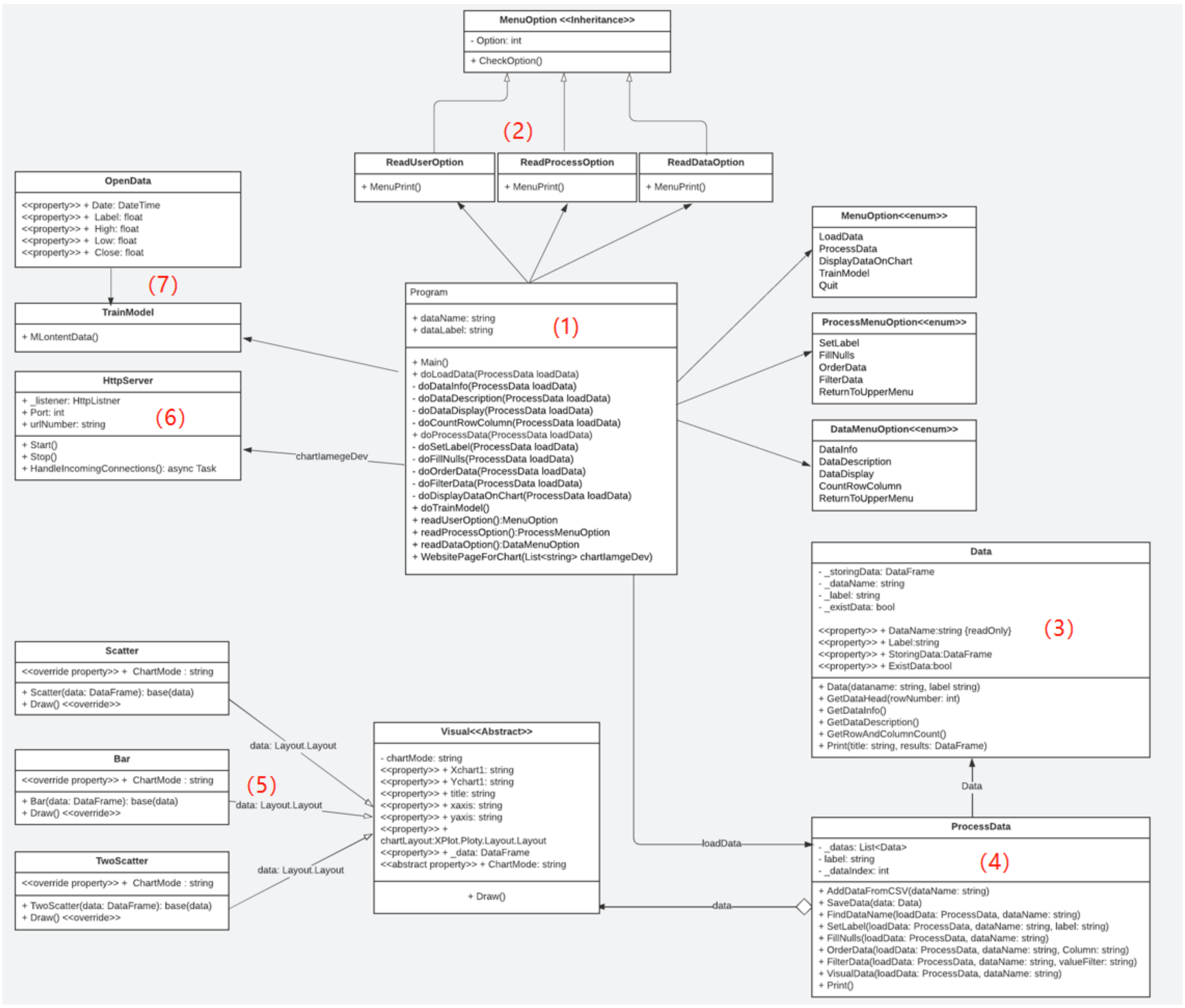
Task 7.3

Video link:

https://video.deakin.edu.au/media/t/1_myadef61



MenuOption <<Inheritance>>
- Option: int
+ CheckOption()
(2)
ReadUserOption
+ MenuPrint()
ReadProcessOption
+ MenuPrint()
ReadDataOption
+ MenuPrint()
OpenData
<<property>> + Date: DateTime
<<property>> + Label: float
<<property>> + High: float
<<property>> + Low: float
<<property>> + Close: float
(7)
TrainModel
+ MLontentData()
HttpServer
+ _listener: HttpListner
+ Port: int
+ urlNumber: string
(6)
+ Start()
+ Stop()
+ HandleIncomingConnections(): async Task
chartlamegeDev
Program
+ dataName: string
+ dataLabel: string
(1)
+ Main()
+ doLoadData(ProcessData loadData)
- doDataInfo(ProcessData loadData)
- doDataDescription(ProcessData loadData)
- doDataDisplay(ProcessData loadData)
- doCountRowColumn(ProcessData loadData)
+ doProcessData(ProcessData loadData)
- doSetLabel(ProcessData loadData)
- doFillNulls(ProcessData loadData)
- doOrderData(ProcessData loadData)
- doFilterData(ProcessData loadData)
- doDisplayDataOnChart(ProcessData loadData)
+ doTrainModel()
+ readUserOption():MenuOption
+ readProcessOption():ProcessMenuOption
+ readDataOption():DataMenuOption
+ WebsitePageForChart(List<string> chartlamgeDev)
MenuOption<<enum>>
LoadData
ProcessData
DisplayDataOnChart
TrainModel
Quit
ProcessMenuOption<<enum>>
SetLabel
FillNulls
OrderData
FilterData
ReturnToUpperMenu
DataMenuOption<<enum>>
DataInfo
DataDescription
DataDisplay
CountRowColumn
ReturnToUpperMenu
Data
- _storingData: DataFrame
- _dataName: string
- _label: string
- _existData: bool
<<property>> + DataName:string {readOnly}
<<property>> + Label:string
<<property>> + StoringData:DataFrame
<<property>> + ExistData:bool
(3)
+ Data(dataname: string, label string)
+ GetDataHead(rowNumber: int)
+ GetDataInfo()
+ GetDataDescription()
+ GetRowAndColumnCount()
+ Print(title: string, results: DataFrame)
Data
Scatter
<<override property>> + ChartMode : string
+ Scatter(data: DataFrame): base(data)
+ Draw() <<override>>
Bar
<<override property>> + ChartMode : string
+ Bar(data: DataFrame): base(data)
+ Draw() <<override>>
TwoScatter
<<override property>> + ChartMode : string
+ TwoScatter(data: DataFrame): base(data)
+ Draw() <<override>>
data: Layout.Layout
(5)
data: Layout.Layout
data: Layout.Layout
Visual<<Abstract>>
- chartMode: string
<<property>> + Xchart1: string
<<property>> + Ychart1: string
<<property>> + title: string
<<property>> + xaxis: string
<<property>> + yaxis: string
<<property>> + chartLayout:XPlot.Ploty.Layout.Layout
<<property>> + _data: DataFrame
<<abstract property>> + ChartMode: string
+ Draw()
loadData
data
ProcessData
- _datas: List<Data>
- label: string
- _dataIndex: int
(4)
+ AddDataFromCSV(dataName: string)
+ SaveData(data: Data)
+ FindDataName(loadData: ProcessData, dataName: string)
+ SetLabel(loadData: ProcessData, dataName: string, label: string)
+ FillNulls(loadData: ProcessData, dataName: string)
+ OrderData(loadData: ProcessData, dataName: string, Column: string)
+ FilterData(loadData: ProcessData, dataName: string, valueFilter: string)
+ VisualData(loadData: ProcessData, dataName: string)
+ Print()

Figure 1

As shown in Figure 1, we have defined seven tags representing seven parts of the program. The main program, menu, data module, preprocessing module, visualization processing, visualization server, and model training.

As shown in label 2, the menu module is divided into the main menu MenuOption and the secondary menu (ProcessMenuOption, DataMenuOption). We have adopted the concept of Inheritance so that we can reuse the fields and methods of the existing class. When we want to create a new class and have a class that contains some code we want, we can derive our new class from the existing class. MeneOption<< Inheritance>>is the base class (parent class) of the inheritance class. ReadUserOption, ReadProcessOption, and ReadDataOption classes are subclasses that inherit the characteristics of MenuOption.

As shown in tag 3, the Data class declares all variables as private variables and sets and obtains the value of variables through the C # attribute in the class. The variables and data in the class are hidden from other classes and can only be accessed by declaring any member function of the Data class. Such as the DataFrame StoringData, data name, label, data existence label and other private variables are declared in the Data class to perform basic operations on data after loading data. This class provides basic data methods, for example, getting the first five rows of data, data information, data description, and print method.

As shown in tag 4, the ProcessData class is a method set. This class defines a list of Data classes to store multiple data sets. The _dataIndex property is the list index that keeps the corresponding dataset name. This class defines the methods of loading data, saving data, finding data names, setting labels, filling empty values, sorting, filtering, visualization, and printing information to preprocess data.

As shown in Tab 5, the visual chart module uses the concept of abstraction to distinguish the properties and fields of objects from objects of similar types, which helps classify/group objects. We defined a parent class of the visual class to define the layer structure method of the chart type for all charts and the parameters of the chart (chart mode, data row and column name, subject, horizontal axis and vertical axis name). This parent class declares the structure of a given abstraction without providing a complete implementation of each method. In other words, we define a class that will be shared by all other subclasses and inherit this class to its subclasses to improve the details. What's more, three subclasses are provided to inherit the characteristics of the visual class, which are Scatter, TwoScatter (double-line scanner) and

Bar class. For example, the Scatter class extends the chart schema attribute, draws methods to display the scatter diagram and shows the javascript code by the string method passed to the visualization method in the processData class.

As shown in tag 6, we use the HttpListener class to create a new instance of a simple HTTP protocol listener that responds to HTTP requests. The listener is active during the lifetime of the new instance and uses the prefixes attribute to define the Uniform Resource Identifier (URI) prefix and port that should be processed. The Start method is used to create an HTTP server and start listening for incoming connections. The Stop() method is used to shut down the HTTP service. The HandleIncomingConnections () method provides a synchronization model for processing client requests. It uses the HttpListenerContext and GetContext methods to make access requests to objects and their associated responses. This program does not adopt a more complex asynchronous model. Because the application block while waiting for client requests and only processes one request at a time, we adopt a synchronous model. In further research and development in the future, we can explore asynchronous models to facilitate concurrent connections from clients. In both models, incoming requests can be accessed using the HttpListenerContext.Request property, and HttpListenerContext.Response can be used when responding to requests.

As shown in Tab 7, for the data analysis module, we use the ML.NET library. We use the method of MLContext class to create new instances for loading data, feature engineering, model training, prediction and model evaluation, and saving the model. The OpenData class is the column name of the input data. In ML.NET, you can customize the input column name to be inconsistent with the output name. Secondly, we load the input data into IDataView, split it into training/test sets, and then normalize it. Next, the regression model is used to predict the stock open price under the condition that a group of other factors are known, the model is saved, and the R-require value is obtained to judge the model's accuracy.

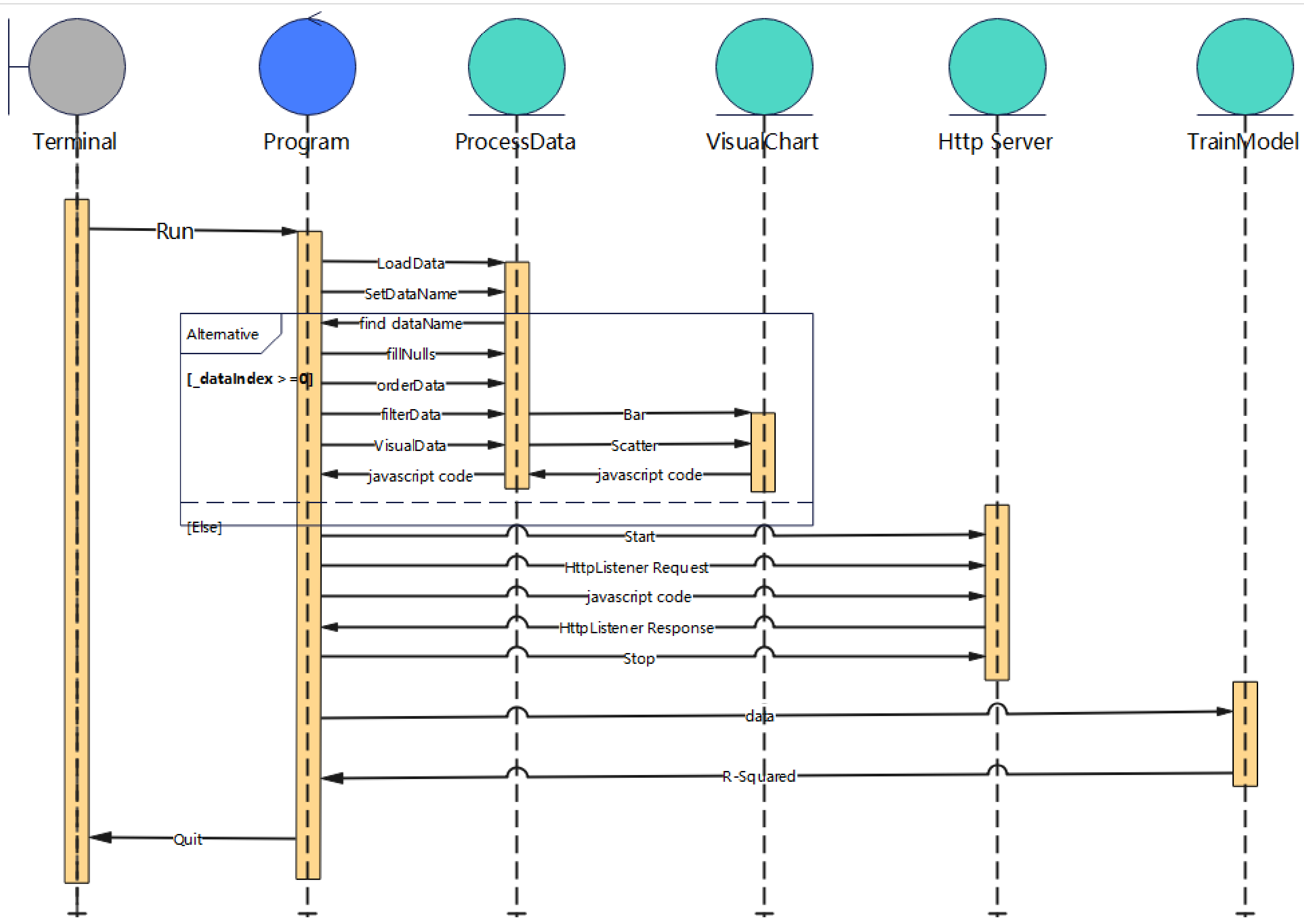


Figure 2

As shown in Figure 2, when the program starts to run, the main function of the Program class will perform the menu function. We can load data through the menu function and set the name of each loaded dataset. Determine the preprocessing operation of the dataset by judging its name. Secondly, we can select the visual data operation in the menu function. The method in the visualChart class can return the javascript code of the visual image to the method in the Program class. Start the HTTP Server through the method of the Program class and provide the service of a simple HTTP protocol listener to respond to HTTP requests so that users can access the visual chart web page. More, we can select the model training menu to conduct regression model training, get R-Squared and save the model for future use. Finally, we can exit the program through the menu.