# DAANY – .NET DAta ANalYtics library

A close up of a sign

Description automatically generated

## Introduction

Daany is .NET data analytics library written in C# language and it supposed to be a tool for data preparation, feature engineering and other kinds of data transformations prior to creating ml-ready data set. It is .NET based library and can be run on Windows Linux based distribution and Mac. It follows the .NET Standard 2.0.

Besides data analysis, the library implements a set of statistics or data science features e.g. time series decompositions, optimization performance parameters and similar.

The Daany project consists of four main components:

* Daany.DataFrame and
* Daany.Stats
* Daany.Math
* Daany.DataFrame.Ext

The main Daany component is Daany.DataFrame the DataFrame implementation for data analysis. It is much like Pandas but this component is not going to follow pandas implementation. Danny.DataFrame is suitable for doing data exploration and preparation with C# notebook. Daany.DataFrame doesn’t require any predefined class type in order to create or load data to DataFrame. All data are parsed internally in order to defined relevant value type of each column. The Daany.DataFrame implements set of powerful features for data manipulation, handling missing values, calculated columns, merging two or more data frame into one, and similar. It is handy for extracting its rows or columns as series of elements and put into the chart to visualizing the data.

Daany.Stat is a collection of statistics features e.g. time series decompositions, optimization, performance parameters and similar.

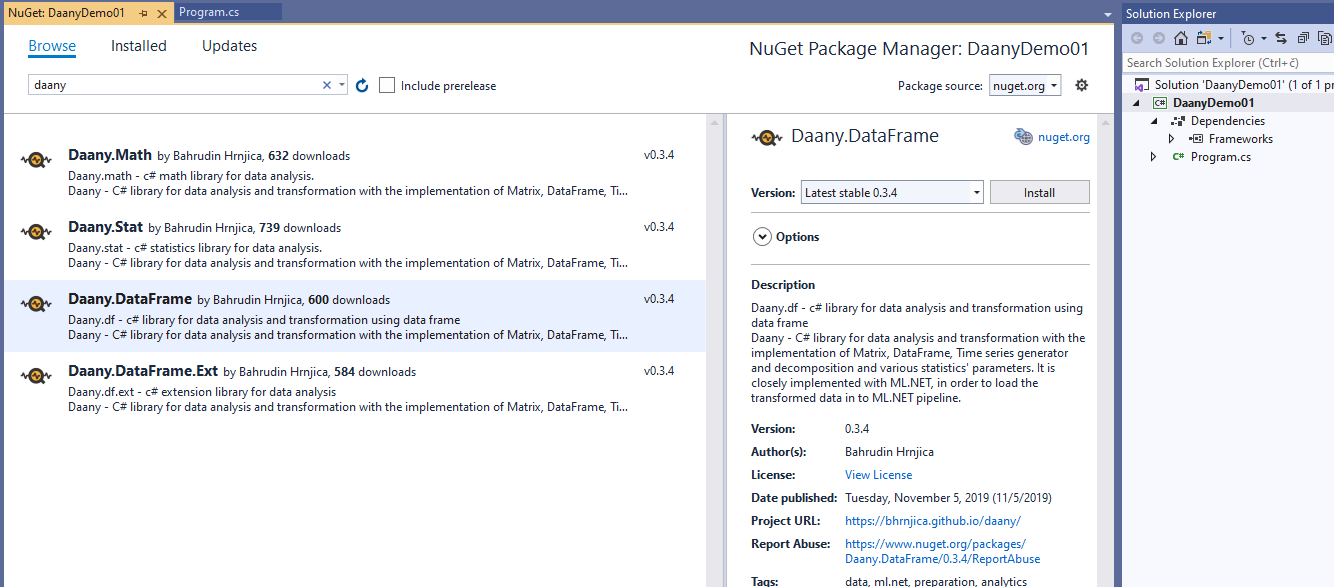
Daany.Math is a component with the implementation of od matrix and related linear algebra capabilities. It contains some implementation of other great open source projects. Daany.DataFrame.Ext contains extensions of DataFrame which are related to other projects mostly to ML.NET. The Daany.DataFrame should not be dependent on Ml.NET and other libraries. So any future DataFrame feature which depends on something other than Daany.Math, should be placed in Daany.Ext.

The project is developed as a need to have a set of data transformation features in one library while I work with machine learning. So I thought it might help others in the same situation. Currently, the library has pretty much data transformation features and might be your number one data transformation library on .NET platform.

# How to start with Daany

Daany is 100% .NET core and can be run on any platform .NET Core supports, from the Windows x86/x64 to Mac or Linux based OS. It can be used by Visual Studio or Visual Studio Code. Daany is a .NET component consisted of 4 NuGet packages, so the easiest way to start with it is to install the packages into your .NET application. Create or open your .NET application and open NuGet packages window within Visual Studio. Type Daany in browse edit box and hit enter. You can find four packages starting with Daany. You have few options to install the packages.

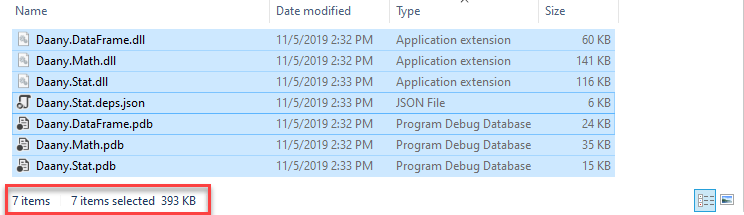
1. Install Daany.DataFrame – only. Use this option if you want only data analysis by using DataFrame. Once you click Install button, Daany.DataFrame and Daany.Math will be installed into your project app.
2. Install Daany.Stat package with time series decomposition and related statistics features.



Once you install the packages, you can start developing your app using Daany packages.

## Using Daany as assembly reference

Since Daany has no dependency on other library you can copy three dlls and add them as reference into your project.



In order to do so clone the project from <http://github.com/bhrnjica/daany> build it and copy Daany.DataFrame.dll, Daany.Math.dll and Daany.Stat.dll to your project as assembly references. Whole project is just 400 KB which is very small and handy. Currently Daany.DataFrame.Ext depends on ML.NET, and should be install as well.

## Namespaces in Daany

Daany project contains several namespaces for separating different implementation. The following list contains relevant namespaces:

* using Daany – data frame and related code implementation,
* using Daany.Ext – data frame extensions, used with dependency on third party library,
* using Daany.MathStuff – math related stuff implemented in Daany,
* using Daany.Optimizers – set of optimizers like SGD,
* using Daany.Stat – set of statistics implementations in the project.

# Working with Daany.DataFrame

Central part of Daany project is Daany.DataFrame an c# implementation of DataFrame. The DataFrame class is used for handling tabular data, especially for data preparation, feature engineering and analysis during development of machine learning models. The concept of DataFrame implementation is based on simplicity and .NET coding standard. DataFrame represents tabular data consisting of columns and rows. Each column has name and type and and each row has its index.

The following image shows a DataFrame structure

A picture containing indoor, map, computer, sky

Description automatically generated

The basic components of the DataFrame are:

* Column names – list of strings containing column names,
* Index label – list of object representing each row
* data – list of values in the DataFrame.
* missing values –

In order to create a DataFrame there are several options:

* from a list of values, by specifying column names and row count
* from a dictionary, letting keys be column names and values be column values,
* from text-based file, where each line represents row values.

## Three ways to create Daany.DataFrame

### Create DataFrame from a list of data.

Daany.DataFrame can be created by passing 1d list of data and column header. The following code shows such action.

//define a list of data

var lst = new List<object>() { 1, "Sarajevo", 77000, "BiH", true, 3.14, DateTime.Now.AddDays(-20),

2, "Seattle", 98101, "USA", false, 3.21, DateTime.Now.AddDays(-10),

3, "Berlin", 10115, "GER", false, 4.55, DateTime.Now.AddDays(-5),

};

//define column header for the DataFrame

var columns = new List<string>() { "ID", "City", "Zip Code", "IsHome", "Values", "Date" };

//create data frame with 3 rows and 7 columns

var df = new DataFrame(lst, columns);

### Create Data Frame from dictionary

Similarly Daany.DataFrame can be creted by pasing dictionary collection. The following code shows how to create DataFrame from the dictionary:

//define a dictionary of data

var dict = new Dictionary<string, List<object>>

{

{ "ID",new List<object>() { 1,2,3} },

{ "City",new List<object>() { "Sarajevo", "Seattle", "Berlin" } },

{ "Zip Code",new List<object>() { 71000,98101,10115 } },

{ "State",new List<object>() {"BiH","USA","GER" } },

{ "IsHome",new List<object>() { true, false, false} },

{ "Values",new List<object>() { 3.14, 3.21, 4.55 } },

{ "Date",new List<object>() { DateTime.Now.AddDays(-20) , DateTime.Now.AddDays(-10) , DateTime.Now.AddDays(-5) } },

};

//create data frame with 3 rows and 7 columns

var df = new DataFrame(dict);

### Create Data Frame by loading data from a file

By using static method DataFrame.FromCsv a user can create data frame object from the csv file. Otherwise data frame can be persisted on disk by calling static method DataFrame.ToCsv.The following code shows how to use ToCsv snd FromCSV:

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt", sep:',', dformat:"MM/dd/yyyy");

## Other ways to create a data frame

Besides above, the data frame can be created on other ways.

Create data frame from existing one. This method can be used in order to create new data frame with different numbers of columns. For example, an new data frame can be created from existing once by specifying specific columns:

//now create a new data frame with only three columns

var newDf = df["City", "Zip Code", "State"];

//check the size of the data frame

Assert.Equal(3, df.RowCount());

Assert.Equal(3, newDf.ColCount());

Also data frame can be created using Create method and passing tuples of existing and new column name. For example:

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt", sep: ',', names: null, dformat: "MM/dd/yyyy");

//now create a new data frame with three columns which can be renamed during createation

var newDf = df.Create(("City","Place"), ("Zip Code", null), ("State", "Country"));

//check the size of the data frame

Assert.Equal(3, newDf.RowCount());

Assert.Equal(new string[] { "Place", "Zip Code", "Country" }, newDf.Columns);

Assert.Equal(3, newDf.ColCount());

In case we want to remain the same column name, we pass null as second tuple item, or write the same name.

It is handy to create an empty data frame with specific column header. To do so we can use similar code like following:

var cols = new string[] { "Place", "Country", "Zip Code", "Values" };

//create empty data frame with 4 columns

var df = DataFrame.CreateEmpty(cols);

//check the size of the data frame

Assert.Equal(0, df.RowCount());

Assert.Equal(new string[] { "Place", "Country", "Zip Code", "Values" }, df.Columns);

Assert.Equal(4, df.ColCount());

Empty data frame is handy when we need to add rows from specific circumstances. Data frame is created almost in any performed operation like sorting, filtering, grouping, aggregation and similar.

## How to enumerate DataFrame

Enumeration of DataFrame means iterating the DataFram row by row. Daany.DataFrame provides three ways for enumeration.

* Strongly typed enumeration – it means that you have to provide a Class type during enumeration, as well as maping logic from dataframe row into class.This is done by providing callback mmaping method.
* Enumeration by dictionary – it means that each Data Frame row is returned as dictionary, where keys are column names and values are row array.
* Enumeration by list – it means the each data frame row is returned as list of data.

### Strongly typed enumeration

This enumeration is suitable when data frame need to be converted into list of custom types. Also, in order to convert Daany.DataFram into ML.NET IDataView you have to use typed enumeration. The following code shows how data set as previously loaded into Daany.DataFrame is converted into ML.NET DataView.

Assume we defined Iris class as follow (two foal and one string propreties):

class Iris

{

public float PetalArea { get; set; }

public float SepalArea { get; set; }

public string Species { get; set; }

}

Previously we created Daany.DataFrame from the file containing classic Iris dataset with: 5 columns: "sepal\_width", "petal\_width", "sepal\_height", "petal\_height", "species". We performed the transformation and defined two new columns: "SepalArea","PetalArea"

Now that we prepare the data for the machine learning, tha final stage would be to convert data frame into ML.NET DataVew collection. To do that we used the following code:

//convert DataFrame into strongly type list

var mlreadydataset = derivedDF.GetEnumerator<Iris>((oRow) =>

{

//convert row object array into Iris row

var prRow = new Iris();

prRow.SepalArea = Convert.ToSingle(oRow["SepalArea"]);

prRow.PetalArea = Convert.ToSingle(oRow["PetalArea"]);

prRow.Species = Convert.ToString(oRow["species"]);

//

return prRow;

});

//Load typed list into Ml.NET data pipeline

IDataView dataView = mlContext.Data.LoadFromEnumerable<Iris>(mlreadydataset);

As can be seen, GetEnumerator takes one argument oRow which is dictionary of the data frame row. The oRow is passed into delegate where we perform conversion. Once we have list of Iris object, we can load them into ML.NET pipeline.

### Enumeration by dictionary

In order to enumerate DataFrame by dictionary call GetEnumeration method. The following code uses previously loaded data frame and perform Linq operators against it.

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt");

//get second data frame row

//row2 is a dictionary with column names as keys

var row2 = df.GetEnumerator().Skip(1).First();

//check some data from the second row

Assert.Equal("Seattle", row2["City"]);

Assert.Equal("USA", row2["State"]);

Assert.Equal(3.21f, row2["Values"]);

### Enumeration by list

Similarly we can enumerate data fram by returning the list or array as row. This method can be use when you need performance, otherwise use previous method. The following code shows how to use enumeration by list.

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt");

//get second data frame row

//row2 is a dictionary with column names as keys

var row2 = df.GetRowEnumerator().FirstOrDefault();

//check some data from the second row

Assert.Equal("Sarajevo", row2[1]);

Assert.Equal("BiH", row2[3]);

Assert.Equal(3.14f, row2[5]);

# Selecting data in Daany data frame

Accessing data in data frame can be achieved in many ways. From data frame it can be selected one column or two or more columns by using column names. Row in data frame is selected by using zero based index.

### Column selection

This code sample select two columns separately from data frame, and convert them into array and list.

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt");

//select one column from the data frame

var cities = df["City"].ToArray();

var zipCodes = df["Zip Code"].ToList();

//check for values

Assert.Equal(3, cities.Length);

Assert.Equal("Sarajevo", cities[0]);

Assert.Equal("Seattle", cities[1]);

Assert.Equal("Berlin", cities[2]);

//check for values

Assert.Equal(3, zipCodes.Count);

Assert.Equal(71000, zipCodes[0]);

Assert.Equal(98101, zipCodes[1]);

Assert.Equal(10115, zipCodes[2]);

### Row Selection

Selecting more than one column at one returns data frame. The following code creates new data frma from selected columns:

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt");

//select one column from the data frame

var citiesDf = df["City", "Zip Code"];

//check for values

Assert.Equal(3, citiesDf.RowCount());

Assert.Equal(2, citiesDf.ColCount());

Select data by rows returns whole row from the data frame. The folloing code returns third rows from the existing data frame.

//select third row from data frame

//3, "Berlin", 10115, "GER", false, 4.55, DateTime.Now.AddDays(-5)

var row = df[2].ToArray();

//check for values

Assert.Equal(7, row.Length);

Assert.Equal(2, row[0]);

Assert.Equal("Berlin", row[1]);

### Data selection

Selecting data is achieved by using zero-based position of row and column, moreover data can be accessed by using column name and rows position. The following code select “City” from the third row:

//create data frame with 3 rows and 7 columns

var df = DataFrame.FromCsv($"{rootfolder}/simple\_data\_frame.txt");

//select city from the third row

var city = df[2,1];

var city1 = df["City", 2];

//check for values

Assert.Equal(city,city1);

Assert.Equal("Berlin", city1);

## Operations in Daany.DataFrame

The following operation are supported:

* AddColumns
* AddRows
* AddCalculatedColumns
* Aggregate
* Drop
* DropNA
* FillNA
* Filter
* GroupBy
* InsertColumn
* Rename
* SortBy
* SortByDescending
* MissingValues
* Rolling
* Select