Raghav - Regression Model with CPI Data

March 17, 2024

[1]:
```
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
```

[ ]:
```

```

[2]:
```
SPX_Prices = yf.download('SPY', start='2002-01-01', end='2024-01-01', interval␣
↪= "1d")
SPX_Prices
```

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

[2]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2002-01-02  115.110001  115.750000  113.809998  115.529999   76.087906
2002-01-03  115.650002  116.949997  115.540001  116.839996   76.950645
2002-01-04  117.169998  117.980003  116.550003  117.620003   77.464355
2002-01-07  117.699997  117.989998  116.559998  116.790001   76.917725
2002-01-08  116.790001  117.059998  115.970001  116.519997   76.739883
…                  …           …           …           …           …
2023-12-22  473.859985  475.380005  471.700012  473.649994  472.182892
2023-12-26  474.070007  476.579987  473.989990  475.649994  474.176697
2023-12-27  475.440002  476.660004  474.890015  476.510010  475.034058
2023-12-28  476.880005  477.549988  476.260010  476.690002  475.213501
2023-12-29  476.489990  477.029999  473.299988  475.309998  473.837769

               Volume
Date
2002-01-02   18651900
2002-01-03   15743000
2002-01-04   20140700
2002-01-07   13106500
2002-01-08   12683700
```

```
…               …
2023-12-22   67126600
2023-12-26   55387000
2023-12-27   68000300
2023-12-28   77158100
2023-12-29  122234100

[5537 rows x 6 columns]
```

[3]:
```python
VIX = yf.download('^VIX', start='2002-01-01', end='2024-01-01', interval = "1d")
VIX
```

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

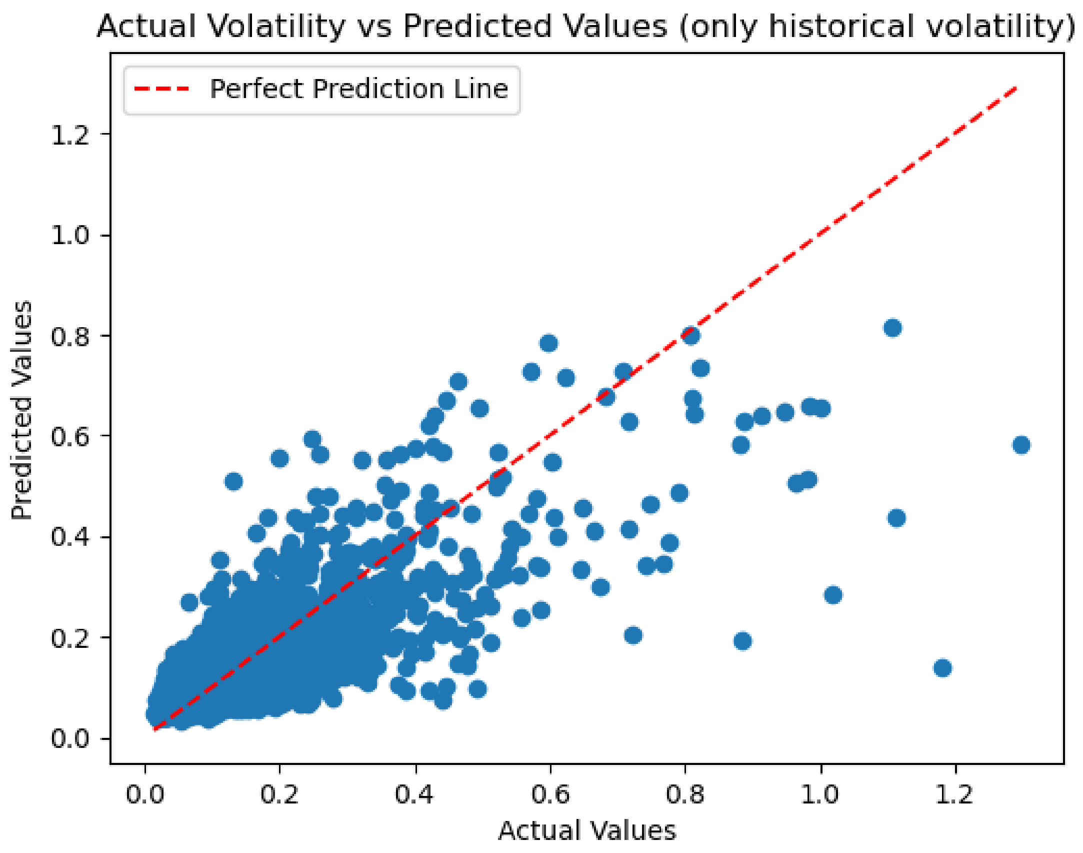
[3]:
```
                 Open       High        Low      Close  Adj Close  Volume
Date
2002-01-02  23.780001  24.200001  22.709999  22.709999  22.709999       0
2002-01-03  22.219999  22.430000  21.330000  21.340000  21.340000       0
2002-01-04  20.969999  21.530001  20.400000  20.450001  20.450001       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 x 6 columns]
```

[4]:
```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
```

[5]:
```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
```

[6]:
```
daily_vol = rolling_volatility(SPX_Prices, 1)
next_day = daily_vol.shift(-1)
weekly_vol = rolling_volatility(SPX_Prices, 5)
monthly_vol = rolling_volatility(SPX_Prices, 21)
quartely_vol = rolling_volatility(SPX_Prices, 63)
volatilities = pd.DataFrame(
    data = {"daily_vol" : daily_vol,
            "weekly_vol" : weekly_vol,
            "monthly_vol" : monthly_vol,
            "quartely_vol" : quartely_vol,
            "next_day" : next_day
           },
    index = (SPX_Prices.index))
# Here we remove the NaN values from the volatility metrics
volatilities.dropna(inplace=True)
far_back = len(volatilities)
```

[7]:
```
### This is our regression model with only historical volatility as our␣
 ↪independent variables ###

# Data with three independent variables (X1, X2, X3) and one dependent variable␣
 ↪(Y)
All_Vols = {'X1': volatilities['daily_vol'],
        'X2': volatilities['weekly_vol'],
        'X3': volatilities['monthly_vol'],
        'Y': volatilities['next_day']}

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 Prediction Line')
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)
```

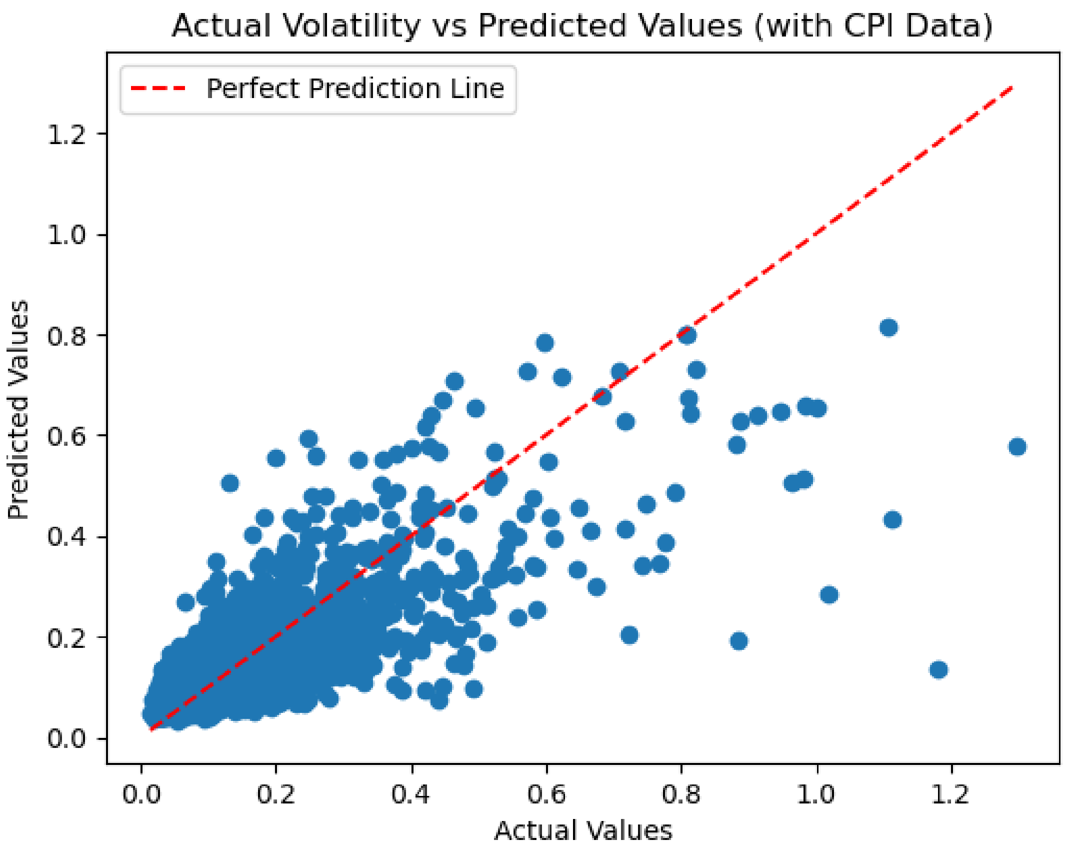


```
Coefficients (Slope): [0.2996955  0.34984096 0.2351574 ]
Intercept: 0.009734076213321813
R-squared: 0.6019867619238796
Adjusted R-squared: 0.6017684731278597
```

[12]:
```
# Import the CPI data from FRED
CPI_Data = pd.read_csv('/Users/ganeshthondikulam/Downloads/Median CPI Data␣
 ↪(Daily).csv')
# Make the data frame the same size as the other data
CPI_Data_df = pd.DataFrame(CPI_Data[-far_back:])
# make the data frame a series so it can be added to the other data
CPI_Data_series = pd.Series(CPI_Data_df['CPIRATE'])
# Change the indicies to the same as the other data
CPI_Data_series.index = df.index
```

[15]:
```
### This is our regression model with historical volatility and CPI Data as our␣
 ↪independent variables ###

# Data with four independent variables (X1, X2, X3, X4) and one dependent␣
 ↪variable (Y)
All_Vols = {'X1': volatilities['daily_vol'],
        'X2': volatilities['weekly_vol'],
        'X3': volatilities['monthly_vol'],
        'X4': CPI_Data_series,
        'Y': volatilities['next_day']}

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()
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 Prediction Line')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual Volatility vs Predicted Values (with CPI Data)')
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): [0.29941964 0.34936615 0.2363022  0.0007142 ]
Intercept: 0.007729433442559552
R-squared: 0.6020899545233773
Adjusted R-squared: 0.6017989250514617
```

[ ]: