Volatility Predictions with Python

Lars Nussberger, 14-611-412

November 9, 2018

1 Introduction

I want to predict volatility using different models and compare the performance. I first set up different Autoregressive models. Then I load the stock data of Amazon, calculate the Garman Klass Volatility and perform some data analysis. I then predict the out of sample volatility and compare the model performance. I use Autoregressive models and a Heterogenious Autoregressive model.

The Autoregressive model AR(P) can be described as:

$$x_t = a_0 + \sum_{i=1}^{P} a_i x_{t-i} + \omega_t$$

The Heterogenious Autoregressive model HAR(1,5,22) is:

$$x_t = a_0 + x_{t-1}^{(1)} + x_{t-1}^{(5)} + x_{t-1}^{(22)} + \omega_t \text{ , where}$$
$$x_t^{(n)} = \frac{1}{n} \sum_{j=1}^{n} x_{t-j+1}$$

The HAR(1,5,22) is basically a restricted AR(22) model (for more information Corsi 2009 [1]).

The models can be estimated using Ordinary Least Square (OLS). I the out of sample predictions compare a AR(3), HAR(1,5,22), AR(22) model.

However since the AR(22) model includes 22 estimators it becomes vonurable to overfitting. I additionally perform this model using Least Absolute Shrinkage and Selection Operator (Lasso) with Cross Validation. Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces (see Wikipedia [2])

I did not follow a Tutorial. However, I made sure that I cover the main steps from the article "Time Series Forecast Case Study with Python: Annual Water Usage in Baltimore" by Brownlee 2017[3].

[1] https://web.stanford.edu/group/SITE/archive/SITE_2009/segment_1/s1_papers/corsi.pdf.
[2] https://en.wikipedia.org/wiki/Lasso_(statistics).
[3] https://machinelearningmastery.com/time-series-forecast-study-python-annual-water-usage-baltimore/.

2 Code Outline

In 1: I load the necessary library

In 2: I define functions for the estimators, i.e. x_t, x_{t-1}, $x_t^{(n)}$, ...

In 3: I define a function to set up a data frame with the dependend variable in the first row and the dependend variables in the following rows:

In 4: The models contain NaNs since we defined the depended variables by autoregression. For an example an AR(22) the first 22 rows are not defined. I define a function which deletes any rows with NaNs. Since I want to compare the models, I make sure that all the columns are deleted and all the models have the same length. For example when I want to compare a AR(3) model with an AR(22) model I delete the first 22 rows so I can compare them.

In 5: I define a function with statsmodel to estimate the models which gives nice output tables.

In 6: To evaluate the models, I divide the data set into a training window and a testing window. I use the training window to estimate the model and then make a prediction and compare the results to the real value.

According to Brownlee 2017 there possible two ways to proceed:

- Estimate the model and use it to predict the entire testing window.
- Estimate the model and use it in a rolling-forecast manner, estimating the transform and model for each time step. This is the preferred method as it is how one would use this model in practice as it would achieve the best performance.

In 7: I download stock market data for Amazon and calculate the Arman Klass Volatility (see Breaking Down Finance [4]). I then calculate some descriptive statistics, plot the volatility and create a histogram and autocorrelation diagram.

In 8: I estimate an AR(3) a HAR(1,5,22) and an AR (22) model for the amazon volatility. Furthermore I divide the sample in two and proceed with out-of-sample rolling window forecasts. I plot the forecasts for all the models. Additionally I estimate the AR(22) model with lasso. A table reports the RMSE of the forecasts.

[4] https://breakingdownfinance.com/finance-topics/risk-management/garman-klass-volatility/.

3 Python Code together with Output

```
In [1]: #Import the necessary libraries
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import statsmodels.api as sm
        from math import sqrt
        from sklearn.metrics import mean_squared_error
        from sklearn import linear_model
        import sklearn
        import pandas_datareader.data as web
        import datetime as dt
        from statsmodels.graphics.tsaplots import plot_acf

        %matplotlib inline
```

```
In [2]: ###Functions for estimators###
        def forecast(raw_data,model,forecast_horion):
            '''
            Function for aggregated volatility dependend variable:

            Parameters
            ----------
            raw_data : Dataframe with one row
                raw Data
            model : Dataframe
                Should be Dataframe to run the regression in a later step
            forecast_horion : int
                aggregated volatility, e.g. 2 for average (agregated) volatility in the next two days

            Returns
            -------
            Dataframe
                model with additional column containg the forecast
            '''
            def moving_average(a): #rolling function
                f1=sum(a[:])/(len(a)) #average of all the previous realizations excl. today (a[-1])
                return f1

            label="RV^(%d)_t"%(forecast_horion)
            raw_data_reveresed=raw_data[::-1] #reverse order to apply rolling function
            raw_data_reveresed[label] = raw_data_reveresed.rolling(forecast_horion).apply(moving_average)
            model[label]=raw_data_reveresed[label]#apply rolling function to
            return model


        def lag_estimator(raw_data,model,lag): #create new column with lag estimator
            '''
            Function for lag volatility estimators:

            Parameters
            ----------
            raw_data : Dataframe with one row
                raw Data
            model : Dataframe
                Should be Dataframe to run the regression in a later step
            lag : int
```

```
        laged volatility estimator, e.g. 3 for a three day lag

    Returns
    -------
    Dataframe
        model with additional column containg the estimator
    '''
    def lag_value(a): #rolling function
        f1=a[0] #average of all the previous realizations excl. today (a[-1])
        return f1

    label="RV_t-%d"%lag
    #apply rolling function
    model[label] = raw_data.iloc[:,0].rolling(lag+1).apply(lag_value)
    return model

def har_estimator(raw_data,model,horrizon):
    '''
    Function for aggregated volatility estimators:

    Parameters
    ----------
    raw_data : Dataframe with one row
        raw Data
    model : Dataframe
        Should be Dataframe to run the regression in a later step
    estimators_horizon : int
        aggregated volatility estimator, e.g. 22 for monthly component

    Returns
    -------
    Dataframe
        model with additional column containg the estimator
    '''
    def moving_average(a): #rolling function
        f1=sum(a[:-1])/horrizon #average of prev. realizations excl. today
        return f1

    label="RV^(%d)"%horrizon #create Lable
    #apply rolling function
    model[label] = raw_data.iloc[:,0].rolling(horrizon+1).apply(moving_average)
    return model
```

In [3]:
```
def AR_model(raw_data,forecast_horizon,estimators_horizon):
    '''
    Function for Autoregresive model:

    Parameters
    ----------
    raw_data : Dataframe with one row
        raw Data
    forecast_horizon : int
        Forecast horizon of independent varaibles, e.g. 1
    estimators_horizon : list with int
        lag estimators, e.g. [1,2,3] for one day, two day, three day lags

    Returns
    -------
    Dataframe
```

```
        Dataframe with first row dependend variables
        and following rows independend variables
    '''
    model=pd.DataFrame(index=raw_data.index)
    forecast(raw_data,model,forecast_horizon) #1st row with dependend variables

    for i in estimators_horizon:
        lag_estimator(raw_data,model,i)

    return model

def har_model(raw_data,forecast_horizon,estimators_horizon):
    '''
    Function for HAR model:

    Parameters
    ----------
    raw_data : Dataframe with one row
        raw Data
    forecast_horizon : int
        Forecast horizon of independent varaibles, e.g. 1
    estimators_horizon : list with int
        aggregated volatility estimators,
        e.g. [1,5,22] for daily, weekly and monthly component

    Returns
    -------
    Dataframe
        Dataframe with 1st row dependend var. and independend var.
    '''
    model=pd.DataFrame(index=raw_data.index) #set DataFrame with index
    forecast(raw_data,model,forecast_horizon) #dependend variables

    for i in estimators_horizon: #Independent variables variables
        har_estimator(raw_data,model,i)

    return model
```

```
In [4]: def remove_NaN(models):
    '''
    Drop any rows with NaN

    The models still contain NaN in the rows,
    as for example har_estimator function is not defined for first 22 rows.
    Therefore we need to exclude all rows with NaN.
    To make the models comparable, we have delete the rows in all the models

    Parameters
    ----------
    models : list with Dataframes
        list of all the models, e.g. [model1,model2,model3]

    Returns
    -------
    list with Dataframes
        Models without any NaN
    '''

    lst=[] #List of rownumbers with NaNs
```

```
for item1 in models: #Find the rows with NaNs
    i=0
    for item2 in item1.isnull().any(axis=1):
        if item2 == True and i not in lst:
            lst.append(i)
        i+=1

for item in models: #Drop the rows with NaNs
    item.drop(item.index[[lst]],inplace=True)
return models
```

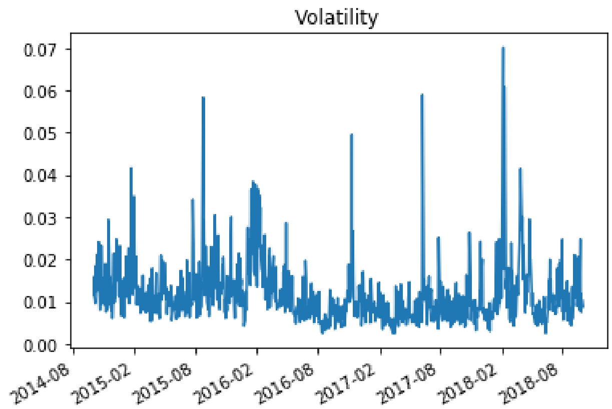
```
In [5]: def regress_model(model):
            '''
            Regress the models
            '''
            Y = model.iloc[:,0]
            X = model.iloc[:,1:]
            X = sm.add_constant(X)
            reg = sm.OLS(Y,X)
            results = reg.fit()
            return results
```

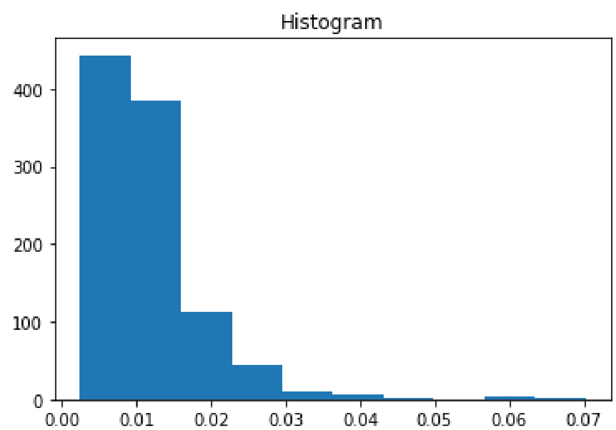
```
In [6]: def test_model(model, training_window,reg):
            '''
            Function for training the model and then test it

            I test the models applying a rolling-forecast manner,
            updating the transform and model for each time step

            Parameters
            ----------
            model : Dataframe
                model definded as above with first row dependend variables
                and following rows independed variabels
            training_window : int
                number of periods in the training window, should be shorter than the model
            reg : sklearn.linear_model
                Can be a Lasso regression or ols regression

            Returns
            -------
            Int
                Root Mean Squared Error of the predictions, the smaller the better
            list
                Predictions made
            list
                Ture values
            '''
            training_testing_window=[training_window,model.shape[0]-training_window]
            #define a vector with training window and testing window

            #Load the model and use it in a rolling-forecast manner, updating the transform and model for each time
            predictions=[]
            for i in range(0,training_testing_window[1]):
                y_train = model.iloc[i:training_testing_window[0]+i,0]
                X_train = model.iloc[i:training_testing_window[0]+i,1:]
                X_test = np.array([model.iloc[training_testing_window[0]+i,1:]])

                model_fit=reg.fit(X_train, y_train)
```

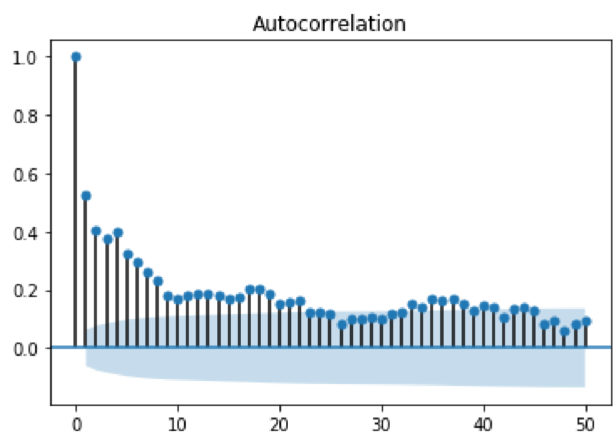
```
            model_prediction=model_fit.predict(X_test)[0]
            predictions.append(model_prediction)

        #put predicitons into a dataframe
        predictions = pd.DataFrame(predictions, index=model.iloc[training_testing_window[0]:,0].index)
        Observed_values = model.iloc[training_testing_window[0]:,0]

        #Calculate mean squared error of predictons
        mse = mean_squared_error(Observed_values, predictions)
        #Take the Root means squared error to evaluate the forecast performance
        rmse = sqrt(mse)

        #Ploting Predcitions vs. Realizations
        return rmse,predictions,Observed_values
```

In [7]:
```
#load data
start = dt.datetime(2014,10,1)
end = dt.datetime(2018,10,1)

#I use the Amazon stocks
df=web.DataReader('AMZN','iex', start, end).reset_index()
df['date']=pd.DatetimeIndex(df['date'])
df = df.set_index(df['date'],drop=True)

'''Calcualte Garman Klass Volatility, see:
https://breakingdownfinance.com/finance-topics/risk-management/garman-klass-volatility/'''
df["volatility"]=np.power(\
                  1/2*np.power(np.log(df["high"]/df["low"]),2)\
                  -(2*np.log(2)-1)*np.power(np.log(df["close"]/df["open"]),2)\
                  ,0.5)
raw_data=pd.DataFrame(df["volatility"])

#Descriptive Statistics
print(raw_data.describe())

#Time Series Plot
plt.plot(raw_data)
plt.title("Volatility")
plt.gcf().autofmt_xdate()
plt.show()

#Histogram
plt.hist(raw_data.iloc[:,0])
plt.title("Histogram")
plt.show()

#Autocorrelation Plot
plot_acf(raw_data.iloc[:,0],lags=50)
plt.show()
```

```
        volatility
count  1008.000000
mean      0.011688
std       0.006934
min       0.002503
25%       0.007228
50%       0.010040
75%       0.014036
max       0.070175
```


Volatility
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
2014-08
2015-02
2015-08
2016-02
2016-08
2017-02
2017-08
2018-02
2018-08


Histogram
400
300
200
100
0
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

In [8]:
```python
#Calculate three different models
AR_model_3=AR_model(raw_data,1,[1,2,3])
HAR_model=har_model(raw_data,1,[1,5,22])
AR_model_22=AR_model(raw_data,1,range(1,23))


models = [AR_model_3,HAR_model,AR_model_22]
labels = ["AR(3) Model","HAR(1,5,22) Model","AR(22) Model with OLS","AR(22) Model with Lasso"]

models=remove_NaN(models)

#print OLS regression tables
i=0
for item in models:
    print(labels[i],"\n")
    print(regress_model(item).summary(),"\n\n")
    i+=1

#print Out of sample performance and tables
RMSE_for_models=pd.DataFrame([],index=labels,columns=["RMSE"]) #list of all the RMSE

reg_OLS=sklearn.linear_model.LinearRegression()
training_window=int((len(AR_model_3)/2)) #set training window as half the sample size

plt.close()
fig = plt.figure(figsize=(8,8))

for item,i in zip(models,range(0,3)) :
    #calculate modles
    rmse, predictions,Observed_values=test_model(item,training_window,reg_OLS)
    RMSE_for_models["RMSE"][i]=rmse #set RMSE

    #Plot predictions vs. observed values
    ax = fig.add_subplot(4,1,i+1)
    ax.plot(Observed_values)
    ax.plot(predictions,color="red")
    ax.set_title(labels[i])
    ax.tick_params()

#estimate AR(22) with Lasso
reg_Lasso=sklearn.linear_model.LassoCV()
rmse, predictions,Observed_values=test_model(AR_model_22,training_window,reg_Lasso)

RMSE_for_models["RMSE"][3]=rmse #set RMSE

ax = fig.add_subplot(4,1,4)
ax.plot(Observed_values)
ax.plot(predictions,color="red")
ax.set_title("AR(22) Model with Lasso")
ax.tick_params()

plt.tight_layout()
plt.show()

print("The models have the following RMSE (lower is better):\n\n",RMSE_for_models)
best_model=pd.DataFrame.idxmin(RMSE_for_models.apply(pd.to_numeric, errors = 'coerce', axis = 0))[0]
print("\nThe %s preforms best in the out of sample rolling-forecasts.\n"%best_model)
```

AR(3) Model

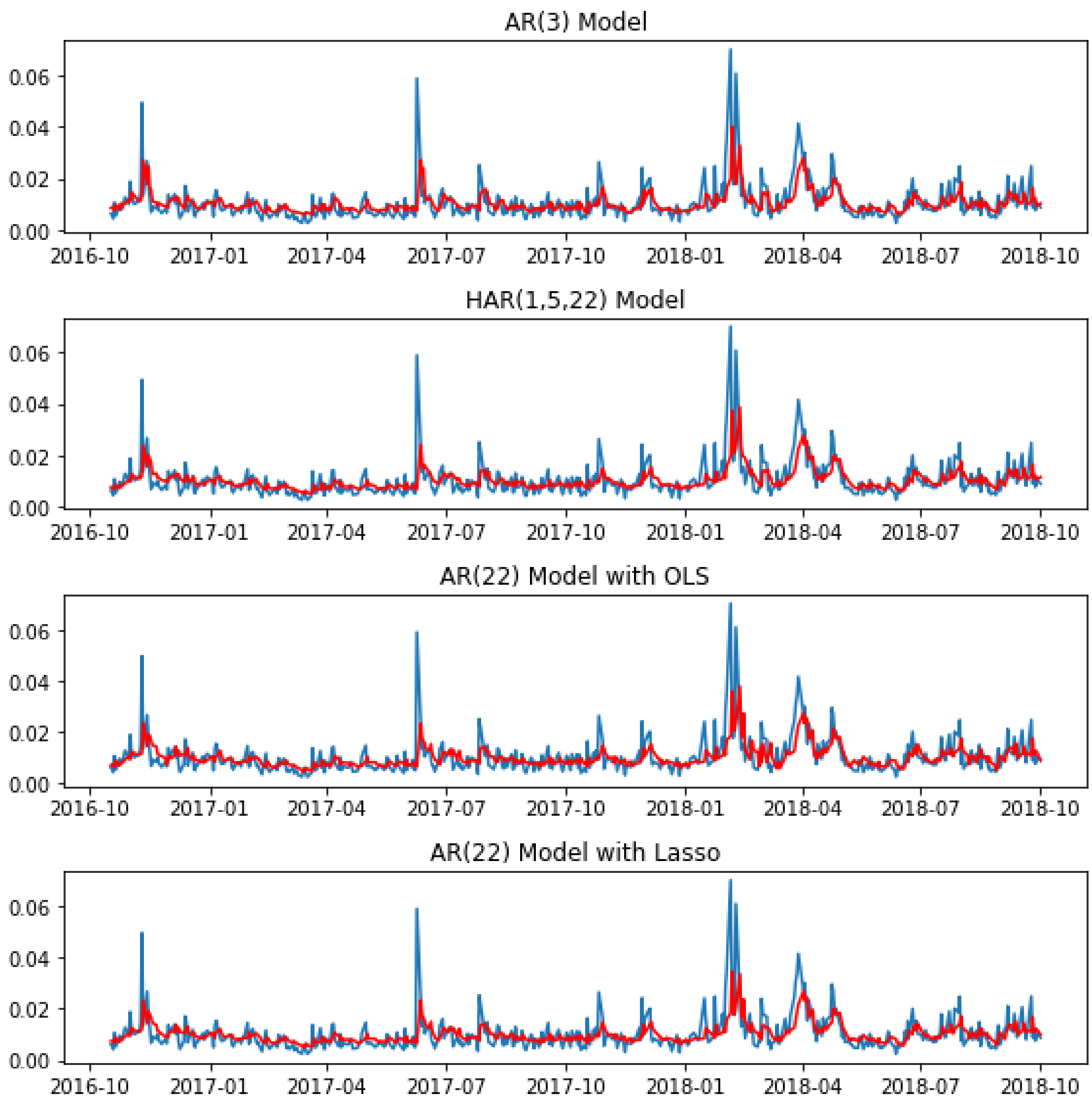
```
                            OLS Regression Results
==============================================================================
Dep. Variable:               RV^(1)_t   R-squared:                       0.316
Model:                            OLS   Adj. R-squared:                  0.313
Method:                 Least Squares   F-statistic:                     150.9
Date:                Fri, 09 Nov 2018   Prob (F-statistic):           1.97e-80
Time:                        14:26:54   Log-Likelihood:                 3685.3
No. Observations:                 986   AIC:                            -7363.
Df Residuals:                     982   BIC:                            -7343.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0038      0.000      9.110      0.000       0.003       0.005
RV_t-1         0.4038      0.032     12.793      0.000       0.342       0.466
RV_t-2         0.1180      0.034      3.482      0.001       0.051       0.184
RV_t-3         0.1474      0.032      4.671      0.000       0.085       0.209
==============================================================================
Omnibus:                      712.980   Durbin-Watson:                   2.052
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15383.264
Skew:                           3.062   Prob(JB):                         0.00
Kurtosis:                      21.356   Cond. No.                         218.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

HAR(1,5,22) Model

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               RV^(1)_t   R-squared:                       0.337
Model:                            OLS   Adj. R-squared:                  0.335
Method:                 Least Squares   F-statistic:                     166.1
Date:                Fri, 09 Nov 2018   Prob (F-statistic):           4.48e-87
Time:                        14:26:54   Log-Likelihood:                 3700.7
No. Observations:                 986   AIC:                            -7393.
Df Residuals:                     982   BIC:                            -7374.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0023      0.001      4.009      0.000       0.001       0.003
RV^(1)         0.2808      0.038      7.428      0.000       0.207       0.355
RV^(5)         0.3755      0.063      5.926      0.000       0.251       0.500
RV^(22)        0.1487      0.066      2.266      0.024       0.020       0.277
==============================================================================
Omnibus:                      725.618   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            16713.418
Skew:                           3.117   Prob(JB):                         0.00
Kurtosis:                      22.182   Cond. No.                         459.
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

AR(22) Model with OLS

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               RV^(1)_t   R-squared:                       0.350
Model:                            OLS   Adj. R-squared:                  0.335
Method:                 Least Squares   F-statistic:                     23.60
Date:                Fri, 09 Nov 2018   Prob (F-statistic):           4.11e-75
Time:                        14:26:54   Log-Likelihood:                 3711.0
No. Observations:                 986   AIC:                            -7376.
Df Residuals:                     963   BIC:                            -7263.
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0022      0.001      3.938      0.000       0.001       0.003
RV_t-1         0.3611      0.032     11.206      0.000       0.298       0.424
RV_t-2         0.0892      0.034      2.605      0.009       0.022       0.156
RV_t-3         0.0690      0.034      2.007      0.045       0.002       0.136
RV_t-4         0.1645      0.034      4.783      0.000       0.097       0.232
RV_t-5         0.0133      0.035      0.382      0.702      -0.055       0.081
RV_t-6         0.0295      0.035      0.847      0.397      -0.039       0.098
RV_t-7         0.0075      0.035      0.215      0.830      -0.061       0.076
RV_t-8         0.0047      0.035      0.135      0.893      -0.064       0.073
RV_t-9        -0.0590      0.035     -1.698      0.090      -0.127       0.009
RV_t-10       -0.0208      0.035     -0.597      0.550      -0.089       0.048
RV_t-11        0.0130      0.035      0.373      0.709      -0.055       0.081
RV_t-12        0.0162      0.035      0.466      0.642      -0.052       0.085
RV_t-13        0.0280      0.035      0.804      0.422      -0.040       0.096
RV_t-14        0.0205      0.035      0.589      0.556      -0.048       0.089
RV_t-15       -0.0103      0.035     -0.298      0.766      -0.079       0.058
RV_t-16        0.0044      0.035      0.128      0.899      -0.064       0.073
RV_t-17        0.0569      0.035      1.639      0.101      -0.011       0.125
RV_t-18        0.0287      0.035      0.826      0.409      -0.040       0.097
RV_t-19        0.0124      0.034      0.361      0.718      -0.055       0.080
RV_t-20       -0.0348      0.034     -1.014      0.311      -0.102       0.033
RV_t-21       -0.0043      0.034     -0.124      0.901      -0.072       0.063
RV_t-22        0.0177      0.032      0.550      0.582      -0.046       0.081
==============================================================================
Omnibus:                      729.428   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17607.666
Skew:                           3.122   Prob(JB):                         0.00
Kurtosis:                      22.738   Cond. No.                         244.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```



The models have the following RMSE (lower is better):

	RMSE
AR(3) Model	0.0061361
HAR(1,5,22) Model	0.00606348
AR(22) Model with OLS	0.00615742
AR(22) Model with Lasso	0.00609591

The HAR(1,5,22) Model preforms best in the out of sample rolling-forecasts.