In [ ]:
```
// london_weather.csv : https://www.kaggle.com/datasets/emmanuelfwerr/london-weather-data
// Install the Microsoft.ML packages we're going to use for ML.NET
#r "nuget:Microsoft.ML"
#r "nuget:Microsoft.Data.Analysis"
#r "nuget:Microsoft.ML.TimeSeries"
```

Installed Packages

- Microsoft.Data.Analysis, 0.21.1
- Microsoft.ML, 3.0.1
- Microsoft.ML.TimeSeries, 3.0.1

```
Loading extensions from `C:\Users\ORHANT\.nuget\packages\microsoft.data.analysis\0.21.1\interactive-extensions\dotnet\Microsoft.Data.Analysis.Interactive.dll`
```

In [ ]:
```
using Microsoft.ML;
using Microsoft.ML.Data;
```

In [ ]:
```
public class WeatherInput
{
    [LoadColumn(0)]
    public DateTime WeatherDate; // Date: yyyyMMdd
    [LoadColumn(1)]
    public float CloudCover; // in oktas
    [LoadColumn(2)]
    public float Sunshine; // in hours
    [LoadColumn(3)]
    public float GlobalRads; // Global radiation in Watt / square meter
    [LoadColumn(4)]
    public float MaxTemp; // Celsius
    [LoadColumn(5)]
    public float MeanTemp; // Celsius
    [LoadColumn(6)]
    public float MinTemp; // Celsius
    [LoadColumn(7)]
    public float Precipitation; // This is the precipitation in millimeters. This is what we want to predict
    [LoadColumn(8)]
    public float Pressure; // In Pascals
    // Ignore snow depth since that won't be known at time of prediction and is related to Precipitatio
}
```

In [ ]:
```
// Create an ML Context. We'll use this for all ML operations
var context = new MLContext(seed: 2024);
```

In [ ]:
```
// Load the data
var data = context.Data
.LoadFromTextFile<WeatherInput>("london_weather.csv"
, separatorChar: ','
, hasHeader: true
, allowQuoting: true);
```

In [ ]:
```
// View schema
data.Schema
```

▼ *[WeatherDate: DateTime, CloudCover: Single, Sunshine: Single, GlobalRads: Single, MaxTemp: Single, MeanTemp: Single, MinTemp: Single, Precipitation: Single, Pressure: Single]*

Count	9

(values)

index	value
0	▶ *WeatherDate: DateTime*
1	▶ *CloudCover: Single*
2	▶ *Sunshine: Single*
3	▶ *GlobalRads: Single*
4	▶ *MaxTemp: Single*
5	▶ *MeanTemp: Single*
6	▶ *MinTemp: Single*
7	▶ *Precipitation: Single*
8	▶ *Pressure: Single*

In [ ]:
```
// Preview data
data.Preview(3).RowView
```

index	value
0	▶ *9 columns*
1	▶ *9 columns*
2	▶ *9 columns*

In [ ]:
```
using Microsoft.Data.Analysis;
var df = data.ToDataFrame();
```

In [ ]:
```
df.Head(5)
```

index	WeatherDate	CloudCover	Sunshine	GlobalRads	MaxTemp	MeanTemp	MinTemp	Precipitation	Pressure
0	1979-01-01 00:00:00Z	2	7	52	2.3	-4.1	-7.5	0.4	101900
1	1979-01-02 00:00:00Z	6	1.7	27	1.6	-2.6	-7.5	0	102530
2	1979-01-03 00:00:00Z	5	0	13	1.3	-2.8	-7.2	0	102050
3	1979-01-04 00:00:00Z	8	0	13	-0.3	-2.6	-6.5	0	100840
4	1979-01-05 00:00:00Z	6	2	29	5.6	-0.8	-1.4	0	102250

In [ ]:
```
df.Info()
```

index	Info	WeatherDate	CloudCover	Sunshine	GlobalRads	MaxTemp	MeanTemp	MinTemp	Precipitation	Pressure
0	DataType	System.DateTime	System.Single	System.Single	System.Single	System.Single	System.Single	System.Single	System.Single	System.Single
1	Length (excluding null values)	100	100	100	100	100	100	100	100	100

In [ ]:
```
df.Description()
```

index	Description	WeatherDate	CloudCover	Sunshine	GlobalRads	MaxTemp	MeanTemp	MinTemp	Precipitation	Pressure
0	Length (excluding null values)	100	100	100	100	100	100	100	100	100
1	Max	<null>	8	10.7	194	16.1	11.8	9.4	28.5	104270
2	Min	<null>	0	0	13	-0.8	-4.1	-7.5	0	98130
3	Mean	<null>	5.72	2.305	60.74	6.463999	3.1069994	-0.082999796	2.4940004	100905.4

In [ ]:
```
// Split into train and test splits to detect overfitting
var split = context.Data.TrainTestSplit(data, testFraction: 0.2);
```

In [ ]:
```
var testSet = split.TestSet; // 20 % of the data (0.2)
```

In [ ]:
```
var trainSet = split.TrainSet; // 80 % of the data (0.8)
```

In [ ]:
```
var featureArray = new string[] {"CloudCover","Sunshine","GlobalRads","MaxTemp","MeanTemp","MinTemp","Pressure"};
```

In [ ]:
```
var processPipeline = context.Transforms.CopyColumns("Label", "Precipitation")
    .Append(context.Transforms.NormalizeMeanVariance("CloudCover", "CloudCover"))
    .Append(context.Transforms.NormalizeMeanVariance("Sunshine", "Sunshine"))
    .Append(context.Transforms.NormalizeMeanVariance("GlobalRads", "GlobalRads"))
    .Append(context.Transforms.NormalizeMeanVariance("MaxTemp", "MaxTemp"))
    .Append(context.Transforms.NormalizeMeanVariance("MeanTemp", "MeanTemp"))
    .Append(context.Transforms.NormalizeMeanVariance("MinTemp", "MinTemp"))
    .Append(context.Transforms.NormalizeMeanVariance("Pressure", "Pressure"))
    .Append(context.Transforms.Concatenate("Features", featureArray))
    .AppendCacheCheckpoint(context);
```

In [ ]:
```
// Some of the available options with the base ML.NET package are: https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers?view=ml-dotnet
//  * LbfgsPoissonRegression
//  * OnlineGradientDescent
//  * Sdca
// Select the algorithm we want
var trainer = context.Regression.Trainers.OnlineGradientDescent(labelColumnName: "Label", featureColumnName: "Features");
```

In [ ]:
```
// Generate a training pipeline based on the processing pipeline mixed with the trainer
var trainingPipeline = processPipeline.Append(trainer);
```

In [ ]:
```
// Train the model
var model = trainingPipeline.Fit(trainSet);
```

In [ ]:
```
var testResults = model.Transform(testSet);
var testMetrics = context.Regression.Evaluate(testResults, "Label", "Score");
testMetrics
```

▼ *Microsoft.ML.Data.RegressionMetrics*

MeanAbsoluteError	2.1189995376190063
MeanSquaredError	13.252040331744652
RootMeanSquaredError	3.6403351949710143
LossFunction	13.252040413413177
RSquared	0.06497927077698107

In [ ]:
```
// Saving the Model
// context.Model.Save(model, data.Schema, "Model.zip")
// Loading the Model
// var (loadedModel, loadedSchema) = context.Model.Load("Model.zip");
```

In [ ]:
```
public class WeatherPrediction
{
    [ColumnName("Score")]
    public float Precipitation; // This is the precipitation in millimeters. This is what we want to predict
}
```

In [ ]:
```
var predictionEngine = context.Model.CreatePredictionEngine<WeatherInput, WeatherPrediction>(model);
```

In [ ]:
```
var conditions = new WeatherInput {
    WeatherDate = DateTime.Now,
    CloudCover = 4f,
    Sunshine = 5.8f,
    GlobalRads = 50f,
    MaxTemp = 5.2f,
    MinTemp = 1.6f,
    MeanTemp = 3.7f,
    Pressure = 101170f,
    Precipitation = 0f, // Needed for the compiler, but its value is ignored
};
```

In [ ]:
```
var prediction = predictionEngine.Predict(conditions);
```

In [ ]:
```
prediction
```

▼ *Submission#24+WeatherPrediction*

Precipitation	1.0105166

In [ ]:
```
public class WeatherInputTimeSeriesForecastedPrediction
{
    public float[] ForecastedPrecipitation { get; set; }
    public float[] ConfidenceLowerBound { get; set; }
    public float[] ConfidenceUpperBound { get; set; }
}
```

In [ ]:
```
using Microsoft.ML.Transforms.TimeSeries;
```

In [ ]:
```
var length = (int)trainSet.ToDataFrame().Rows.Count;
var forecastEstimator = context.Forecasting.ForecastBySsa(
    outputColumnName: nameof(WeatherInputTimeSeriesForecastedPrediction.ForecastedPrecipitation),
    inputColumnName: nameof(WeatherInput.Precipitation),
    windowSize: 12,
    seriesLength: length,
    trainSize: length,
    horizon: 4,
    confidenceLevel: 0.95f,
    confidenceLowerBoundColumn: nameof(WeatherInputTimeSeriesForecastedPrediction.ConfidenceLowerBound),
    confidenceUpperBoundColumn: nameof(WeatherInputTimeSeriesForecastedPrediction.ConfidenceUpperBound));
```

In [ ]:
```
var forecastTransformerModel = forecastEstimator.Fit(trainSet);
```

In [ ]:
```
var forecastingEngine = forecastTransformerModel.CreateTimeSeriesEngine<WeatherInput, WeatherInputTimeSeriesForecastedPrediction>(context);
```

In [ ]:
```
var forecasts = forecastingEngine.Predict();
```

In [ ]:
```
forecasts.Display()
```

▼ *Submission#29+WeatherInputTimeSeriesForecastedPrediction*

ForecastedPrecipitation	[2.9159963, 2.937026, 2.9426816, 2.9395862]
ConfidenceLowerBound	[-5.3402095, -5.3204994, -5.316504, -5.3216367]
ConfidenceUpperBound	[11.172202, 11.194551, 11.201867, 11.200809]