In [82]:
```
import pandas as pd
import numpy as np
import os
import yfinance as yf
from datetime import timedelta
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [83]:
```
SPX_Prices = yf.download('SPY', start='2002-01-01', end='2024-01-01', interval = "1d")
SPX_Prices
```

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

Out[83]:

Date	Open	High	Low	Close	Adj Close	Volume
2002-01-02	115.110001	115.750000	113.809998	115.529999	76.087868	18651900
2002-01-03	115.650002	116.949997	115.540001	116.839996	76.950668	15743000
2002-01-04	117.169998	117.980003	116.550003	117.620003	77.464363	20140700
2002-01-07	117.699997	117.989998	116.559998	116.790001	76.917740	13106500
2002-01-08	116.790001	117.059998	115.970001	116.519997	76.739929	12683700
...	...	...	...	...	...	...
2023-12-22	473.859985	475.380005	471.700012	473.649994	472.182892	67126600
2023-12-26	474.070007	476.579987	473.989990	475.649994	474.176697	55387000
2023-12-27	475.440002	476.660004	474.890015	476.510010	475.034058	68000300
2023-12-28	476.880005	477.549988	476.260010	476.690002	475.213501	77158100
2023-12-29	476.489990	477.029999	473.299988	475.309998	473.837769	122234100

5537 rows × 6 columns

In [84]:
```
VIX = yf.download('^VIX', start='2002-01-01', end='2024-01-01', interval = "1d")
VIX
```

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

Out[84]:

Date	Open	High	Low	Close	Adj Close	Volume
2002-01-02	23.780001	24.200001	22.709999	22.709999	22.709999	0
2002-01-03	22.219999	22.430000	21.330000	21.340000	21.340000	0
2002-01-04	20.969999	21.530001	20.400000	20.450001	20.450001	0
2002-01-07	21.410000	22.150000	21.350000	21.940001	21.940001	0
2002-01-08	21.629999	22.290001	21.280001	21.830000	21.830000	0
...	...	...	...	...	...	...
2023-12-22	13.720000	13.960000	13.000000	13.030000	13.030000	0
2023-12-26	13.770000	13.800000	12.960000	12.990000	12.990000	0
2023-12-27	13.020000	13.040000	12.370000	12.430000	12.430000	0
2023-12-28	12.440000	12.650000	12.380000	12.470000	12.470000	0

2023-12-29	12.550000	13.190000	12.360000	12.450000	12.450000	0

5537 rows × 6 columns

In [85]:
```python
def garman_klass_daily_variance(data):
    """
    Calculate daily Garman-Klass variance for given price data.
    """
    log_hl = np.log(data['High'] / data['Low'])
    log_co = np.log(data['Close'] / data['Open'])
    daily_variance = 0.5 * log_hl**2 - (2 * np.log(2) - 1) * log_co**2
    return daily_variance
```

In [86]:
```python
def rolling_volatility(data, rolling_window):
    """
    Calculate annualized Garman-Klass volatility over a given period
    """
    Daily_Volatility = garman_klass_daily_variance(data)
    Rolling_Vol = np.sqrt((Daily_Volatility.rolling(rolling_window).mean())*252)
    return Rolling_Vol
```

In [87]:
```python
daily_vol = rolling_volatility(SPX_Prices, 1)
weekly_vol = rolling_volatility(SPX_Prices, 5)
monthly_vol = rolling_volatility(SPX_Prices, 21)
quartely_vol = rolling_volatility(SPX_Prices, 63)

# Here we remove the NaN values from the volatility metrics
weekly_vol_nan = (weekly_vol[~np.isnan(weekly_vol)])
monthly_vol_nan = (monthly_vol[~np.isnan(monthly_vol)])
quartely_vol_nan = (quartely_vol[~np.isnan(quartely_vol)])
# We go as far back as the minimum of the amount of valid data points
far_back = min(len(weekly_vol_nan) , len(monthly_vol_nan) , len(quartely_vol_nan))
# Adjust the data to be the same size
weekly_vol_nan = weekly_vol_nan[-far_back:]
monthly_vol_nan = monthly_vol_nan[-far_back:]
quartely_vol_nan = quartely_vol_nan[-far_back:]
daily_vol_nan = daily_vol[-far_back:]
volatilities = pd.DataFrame(
    data = {"daily_vol" : daily_vol_nan,
            "weekly_vol" : weekly_vol_nan,
            "monthly_vol" : monthly_vol_nan,
            "quartely_vol" : quartely_vol_nan
           },
    index = (SPX_Prices.index))

volatilities["next_day"] = volatilities["daily_vol"].shift(-1)
volatilities = volatilities.dropna()
volatilities
```

Out[87]:

	daily_vol	weekly_vol	monthly_vol	quartely_vol	next_day
Date					
2002-04-03	0.188741	0.141114	0.129831	0.156948	0.116253
2002-04-04	0.116253	0.141567	0.128777	0.155872	0.137383
2002-04-05	0.137383	0.145083	0.128437	0.156408	0.111350
2002-04-08	0.111350	0.136978	0.125265	0.156159	0.085117
2002-04-09	0.085117	0.132406	0.123316	0.155873	0.087609
...	...	...	...	...	...

2023-12-21	0.094439	0.093619	0.078902	0.101646	0.087123
2023-12-22	0.087123	0.095185	0.080046	0.101953	0.051613
2023-12-26	0.051613	0.092716	0.080736	0.101349	0.035382
2023-12-27	0.035382	0.092868	0.080752	0.099901	0.030107
2023-12-28	0.030107	0.065317	0.079735	0.098773	0.084652

5474 rows × 5 columns

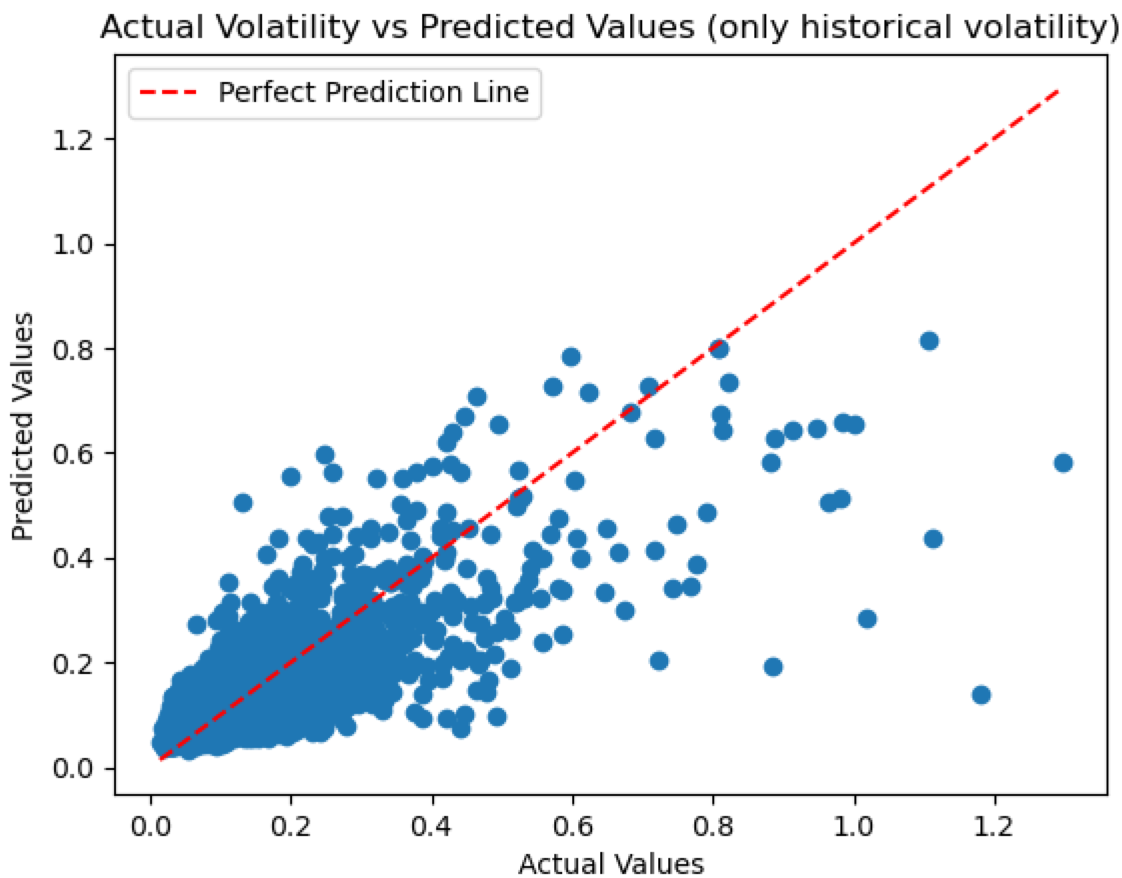
In [88]:

```
GDP = pd.read_csv('/Users/henryhartwell/Downloads/GDPC1.csv')

GDP["GDP_Change"] = GDP["GDPC1"].pct_change()
GDP.dropna(inplace=True)

GDP.set_index("DATE", inplace=True)
GDP.index = pd.to_datetime(GDP.index)


date_range = pd.date_range(start=GDP.index.min(), end = GDP.index.max(), freq='D')
GDP_Extended = GDP.reindex(date_range, method='ffill')

vol_GDP = pd.merge(volatilities, GDP_Extended, how="left", left_index=True, right_index=
vol_GDP.dropna(axis=0, inplace=True)

GDP_Change = vol_GDP["GDP_Change"]
weekly_vol_nan = vol_GDP["weekly_vol"]
monthly_vol_nan = vol_GDP["monthly_vol"]
quartely_vol_nan = vol_GDP["quartely_vol"]
daily_vol_nan = vol_GDP["daily_vol"]
next_day_vol_nan = vol_GDP["next_day"]
```

In [96]:

```
### This is our regression model with only historical volatility as our independent vari

# Data with three independent variables (X1, X2, X3) and one dependent variable (Y)
All_Vols = {'X1': daily_vol_nan,
        'X2': weekly_vol_nan ,
        'X3': monthly_vol_nan,
        'Y': next_day_vol_nan}



df = pd.DataFrame(All_Vols)

# Separate independent variables (features) and dependent variable
X = df[['X1', 'X2', 'X3']]
y = df['Y']

# Create and fit the multiple regression model
all_preds = LinearRegression()
all_preds.fit(X, y)

# Predictions
y_pred = all_preds.predict(X)

# Plotting the actual vs predicted values
plt.scatter(y, y_pred)
plt.plot([min(y), max(y)], [min(y), max(y)], linestyle='--', color='red', label='Perfect
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual Volatility vs Predicted Values (only historical volatility)')
plt.legend()
```

```
plt.show()

# Finding the values of R-squared and Adjusted R-squared
r_squared = r2_score(y , y_pred)
observations = len(y) # number of observations
predictors = len(All_Vols) - 1 #number of predictors
adj_r_squared = 1 - ((1-r_squared)*((observations-1)/(observations-predictors-1)))


# Print the coefficients (slope) and intercept
print('Coefficients (Slope):', all_preds.coef_)
print('Intercept:', all_preds.intercept_)
print('R-squared:' , r_squared)
print('Adjusted R-squared:' , adj_r_squared)
```

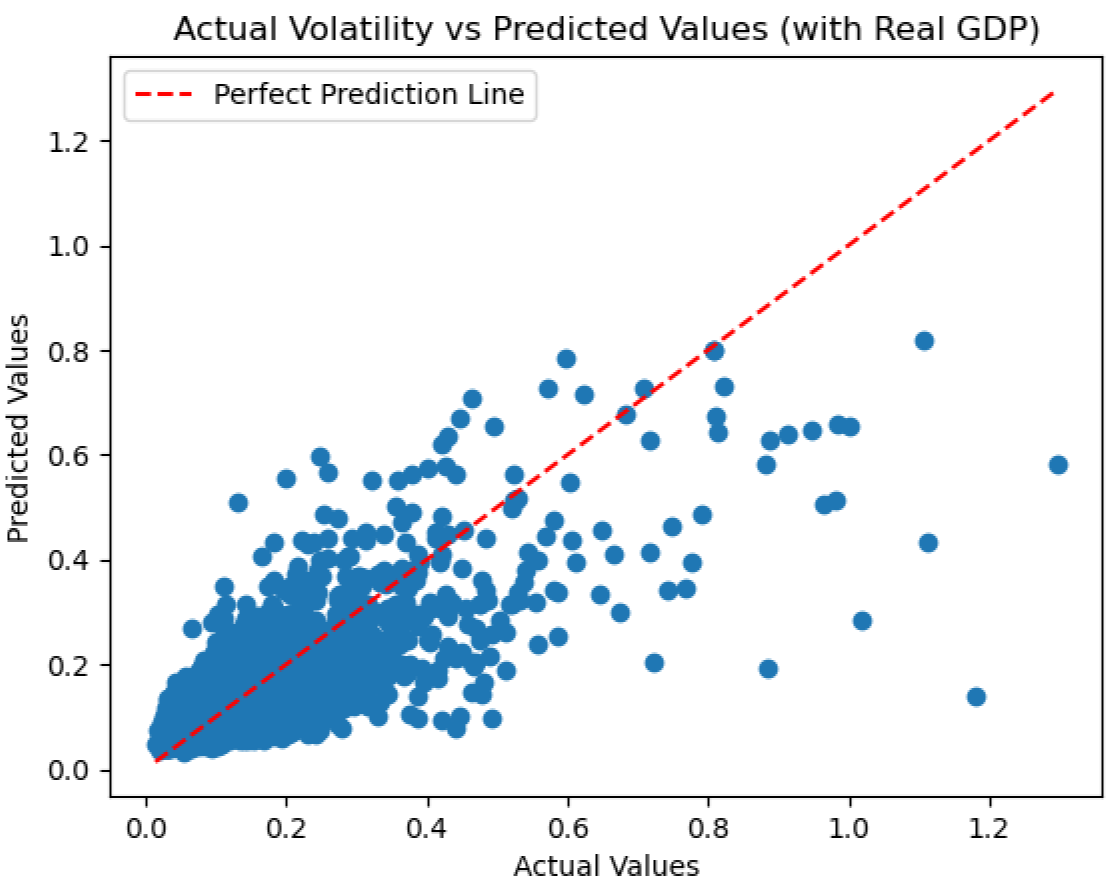



```
Coefficients (Slope): [0.30030643 0.34908524 0.23523748]
Intercept: 0.009788757557380026
R-squared: 0.6020835254211909
Adjusted R-squared: 0.6018627877318905
```

In [95]:

```
### This is our regression model with historical volatility and Real GDP as our independ

# Data with four independent variables (X1, X2, X3, X4) and one dependent variable (Y)
All_Vols = {'X1': daily_vol_nan,
        'X2': weekly_vol_nan ,
        'X3': monthly_vol_nan,
        'X4': GDP_Change,
        'Y': next_day_vol_nan}

df = pd.DataFrame(All_Vols)

# Separate independent variables (features) and dependent variable
X = df[['X1', 'X2', 'X3', 'X4']]
y = df['Y']

# Create and fit the multiple regression model
all_preds = LinearRegression()
```

```python
all_preds.fit(X, y)

# Predictions
y_pred = all_preds.predict(X)

# Plotting the actual vs predicted values
plt.scatter(y, y_pred)
plt.plot([min(y), max(y)], [min(y), max(y)], linestyle='--', color='red', label='Perfect
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual Volatility vs Predicted Values (with Real GDP)')
plt.legend()
plt.show()

# Print the coefficients (slope) and intercept
print('Coefficients (Slope):', all_preds.coef_)
print('Intercept:', all_preds.intercept_)
print('R-squared:' , r_squared)
print('Adjusted R-squared:' , adj_r_squared)
```




```
Coefficients (Slope): [ 0.29992403  0.35044077  0.22840138 -0.10081716]
Intercept: 0.011120417371729807
R-squared: 0.6020835254211909
Adjusted R-squared: 0.6018627877318905
```

In [ ]: