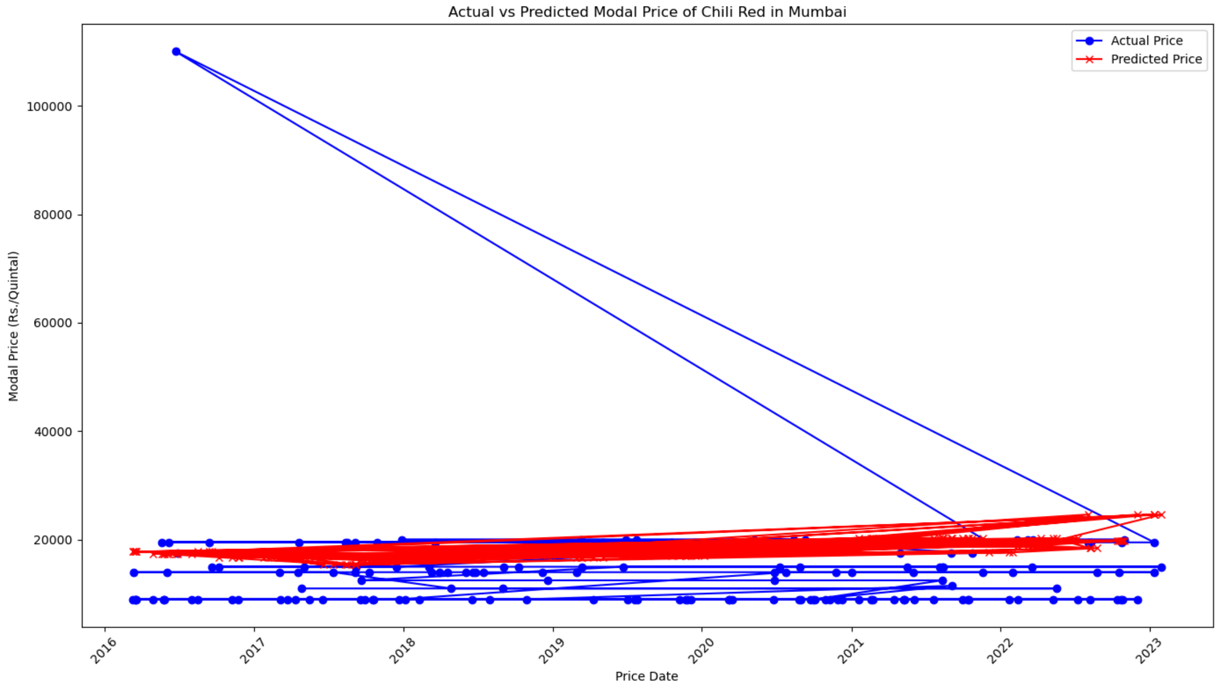
In [25]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

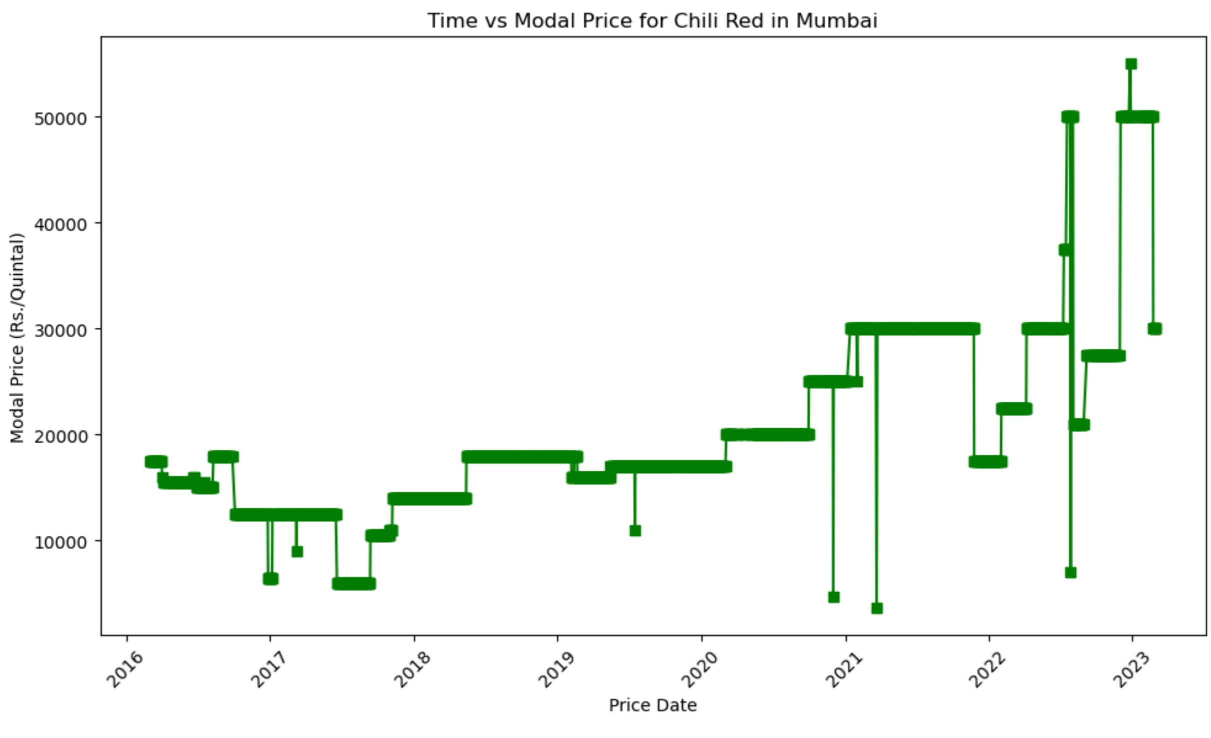
```
# Load the dataset
data = pd.read_csv(r"C:\Users\LENOVO\Downloads\dataset_red_chilli.csv")

# Convert 'Price Date' to datetime
data['Price Date'] = pd.to_datetime(data['Price Date'], errors='coerce')

# Drop any rows with NaT in 'Price Date' if they exist
data = data.dropna(subset=['Price Date'])

# Select features and target variable
X = data[['Min Price (Rs./Quintal)', 'Max Price (Rs./Quintal)']]
y = data['Modal Price (Rs./Quintal)']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, ran

# Initialize models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
    "Support Vector Regressor": SVR(kernel='linear')
}

# Train models and evaluate
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Model: {name}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE: {mae:.4f}")
    print(f"R²: {r2:.4f}\n")

# Load the actual price data for comparison
actual_price_data = pd.read_csv('C:/Users/LENOVO/Downloads/actual_price_red_chilli.
actual_price_data['Price Date'] = pd.to_datetime(actual_price_data['Price Date'], e

# Align actual price data with the test set based on dates
actual_price_data = actual_price_data.dropna(subset=['Price Date'])

# Ensure the test set dates and actual price dates align
test_dates = data['Price Date'][y_test.index].reset_index(drop=True)
actual_prices = actual_price_data['Modal Price (Rs./Quintal)'].reset_index(drop=Tru
predicted_prices = pd.Series(y_pred)

# Trim the actual_prices if it is longer than test dates
```

```python
min_len = min(len(test_dates), len(actual_prices), len(predicted_prices))
comparison_data = pd.DataFrame({
    'Price Date': test_dates[:min_len],
    'Actual Price': actual_prices[:min_len],
    'Predicted Price': predicted_prices[:min_len]
})

# Plot actual vs predicted modal prices over time
plt.figure(figsize=(14, 8))
plt.plot(comparison_data['Price Date'], comparison_data['Actual Price'], label='Act
plt.plot(comparison_data['Price Date'], comparison_data['Predicted Price'], label='
plt.xlabel('Price Date')
plt.ylabel('Modal Price (Rs./Quintal)')
plt.title('Actual vs Predicted Modal Price of Chili Red in Mumbai')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plot time vs modal price for the predicted data
plt.figure(figsize=(10, 6))
plt.plot(data['Price Date'], data['Modal Price (Rs./Quintal)'], label='Modal Price'
plt.xlabel('Price Date')
plt.ylabel('Modal Price (Rs./Quintal)')
plt.title('Time vs Modal Price for Chili Red in Mumbai')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_18988\334639999.py:16: UserWarning: Cou
ld not infer format, so each element will be parsed individually, falling back to `d
ateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  data['Price Date'] = pd.to_datetime(data['Price Date'], errors='coerce')
Model: Linear Regression
RMSE: 1246.4827
MAE: 1089.2014
R²: 0.9771

Model: Random Forest
RMSE: 328.4402
MAE: 129.6791
R²: 0.9984

Model: Gradient Boosting
RMSE: 331.9667
MAE: 151.7611
R²: 0.9984

Model: Support Vector Regressor
RMSE: 6688.0873
MAE: 4456.2045
R²: 0.3410
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_18988\334639999.py:56: UserWarning: Cou
ld not infer format, so each element will be parsed individually, falling back to `d
ateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  actual_price_data['Price Date'] = pd.to_datetime(actual_price_data['Price Date'],
errors='coerce')
```





In [ ]: