VAR Combination Function Explanation

1. The datasetTwo variable is created by selecting the columns specified in listt from the dataset. This subset is what will be used in the VAR model.

```
datasetTwo=data1[listt]
```

```
datasetTwo
```

	Close Price	High Price	Open Price	Low Price
0	1.000000	1.000000	1.000000	1.000000
1	0.989929	0.992387	0.996183	0.984549
2	0.995573	0.989649	0.991495	0.992444
3	0.994153	0.993055	0.985770	0.994138
4	0.989186	0.984374	0.981450	0.988039
...	...	...	...	...
2388	0.112876	0.111953	0.112168	0.113645
2389	0.113282	0.112688	0.112168	0.113679
2390	0.111997	0.111586	0.112871	0.113035
2391	0.113282	0.111419	0.111565	0.113170
2392	0.114464	0.112855	0.112235	0.114356

2393 rows × 4 columns

2. test_obs is set to 28, meaning the last 28 observations in the time series will be reserved for testing (i.e., validation of the model).

```
test_obs = 28
```

3. The dataset is split into train and test sets. train contains all except the last 28 observations, and test contains the last 28 observations.

```
train =datasetTwo[:-test_obs]
test = datasetTwo[-test_obs:]
```

```
train
```

	Close Price	High Price	Open Price	Low Price
0	1.000000	1.000000	1.000000	1.000000
1	0.989929	0.992387	0.996183	0.984549
2	0.995573	0.989649	0.991495	0.992444
3	0.994153	0.993055	0.985770	0.994138
4	0.989186	0.984374	0.981450	0.988039
...	...	...	...	...
2360	0.128388	0.130417	0.127101	0.128486
2361	0.126766	0.127279	0.127972	0.126690
2362	0.120412	0.124841	0.125661	0.120625
2363	0.118182	0.118731	0.120070	0.118795
2364	0.120379	0.119599	0.116922	0.119202

2365 rows × 4 columns

```
test
```

	Close Price	High Price	Open Price	Low Price
2365	0.120277	0.119232	0.120003	0.120184
2366	0.119196	0.118598	0.119065	0.118965
2367	0.115985	0.117462	0.118730	0.116660
2368	0.114836	0.113523	0.114244	0.112628
2369	0.118216	0.116394	0.115047	0.116084
2370	0.116830	0.116494	0.116922	0.117067
2371	0.117033	0.116628	0.117257	0.117745
2372	0.118587	0.116728	0.115583	0.116762
2373	0.115106	0.116227	0.116922	0.115237
2374	0.115918	0.114891	0.114880	0.116491
2375	0.117810	0.116427	0.115884	0.117406
2376	0.116593	0.116260	0.117525	0.117236
2377	0.113146	0.113556	0.114846	0.114018
2378	0.110308	0.111753	0.112837	0.110189
2379	0.112234	0.110551	0.109355	0.110426

4.The VAR (Vector Autoregression) model class is imported from the statsmodels library.

5. This loop fits a VAR model to the train dataset for various lag orders (from 1 to 10).

- model = VAR(train): Initializes the VAR model with the training data.
- results = model.fit(i): Fits the VAR model with i lags.
- print('Order =', i): Prints the current lag order being tested.
- print('AIC: ', results.aic): Prints the Akaike Information Criterion (AIC) for the model, a measure of model quality.
- print('BIC: ', results.bic): Prints the Bayesian Information Criterion (BIC) for the model, another measure of model quality.

```python
from statsmodels.tsa.api import VAR
for i in [1,2,3,4,5,6,7,8,9,10]:
    model = VAR(train)
    results = model.fit(i)
    print('Order =', i)
    print('AIC: ', results.aic)
    print('BIC: ', results.bic)
    print()
```

```
Order = 1
AIC:  -42.34498608158102
BIC:  -42.296186501390444

Order = 2
AIC:  -42.36259939092901
BIC:  -42.27472941971499

Order = 3
AIC:  -42.36571329501595
BIC:  -42.238745586365916

Order = 4
AIC:  -42.37370393795882
BIC:  -42.20761111435788

Order = 5
AIC:  -42.39758426616225
BIC:  -42.1923389189446

Order = 6
AIC:  -42.528619408252624
BIC:  -42.28419409755285

Order = 7
AIC:  -42.615087149227136
```

6. The optimal lag order is selected based on the AIC criterion.

- model.select_order(maxlags=12): Tests lag orders up to 12 and returns the optimal order based on various criteria.
- order = x.selected_orders["aic"]: Retrieves the lag order that minimizes the AIC value.

```
x = model.select_order(maxlags=12)
```

```
x
```

```
<statsmodels.tsa.vector_ar.var_model.LagOrderResults at 0x177eafe50>
```

```
order=x.selected_orders["aic"]
```

```
order
```

```
11
```

7. Fits the VAR model with the optimal lag order determined by AIC.

```
result = model.fit(order)
```

```
result
```

```
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper at 0x177eaff10>
```

8.Forecasts the next 28 steps using the fitted model.

- lagged_Values = train.values[-order:]: Retrieves the last order number of observations from the training data. For example, if order is 3, then train.values[-3:] will return the last 3 rows of the train dataset
- pred = result.forecast(y=lagged_Values, steps=28): Uses these lagged values to forecast the next 28 time points.

```
lagged_Values = train.values[-order:]
```

```
lagged_Values
```

```
array([[0.13065225, 0.13001669, 0.13028193, 0.13153526],
       [0.12615749, 0.12954925, 0.1299471 , 0.1257412 ],
       [0.12767827, 0.12554257, 0.12643139, 0.1271643 ],
       [0.12865833, 0.12808013, 0.12730195, 0.12882459],
       [0.12865833, 0.12808013, 0.12730195, 0.12882459],
       [0.1309564 , 0.12891486, 0.12773723, 0.12994274],
       [0.12838797, 0.13041736, 0.12710105, 0.12848575],
       [0.1267658 , 0.1272788 , 0.12797161, 0.12668993],
       [0.1204123 , 0.1248414 , 0.12566129, 0.12062481],
       [0.11818182, 0.11873122, 0.12006964, 0.11879511],
       [0.12037851, 0.11959933, 0.11692225, 0.11920171]])
```

```
pred = result.forecast(y=lagged_Values,steps=28)
```

```
pred
```

```
array([[0.1217878 , 0.1217821 , 0.12073247, 0.12150167],
       [0.12220853, 0.12232084, 0.12110624, 0.12167262],
       [0.12227584, 0.12203179, 0.12027382, 0.12177288],
       [0.12175208, 0.12191634, 0.1212719 , 0.12126583],
       [0.12166122, 0.12182199, 0.12081381, 0.12075866],
       [0.12130974, 0.12216011, 0.12157467, 0.12054079],
       [0.12118984, 0.12149157, 0.11994382, 0.12023713],
       [0.12099209, 0.12178954, 0.12032233, 0.12001349],
```

9. Converts the forecasted values into a DataFrame and saves it as a CSV file named varforecasted_28.csv

```
    [0.11660885, 0.11699269, 0.11590364, 0.11598147]])
```

```
preds=pd.DataFrame(pred,columns=listt)
preds.to_csv("varforecasted_{}.csv".format(test_obs))
```

10.Computes the performance metrics RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error) to evaluate the forecast.

- rmse = round(mean_squared_error(test, pred, squared=False)): Computes RMSE between the actual (test) and forecasted (pred) values.
- mape = mean_absolute_percentage_error(test, pred): Computes MAPE between the actual and forecasted values.

11. Appends the model's performance metrics (like RMSE, MAPE, and lag order) to the performance dictionary, which tracks these values for multiple runs/models.
12.Converts the performance dictionary into a DataFrame and returns it along with the fitted VAR model (result) and the forecasted values (pred).

```
from sklearn.metrics import mean_squared_error
rmse= round(mean_squared_error(test,pred,squared=False))
from sklearn.metrics import mean_absolute_percentage_error
mape=mean_absolute_percentage_error(test,pred)
performance["Model"].append(listt)
performance["RMSE"].append(rmse)
performance["MaPe"].append(mape)
performance["Lag"].append(order)
performance["Test"].append(test_obs)
perf=pd.DataFrame(performance)
```

```
perf
```

	Model	RMSE	MaPe	Lag	Test
0	[Close Price, High Price, Open Price, Low Price]	0	0.04286	11	28