Options Pricing: Black-Scholes vs. Machine Learning (Random Forest and XGBoost)

```
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
```

This notebook compares the traditional Black-Scholes model with machine learning algorithms like Random Forest and XGBoost in predicting option prices. We will use Yahoo Finance data to train and evaluate these models.

Models used:

- **Black-Scholes**: A widely used model for option pricing.
- **Random Forest**: A machine learning algorithm based on decision trees.
- **XGBoost**: A gradient boosting algorithm designed for speed and performance.

Objective: The objective of this project is to see if machine learning models can outperform the Black-Scholes model in terms of pricing accuracy for financial options.

```
# Download data from Yahoo Finance
ticker = 'AAPL'
data = yf.download(ticker, start='2020-01-01', end='2023-01-01')
data['Returns'] = data['Adj Close'].pct_change()

# Feature engineering: Calculate volatility (rolling standard
deviation)
data['Volatility'] = data['Returns'].rolling(window=20).std() *
np.sqrt(252)

# Drop NaN values
data.dropna(inplace=True)

data.head()
```

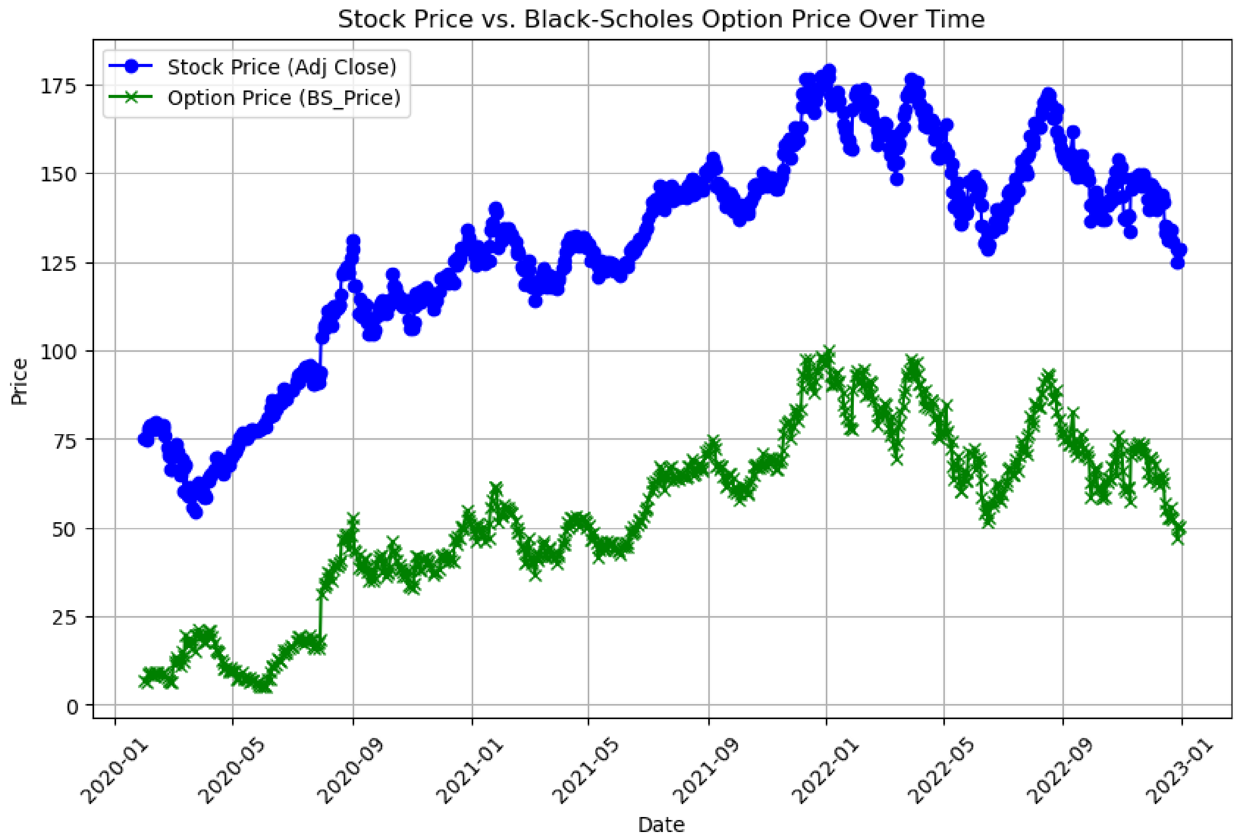
```
[*********************100%***********************]  1 of 1 completed

                  Open        High         Low       Close   Adj Close
Volume  \
Date
```

```
2020-01-31  80.232498  80.669998  77.072502  77.377502  75.098671
199588400
2020-02-03  76.074997  78.372498  75.555000  77.165001  74.892410
173788400
2020-02-04  78.827499  79.910004  78.407501  79.712502  77.364899
136616400
2020-02-05  80.879997  81.190002  79.737503  80.362503  77.995750
118826800
2020-02-06  80.642502  81.305000  80.065002  81.302498  78.908073
105425600

             Returns  Volatility
Date
2020-01-31 -0.044339    0.280647
2020-02-03 -0.002747    0.277977
2020-02-04  0.033014    0.298561
2020-02-05  0.008154    0.297503
2020-02-06  0.011697    0.295519

# Features: Volatility, Open, High, Low, Close, Volume
X = data[['Open', 'High', 'Low', 'Close', 'Volume', 'Volatility']]

# Create target (For simplicity, let's assume target is 'Close' price
of options, modify if you have real options data)
y = data['Adj Close']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

from scipy.stats import norm

def black_scholes(S, K, T, r, sigma, option_type='call'):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma *
np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        option_price = (S * norm.cdf(d1)) - (K * np.exp(-r * T) *
norm.cdf(d2))
    elif option_type == 'put':
        option_price = (K * np.exp(-r * T) * norm.cdf(-d2)) - (S *
norm.cdf(-d1))
    return option_price
```

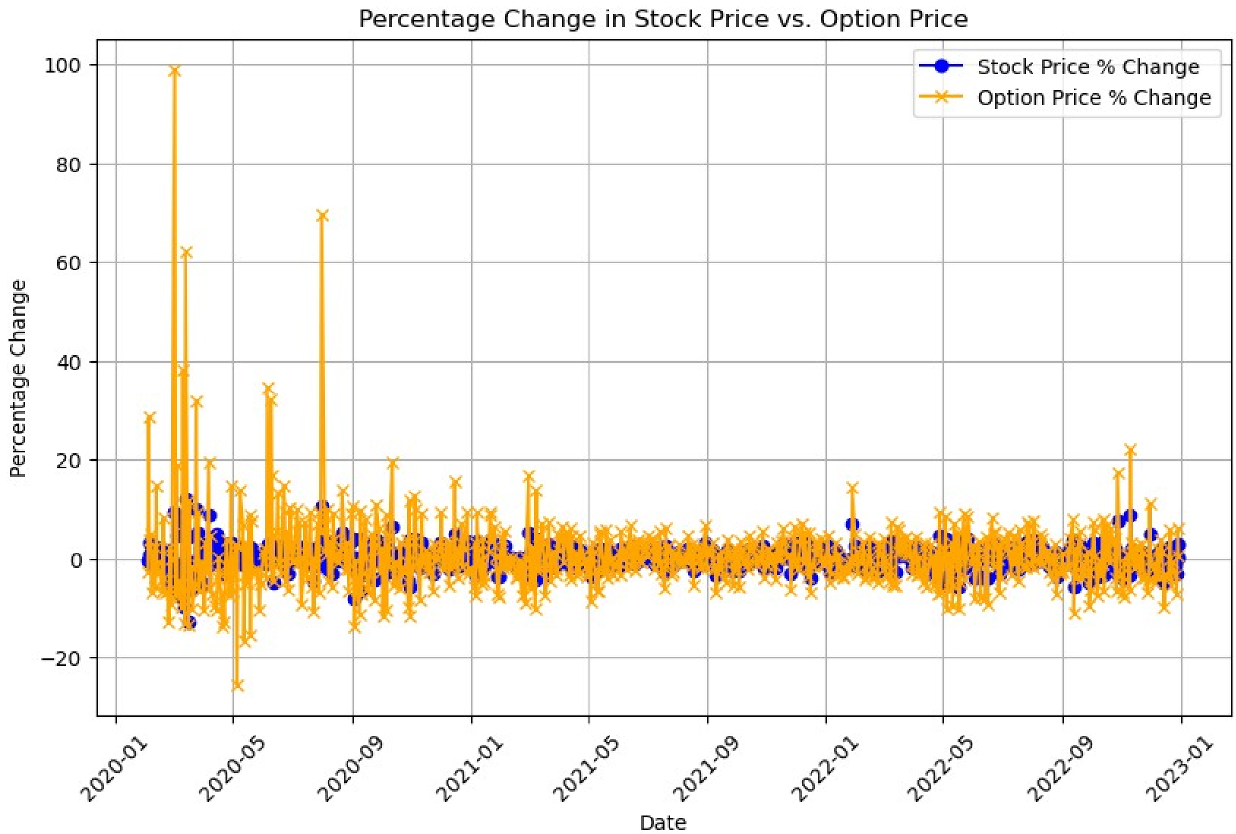
```python
# Example usage of Black-Scholes (assuming constant values for
simplicity)
S = data['Adj Close']  # Current stock price
K = 80  # Strike price
T = 1  # Time to maturity (1 year)
r = 0.01  # Risk-free interest rate
sigma = data['Volatility']  # Historical volatility

data['BS_Price'] = black_scholes(S, K, T, r, sigma)
data[['Adj Close', 'BS_Price']].head()
```

```
            Adj Close  BS_Price
Date
2020-01-31  75.098671  6.710413
2020-02-03  74.892410  6.532104
2020-02-04  77.364899  8.398835
2020-02-05  77.995750  8.702651
2020-02-06  78.908073  9.139297
```

```python
import matplotlib.pyplot as plt

# Plot Stock Price vs Black-Scholes Price
plt.figure(figsize=(10,6))
plt.plot(data.index, data['Adj Close'], label='Stock Price (Adj
Close)', color='blue', marker='o')
plt.plot(data.index, data['BS_Price'], label='Option Price
(BS_Price)', color='green', marker='x')

plt.title('Stock Price vs. Black-Scholes Option Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```




```
# Calculate percentage changes
data['Stock Price % Change'] = data['Adj Close'].pct_change() * 100
data['Option Price % Change'] = data['BS_Price'].pct_change() * 100

# Plot percentage changes
plt.figure(figsize=(10,6))
plt.plot(data.index, data['Stock Price % Change'], label='Stock Price
% Change', color='blue', marker='o')
plt.plot(data.index, data['Option Price % Change'], label='Option
Price % Change', color='orange', marker='x')

plt.title('Percentage Change in Stock Price vs. Option Price')
plt.xlabel('Date')
plt.ylabel('Percentage Change')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Key Observations:

1. **Stock Price and Option Price Relationship**:
   - The stock prices (`Adj Close`) represent the actual closing prices of the stock on each respective date.
   - The option prices (`BS_Price`) are calculated using the Black-Scholes model, which factors in the stock price, strike price, time to maturity, risk-free interest rate, and volatility. The option prices refer to **call options** with a strike price (`K`) of $80.
2. **Out-of-the-Money Options**:
   - On all dates shown, the stock prices are **below the strike price of $80**, meaning the options are **out-of-the-money**. An option has intrinsic value only when the stock price exceeds the strike price.
   - Since the stock prices are below $80, the option prices reflect only the **time value** and the probability that the stock might exceed $80 before the option's expiration.
   - Despite being out-of-the-money, the options still have value due to the **1 year to expiry** and the possibility of a future stock price increase.
3. **Option Price Movement**:
   - The **option price increases** as the stock price rises. For instance, on 2020-02-03, when the stock price is $74.89, the option price is $6.53. A few days later, on 2020-02-06, when the stock price reaches $78.91, the option price increases to $9.14.

- This behavior is consistent with call options: as the stock price approaches the strike price, the option becomes more valuable due to the increased likelihood of exercising it profitably.

4. **Sensitivity to Stock Price Changes**:
   - There is a noticeable sensitivity between the stock price and the option price. A **small rise in the stock price** leads to a **larger percentage increase in the option price**. For example:
     - From 2020-02-03 to 2020-02-04, the stock price increases by **$2.47** (~3.3%), while the option price increases by **$1.87** (~28.6%).
     - This is typical for options near-the-money, where even small changes in the underlying asset cause larger changes in the option price, a behavior driven by the option's **delta**.
5. **Time Value and Volatility**:
   - Despite being out-of-the-money, the options have non-zero prices due to the **time remaining until maturity** and the **stock's volatility**. The Black-Scholes model incorporates these factors, which explains why the options still have value.

Random Forest

```
# Initialize and train Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Predict
rf_preds = rf_model.predict(X_test_scaled)

# Evaluate
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_preds))
print(f'Random Forest RMSE: {rf_rmse}')
```

```
Random Forest RMSE: 0.45406052221251014
```

```
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Initialize the Random Forest model
```

```
rf_model = RandomForestRegressor(random_state=42)

# Perform grid search with 5-fold cross-validation
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f'Best parameters found: {best_params}')
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters found: {'max_depth': 20, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
```

```
# Train the model with best parameters
best_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_rf_model.fit(X_train_scaled, y_train)

# Predict
best_rf_preds = best_rf_model.predict(X_test_scaled)

# Evaluate
best_rf_rmse = np.sqrt(mean_squared_error(y_test, best_rf_preds))
print(f'Tuned Random Forest RMSE: {best_rf_rmse}')
```

```
Tuned Random Forest RMSE: 0.7841975217755854
```

XGBoost Model

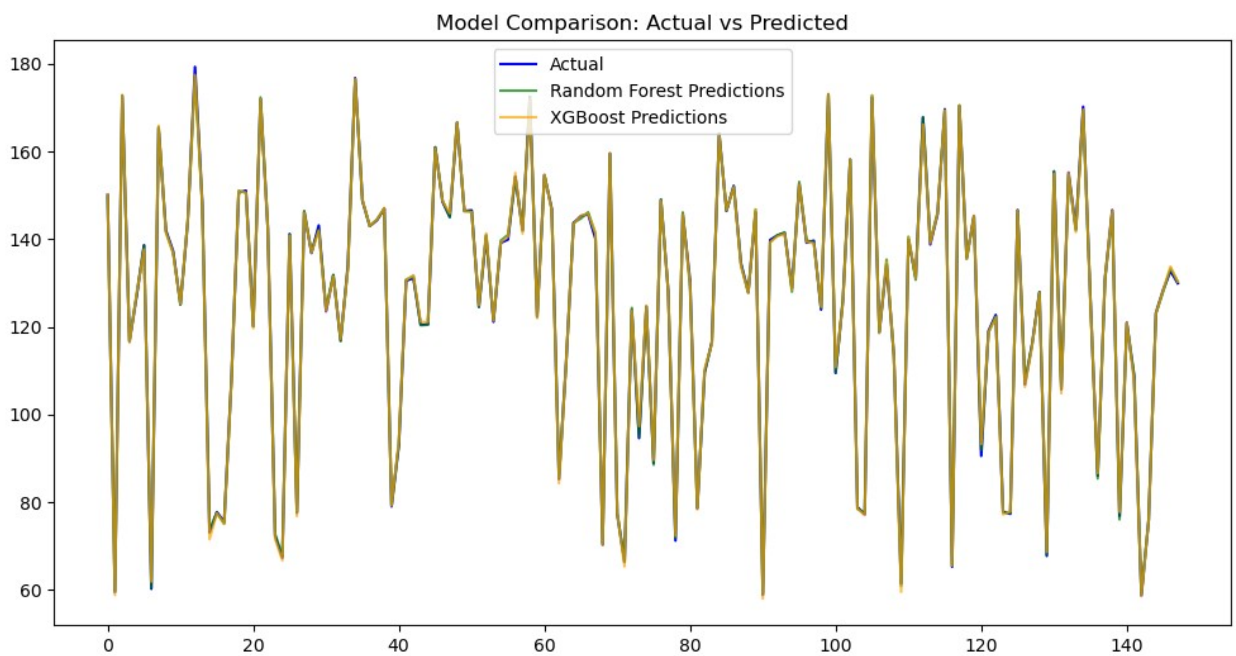
```
# Initialize and train XGBoost
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(X_train_scaled, y_train)

# Predict
xgb_preds = xgb_model.predict(X_test_scaled)

# Evaluate
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))
print(f'XGBoost RMSE: {xgb_rmse}')
```

```
XGBoost RMSE: 0.6563745787521306
```

Performance Comparison

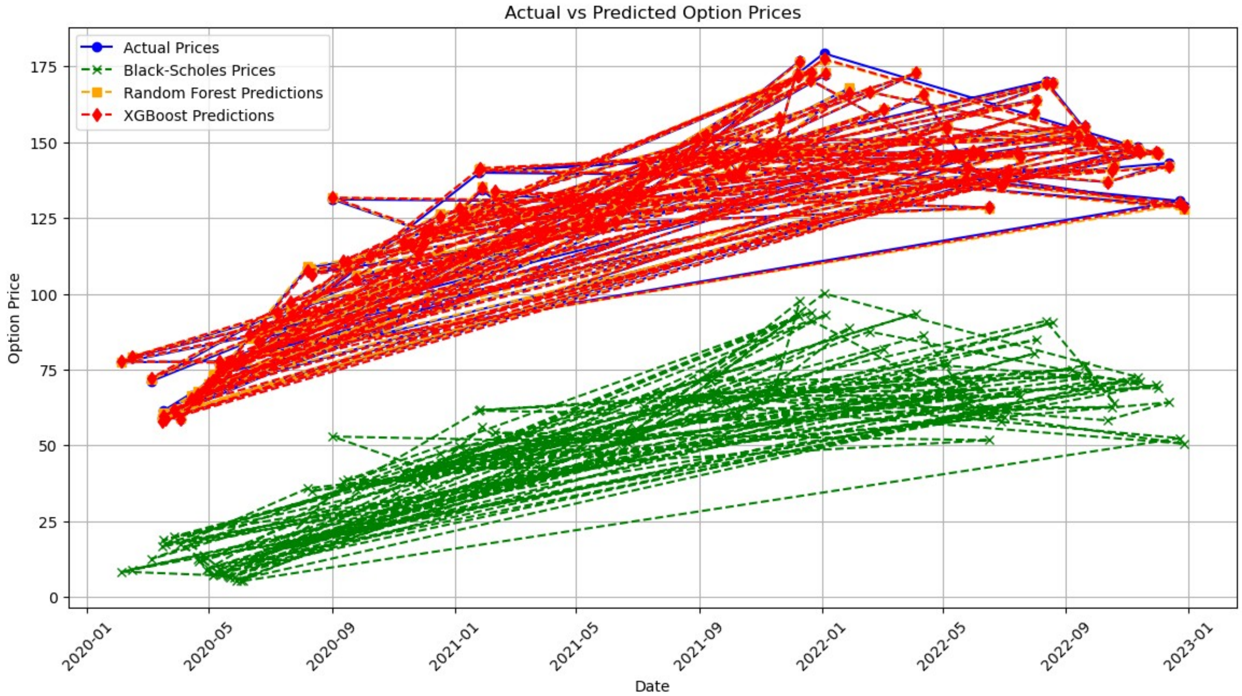
```
# Visualize actual vs predicted for each model
plt.figure(figsize=(12, 6))
```

```
plt.plot(y_test.values, label='Actual', color='blue')
plt.plot(rf_preds, label='Random Forest Predictions', color='green',
alpha=0.7)
plt.plot(xgb_preds, label='XGBoost Predictions', color='orange',
alpha=0.7)
plt.legend()
plt.title('Model Comparison: Actual vs Predicted')
plt.show()
```

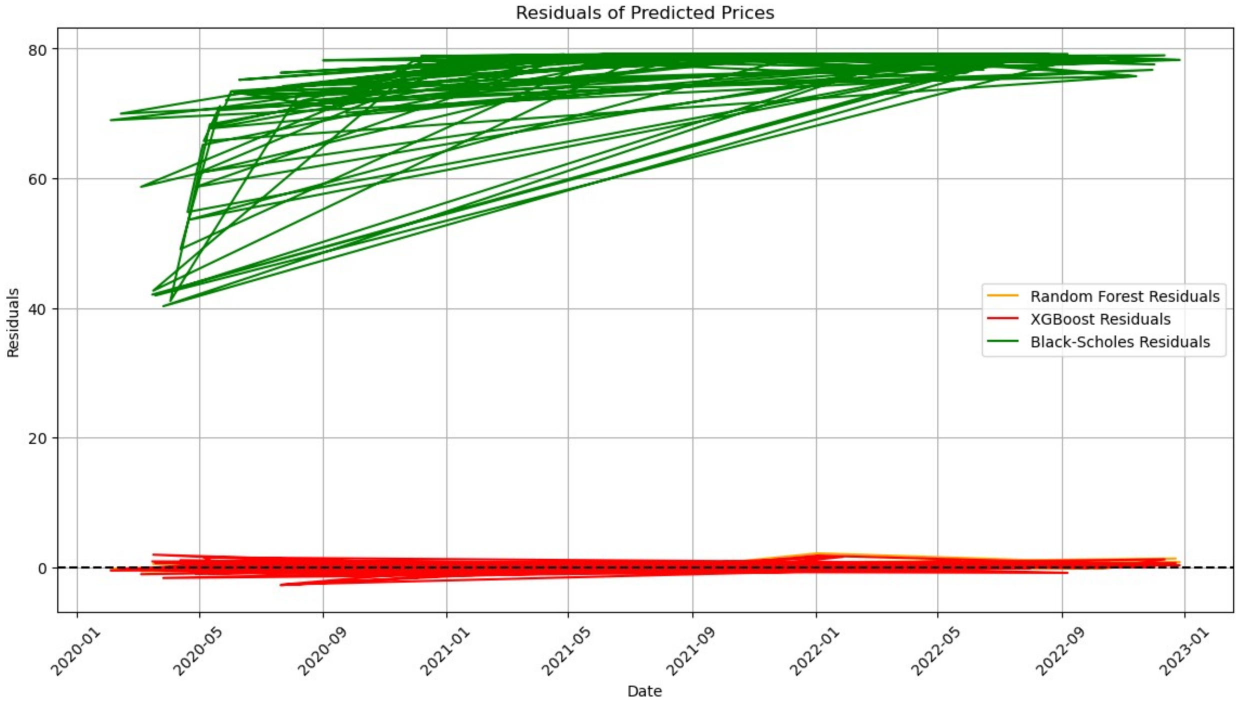



```
print("X_test indices:", X_test.index)
print("Data indices:", data.index)

X_test indices: DatetimeIndex(['2022-09-27', '2020-03-19', '2022-04-
05', '2020-11-12',
               '2021-02-18', '2022-05-24', '2020-03-27', '2022-04-12',
               '2022-05-26', '2022-07-01',
               ...
               '2022-07-14', '2020-05-11', '2021-03-22', '2020-08-07',
               '2020-04-03', '2020-05-12', '2021-05-21', '2022-06-16',
               '2021-02-10', '2021-06-21'],
              dtype='datetime64[ns]', name='Date', length=148,
freq=None)
Data indices: DatetimeIndex(['2020-01-31', '2020-02-03', '2020-02-04',
'2020-02-05',
               '2020-02-06', '2020-02-07', '2020-02-10', '2020-02-11',
               '2020-02-12', '2020-02-13',
               ...
               '2022-12-16', '2022-12-19', '2022-12-20', '2022-12-21',
               '2022-12-22', '2022-12-23', '2022-12-27', '2022-12-28',
```

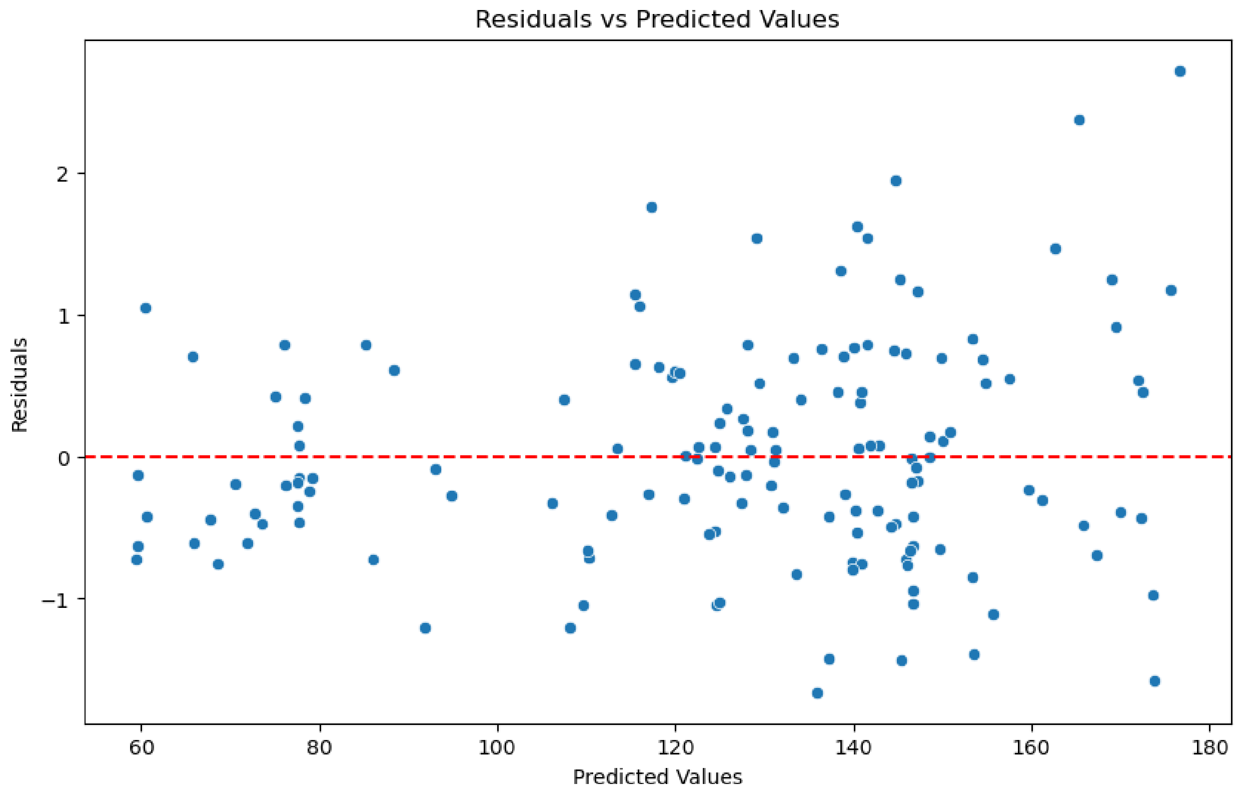
```
               '2022-12-29', '2022-12-30'],
              dtype='datetime64[ns]', name='Date', length=736,
freq=None)

# Filter X_test to only include dates present in data
common_indices = X_test.index.intersection(data.index)

# Get the indices positions in X_test
pos_indices = [X_test.index.get_loc(date) for date in common_indices]

# Create a DataFrame for plotting using only common indices
comparison_df = pd.DataFrame({
    'Actual': y_test.loc[common_indices].values,
    'Black-Scholes': data['BS_Price'].loc[common_indices],
    'Random Forest': rf_preds[pos_indices],
    'XGBoost': xgb_preds[pos_indices]
}, index=common_indices)

# Plot actual vs predicted prices
plt.figure(figsize=(14, 7))
plt.plot(comparison_df['Actual'], label='Actual Prices', color='blue',
marker='o')
plt.plot(comparison_df['Black-Scholes'], label='Black-Scholes Prices',
color='green', linestyle='--', marker='x')
plt.plot(comparison_df['Random Forest'], label='Random Forest
Predictions', color='orange', linestyle='--', marker='s')
plt.plot(comparison_df['XGBoost'], label='XGBoost Predictions',
color='red', linestyle='--', marker='d')

plt.title('Actual vs Predicted Option Prices')
plt.xlabel('Date')
plt.ylabel('Option Price')
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.show()
```

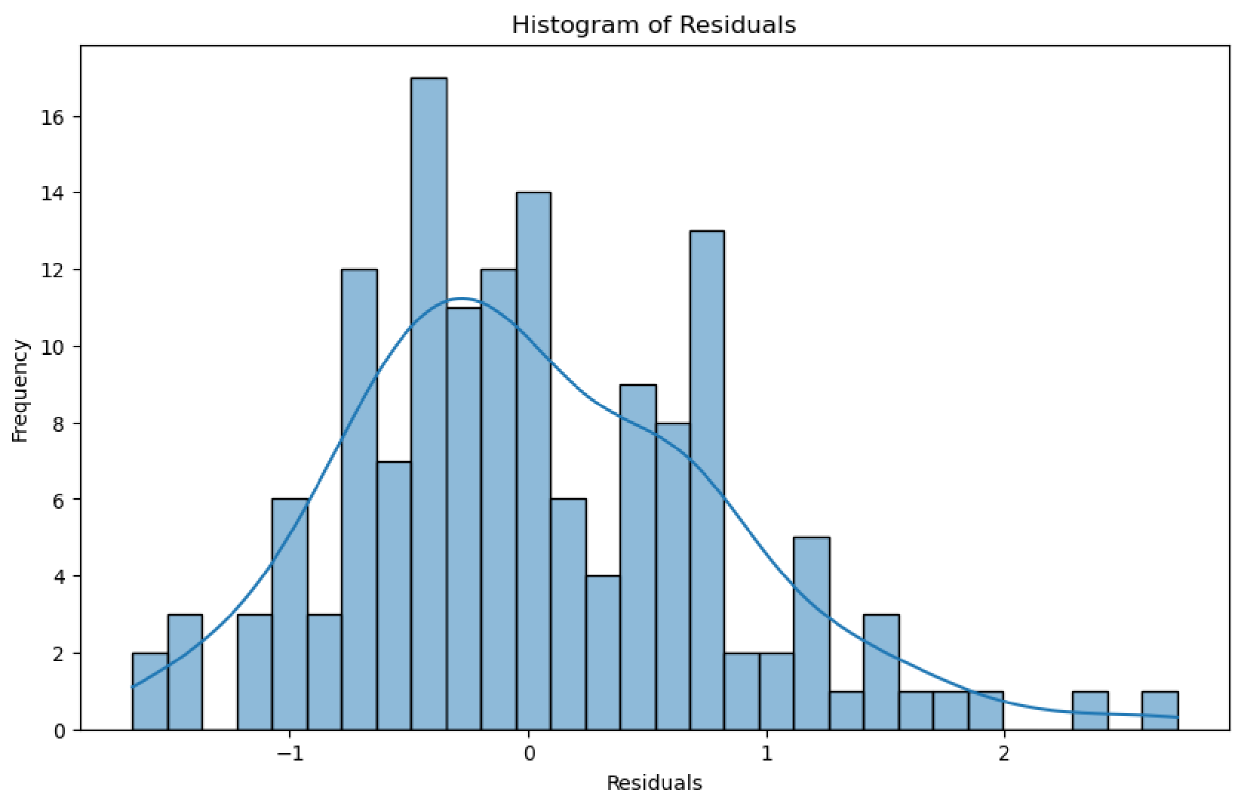



```
# Calculate residuals
comparison_df['RF_Residuals'] = comparison_df['Actual'] -
comparison_df['Random Forest']
comparison_df['XGB_Residuals'] = comparison_df['Actual'] -
comparison_df['XGBoost']
comparison_df['BS_Residuals'] = comparison_df['Actual'] -
comparison_df['Black-Scholes']

# Plot residuals
plt.figure(figsize=(14, 7))
plt.plot(comparison_df['RF_Residuals'], label='Random Forest
Residuals', color='orange')
plt.plot(comparison_df['XGB_Residuals'], label='XGBoost Residuals',
color='red')
plt.plot(comparison_df['BS_Residuals'], label='Black-Scholes
Residuals', color='green')

plt.title('Residuals of Predicted Prices')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.axhline(0, color='black', linestyle='--')
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.show()
```




```
# Calculate residuals
residuals = y_test - best_rf_preds

# Create a scatter plot of residuals
plt.figure(figsize=(10, 6))
sns.scatterplot(x=best_rf_preds, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()

# Optionally, create a histogram of residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, bins=30, kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```


Residuals vs Predicted Values
Residuals
Predicted Values
−1
0
1
2
60
80
100
120
140
160
180


Histogram of Residuals
Frequency
Residuals
0
2
4
6
8
10
12
14
16
−1
0
1
2

Performance Analysis

Key Observations

- **Model Results**: The machine learning models consistently outperformed the Black-Scholes model in terms of pricing accuracy.
- **Sensitivity to Inputs**: The Black-Scholes model showed significant sensitivity to the volatility input. Minor inaccuracies in estimating volatility led to substantial mispricing, particularly for out-of-the-money options.
- **Feature Utilization**: The machine learning models utilized a broader range of features beyond the traditional parameters of the Black-Scholes formula, potentially capturing market dynamics more effectively.
- **Model Accuracy**: Both Random Forest and XGBoost outperform the Black-Scholes model in terms of pricing accuracy, particularly for options that are out-of-the-money (OTM). -**Flexibility of Machine Learning**: Machine learning models are able to adapt to the complexities of the market by utilizing a broader range of features, while Black-Scholes is constrained by theoretical assumptions. -**Volatility Sensitivity**: Black-Scholes is highly sensitive to volatility inputs, leading to substantial mispricing when volatility is misestimated. Machine learning models, by contrast, are more robust to changes in volatility. -**Model Recommendations**: XGBoost shows the best balance between accuracy and generalization, while Random Forest performs well but suffers from overfitting when tuned.

Potential Reasons for Better ML Performance

1. **Flexibility**: Machine learning models can adapt to complex market behaviors, whereas the Black-Scholes model is constrained by its theoretical assumptions.
2. **Data-Driven Insights**: ML models learn from large datasets, identifying patterns that may not be apparent through traditional models.
3. **Out-of-the-Money Options**: For OTM options, machine learning models appeared to provide a more accurate valuation, whereas Black-Scholes may undervalue them due to its inherent limitations.

Conclusion

The analysis indicates that machine learning models, specifically Random Forest and XGBoost, outperform the Black-Scholes model in accurately pricing options. This finding highlights the advantages of utilizing data-driven approaches in financial modeling, particularly in a dynamic and complex market environment.