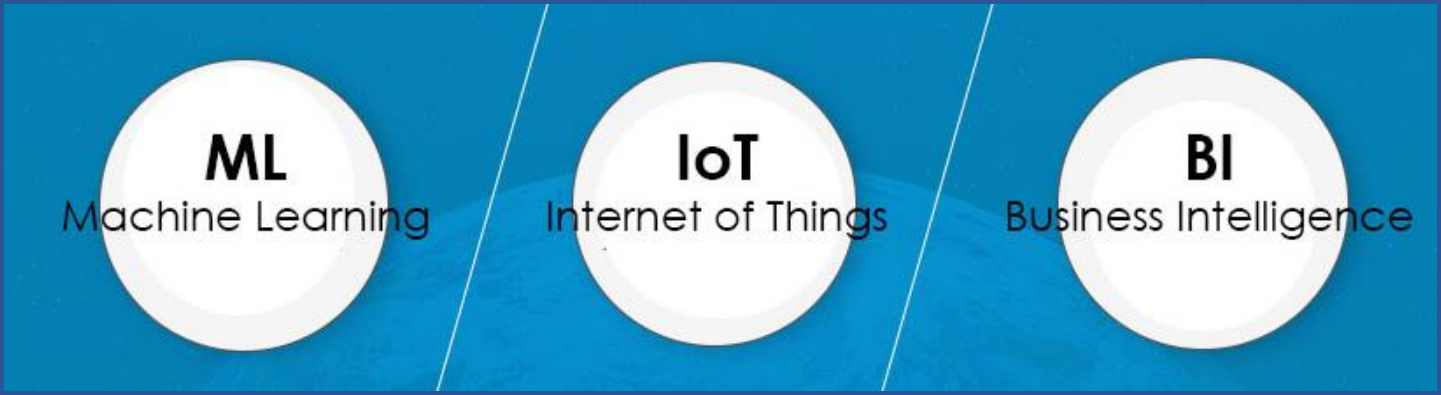
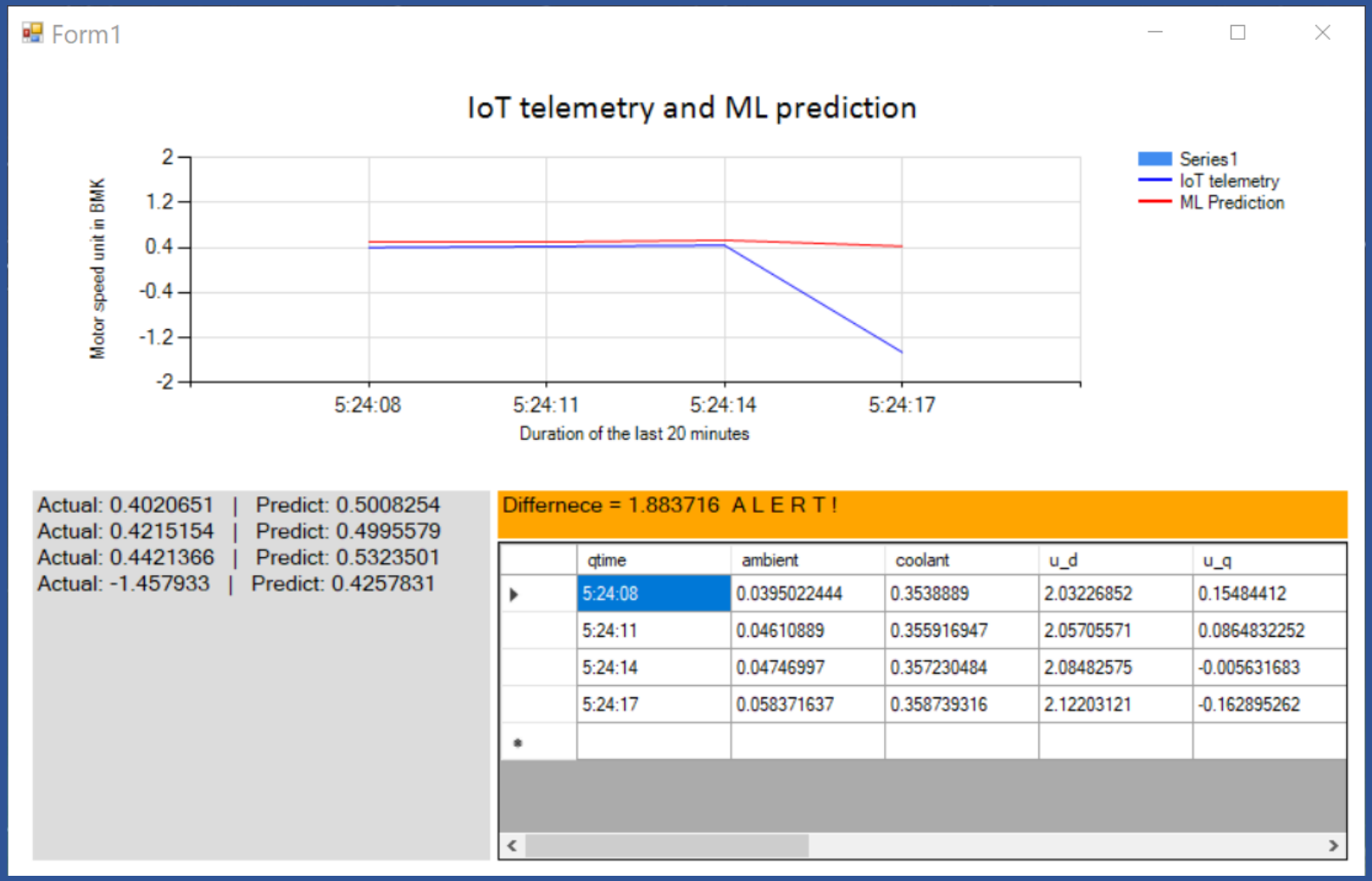
MIB App

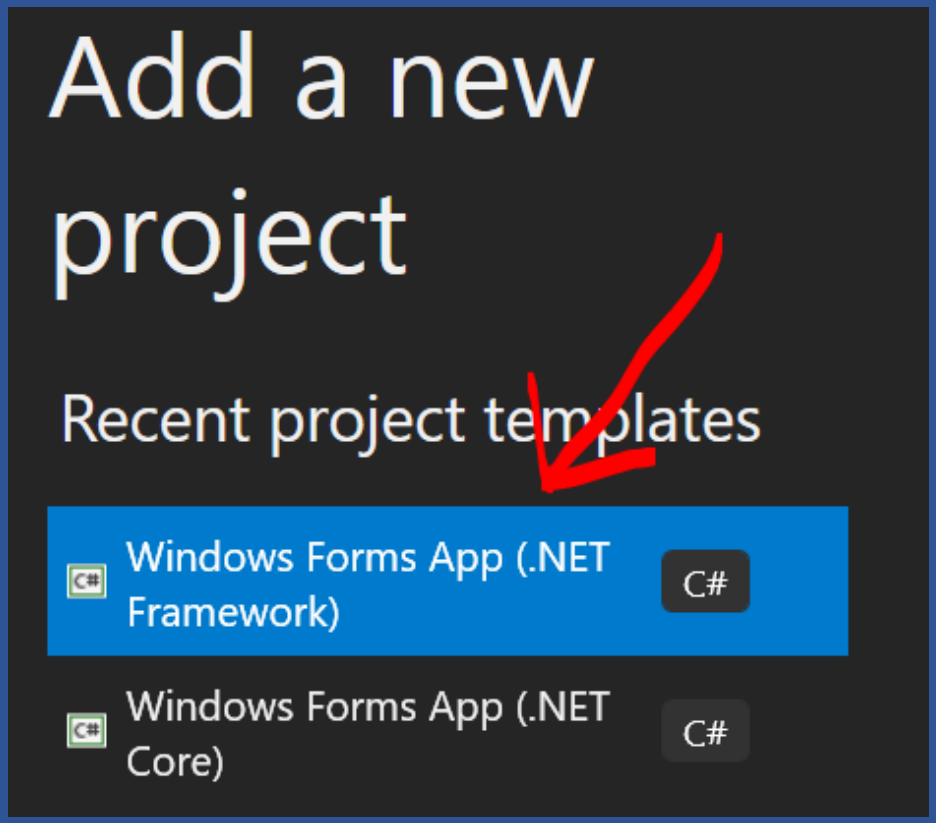

ML
Machine Learning
IoT
Internet of Things
BI
Business Intelligence

Create WinForm show a Line chart real-time D2C & ML



qtime	ambient	coolant	u_d	u_q
5:24:08	0.0395022444	0.3538889	2.03226852	0.15484412
5:24:11	0.04610889	0.355916947	2.05705571	0.0864832252
5:24:14	0.04746997	0.357230484	2.08482575	-0.005631683
5:24:17	0.058371637	0.358739316	2.12203121	-0.162895262

Open Visual Studio / Create C# .NET Framework WinForm



Add NuGet packages

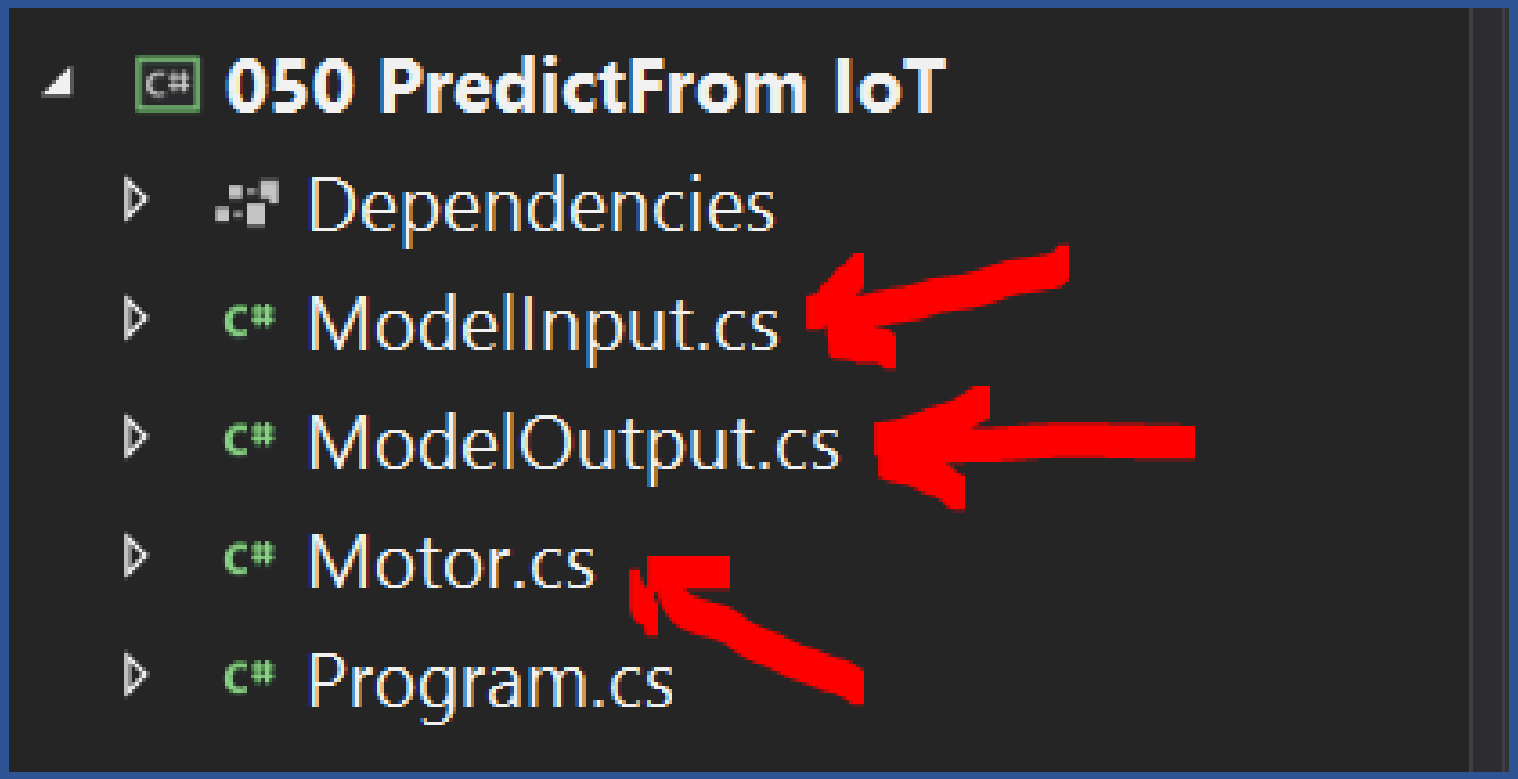
Microsoft.Azure.EventHubs by Microsoft v4.1.0
This is the next generation Azure Event Hubs .NET Standard client library....

Microsoft.ML by Microsoft v1.3.1
ML.NET is a cross-platform open-source machine learning framework whi...

Microsoft.ML.LightGbm by Microsoft v1.3.1
ML.NET component for LightGBM

Newtonsoft.Json by James Newton-King v12.0.2
Json.NET is a popular high-performance JSON framework for .NET

Copy data models to project



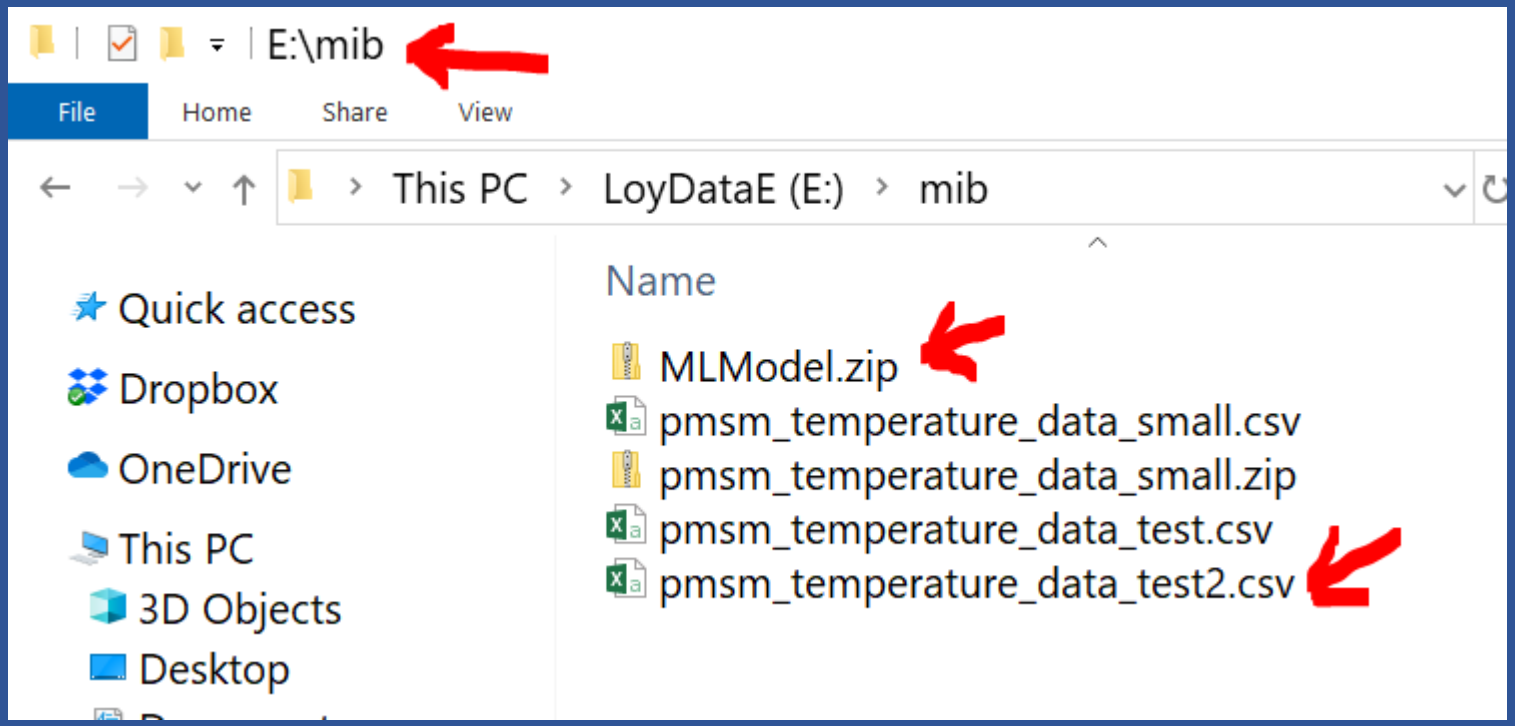
Add base class reference

```
3   using System;
4   using Microsoft.ML.Data;
5
6   namespace test
7   {
8       public class ModelOutput: ModelInput
9       {
10          public float Score { get; set; }
11      }
12  }
13
```

Add using to Program

```
using Microsoft.Azure.EventHubs;
using Microsoft.ML;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
```

Add ML Model and test data to mib folder



Class Motor add predict_speed

```
public class Motor
{
    public string qtime { get; set; }
    public float ambient { get; set; }
    public float coolant { get; set; }
    public float u_d { get; set; }
    public float u_q { get; set; }
    public float motor_speed { get; set; }
    public float torque { get; set; }
    public float i_d { get; set; }
    public float i_q { get; set; }
    public float pm { get; set; }
    public float stator_yoke { get; set; }
    public float stator_tooth { get; set; }
    public float stator_winding { get; set; }
    public float predict_speed { get; set; }
}
```

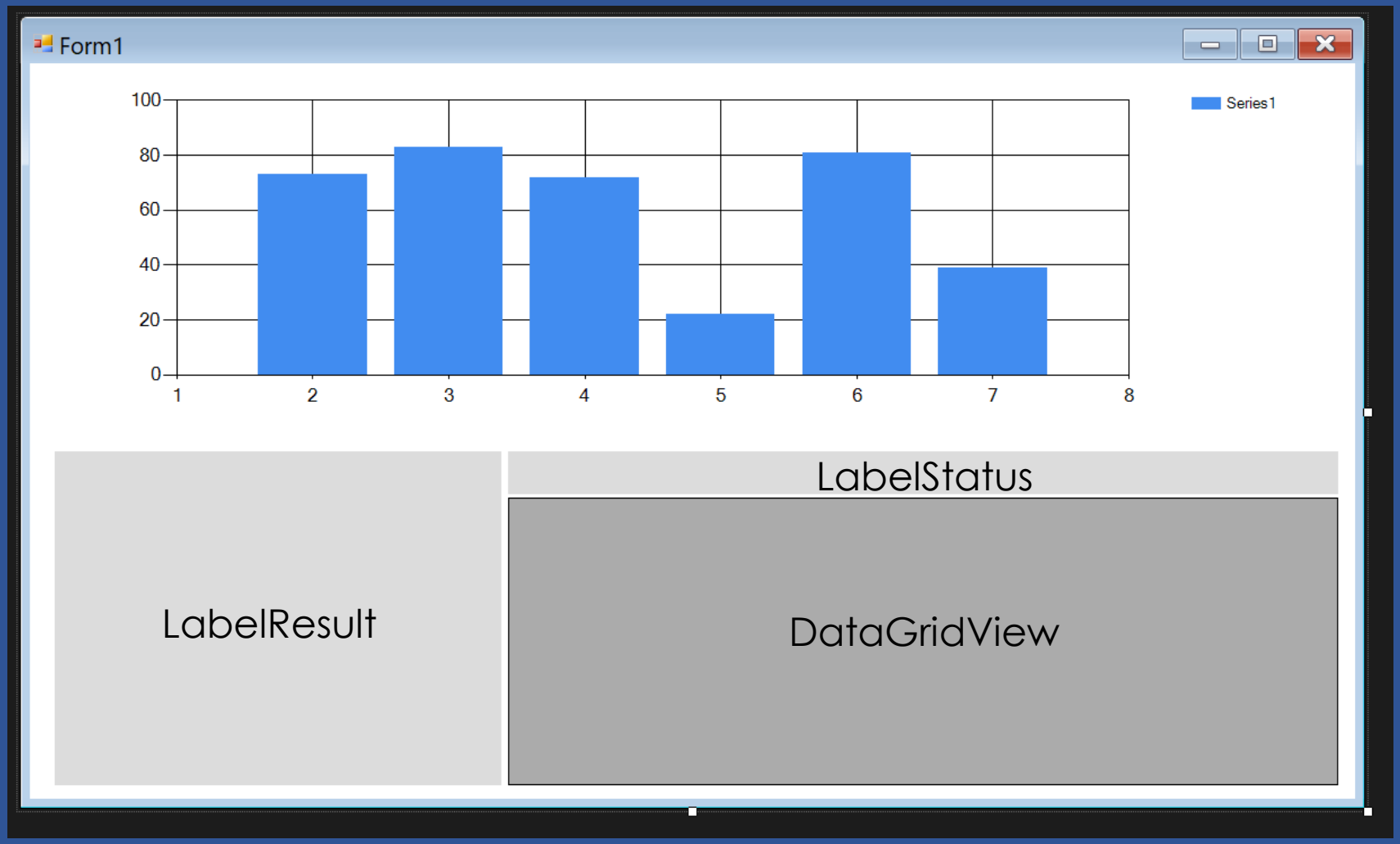
Add class members

```
#region members
private const float DIFF = 1;
private Timer myTimer = new Timer();
private bool ready = true;
private List<Motor> datasetIoT = new List<Motor>();

private readonly string s_eventHubsCompatibleEndpoint =
    "sb://ihsuprodsgres001dednamespace.servicebus.windows.net/";
private readonly string s_eventHubsCompatiblePath =
    "iothub-ehub-l[redacted]4bf924f3c";
private readonly string s_iotHubSasKey =
    "KCyf3omKkmWnc[redacted]xBWRoWgmAw=";
private readonly string s_iotHubSasKeyName = "service";
private EventHubClient s_eventHubClient;
private PartitionReceiver eventHubReceiver;

private static ITransformer mlModel;
private static string modelPath = @"E:\mib\MLModel.zip";
private static MLContext mlContext = new MLContext(seed: 0);
#endregion
```

Add a Label and a Chart control to Form1



Add method SetChartArea()

```
private void SetChartArea()
{
    Title title1 = new Title();
    title1.Font = new Font("Calibri", 16.2F, FontStyle.Regular,
        GraphicsUnit.Point, ((byte)(0)));
    title1.Name = "Title1";
    title1.Text = "IoT telemetry and ML prediction";
    this.chart1.Titles.Add(title1);

    chart1.ChartAreas[0].AxisX.MajorGrid.LineColor = Color.Gainsboro;
    chart1.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.Gainsboro;
    chart1.ChartAreas[0].AxisY.Maximum = 2;
    chart1.ChartAreas[0].AxisY.Minimum = -2;

    chart1.ChartAreas[0].AxisX.Title = "Duration of the last 20 minutes";
    chart1.ChartAreas[0].AxisX.TitleAlignment = StringAlignment.Center;
    chart1.ChartAreas[0].AxisX.TextOrientation = TextOrientation.Horizontal;

    chart1.ChartAreas[0].AxisY.Title = "Motor speed unit in BMK";
    chart1.ChartAreas[0].AxisY.TitleAlignment = StringAlignment.Center;
    chart1.ChartAreas[0].AxisY.TextOrientation = TextOrientation.Rotated270;
```

Add Method SetChartArea() (continue)

```
    var speedSeries1 = new Series("IoT");
    speedSeries1.ChartType = SeriesChartType.Line;
    speedSeries1.Color = Color.Blue;
    chart1.Series.Add(speedSeries1);
    chart1.Series["IoT"].LegendText = "IoT telemetry";

    var speedSeries2 = new Series("ML");
    speedSeries2.ChartType = SeriesChartType.Line;
    speedSeries2.Color = Color.Red;
    chart1.Series.Add(speedSeries2);
    chart1.Series["ML"].LegendText = "ML Prediction";
}
```

Add method GetD2CMessage()

```
private async Task GetD2CMessage()
{
    var events = await eventHubReceiver.ReceiveAsync(100);
    if (events == null) { ready = true; return; }
    foreach (EventData eventData in events)
    {
        string s = Encoding.UTF8.GetString(eventData.Body.Array);
        Motor m = new Motor();
        m = JsonConvert.DeserializeObject<Motor>(s);

        m.qtime = DateTime.Now.ToString("h:mm:ss");
        var predict = GetPrediction(m);
        labelResult.Text += $"Actual: {m.motor_speed}   |   Predict: {predict.Score}\r\n";
        FindDifference(m.motor_speed, predict.Score);
        m.predict_speed = predict.Score;
        datasetIoT.Add(m);

        var bindingList = new BindingList<Motor>(datasetIoT);
        var source = new BindingSource(bindingList, null);
        dataGridView1.DataSource = source;
        ready = true;
    }
}
```

Add method IoTLine()

```
private void IoTLine()
{
    List<double> yl = new List<double>();
    List<string> xl = new List<string>();
    foreach(var v in datasetIoT)
    {
        yl.Add(v.motor_speed);
        xl.Add(v.qtime);
    }
    chart1.Series["IoT"].Points.DataBindXY(xl, yl);
}
```

Add Method MLLine()

```
private void MLLine()
{
    List<double> yl = new List<double>();
    List<string> xl = new List<string>();
    foreach (var v in datasetIoT)
    {
        yl.Add(v.predict_speed);
        xl.Add(v.qtime);
    }
    chart1.Series["ML"].Points.DataBindXY(xl, yl);
}
```

Add method GetPrediction

```
private ModelOutput GetPrediction(Motor m)
{
    var motor = new ModelInput()
    {
        Ambient = m.ambient,
        Coolant = m.coolant,
        U_d = m.u_d,
        U_q = m.u_q,
        Motor_speed = m.motor_speed,
        Torque = m.torque,
        I_d = m.i_d,
        I_q = m.i_q,
        Pm = m.pm,
        Stator_yoke = m.stator_yoke,
        Stator_tooth = m.stator_tooth,
        Stator_winding = m.stator_winding
    };
    var predEngine = mlContext.Model.CreatePredictionEngine
        <ModelInput, ModelOutput>(mlModel);
    ModelOutput result = predEngine.Predict(motor);
    return result;
}
```

Add method FindDifference

```
private void FindDifference(float iot, float ml)
{
    var diff = Math.Abs(iot - ml);
    labelStatus.Text = $"Differnece = {diff} ";
    if(diff < DIFF)
    {
        labelStatus.BackColor = Color.Gainsboro;
        labelStatus.Text += " Motor speed is in normal parameters";
    }
    else
    {
        labelStatus.BackColor = Color.Orange;
        labelStatus.Text += " A L E R T !";
    }
}
```

Add method MyTimer_Tick

```
private async void MyTimer_Tick(object sender, EventArgs e)
{
    if(ready)
    {
        ready = false;
        await GetD2CMessage();
        IoTLine();
        MLLine();
    }
}
```

Add code to Form1_Load

```
private void Form1_Load(object sender, EventArgs e)
test
{
    mlModel = mlContext.Model.Load(modelPath, out var modelInputSchema);
    SetChartArea();
    myTimer.Enabled = true;
    myTimer.Interval = 3000;
    myTimer.Tick += MyTimer_Tick;

    var connectionString = new EventHubsConnectionStringBuilder(
        new Uri(s_eventHubsCompatibleEndpoint),
        s_eventHubsCompatiblePath,
        s_iotHubSasKeyName,
        s_iotHubSasKey);
    s_eventHubClient = EventHubClient.CreateFromConnectionString(
        connectionString.ToString());
    eventHubReceiver = s_eventHubClient.CreateReceiver(
        "$Default",
        "0",
        EventPosition.FromEnqueuedTime(DateTime.Now));
}
```

Evaluation Form

https://bit.ly/gfbiz-eval

คุณกำลังทำแบบประเมินผล สำหรับหลักสูตรใด *

- ○ Advanced ASP.NET Core API and MVC
- ○ Essential ASP.NET Core MVC
- ○ Mastering C# and .NET Framework
- ○ Essential OOP and Design Patterns .NET
- ○ Essential OOAD (object-oriented analysis and design)
- ◉ Option 6