The application consists of 1 main window. And 3 ViewModels are used for different level of data control:

1. MainViewModel.cs is the view model object instantiated by the MainWindow.xmal, where the DataSource is created and subscribed by the UpdateFinancialData callback.
2. FinanacialDataViewModel.cs is the view model that wraps around FinancialData. It exposes its properties binding to the PricingSpec ComboBox and the TimeStamp Lable. Specially, whenever the model of FinancialData is updated/set, change is notified to its property PricingSpecRows via UpdatePriceSpecRows()(Observer Pattern).
3. Similar pattern is with PricingSpecViewModel.cs – all properties updated via setting the Model property. PricingSpecViewModel’s ContratRows is bound to the ItemsSource of the DataGrid, and its properties to each DataGridColumn. Since we have no interest in keeping the ContractData, no further ViewModel is constructed upon it.

For formatting and styling data inside the DataGrid, I use 2 converters – NegativeColourConverter and PercentageConverter. Additional ValueHelperExtension is created to help validate the value when using the value converters. The reasons I chose to use value converters are:

1. Advantageous over wrapping the logic of percentage conversion in ViewModel because the segregation of this logic can be unit tested, further extended with more complex business logics, and ready to be used by any data that needs this conversion in the UI.
2. Advantageous over using Style with DataTrigger for styling negative numbers because it provides a more interactive data conversion between the View and ViewModel by the ConvertBack method, especially when you need Two-way binding (potentially). And it has more flexible usage provided with parameters, i.e. a negative converter can be used to convert many different ways.