Excel to PDF .Net

(Multi-platform .Net library)

SautinSoft

Linux development manual

Table of Contents

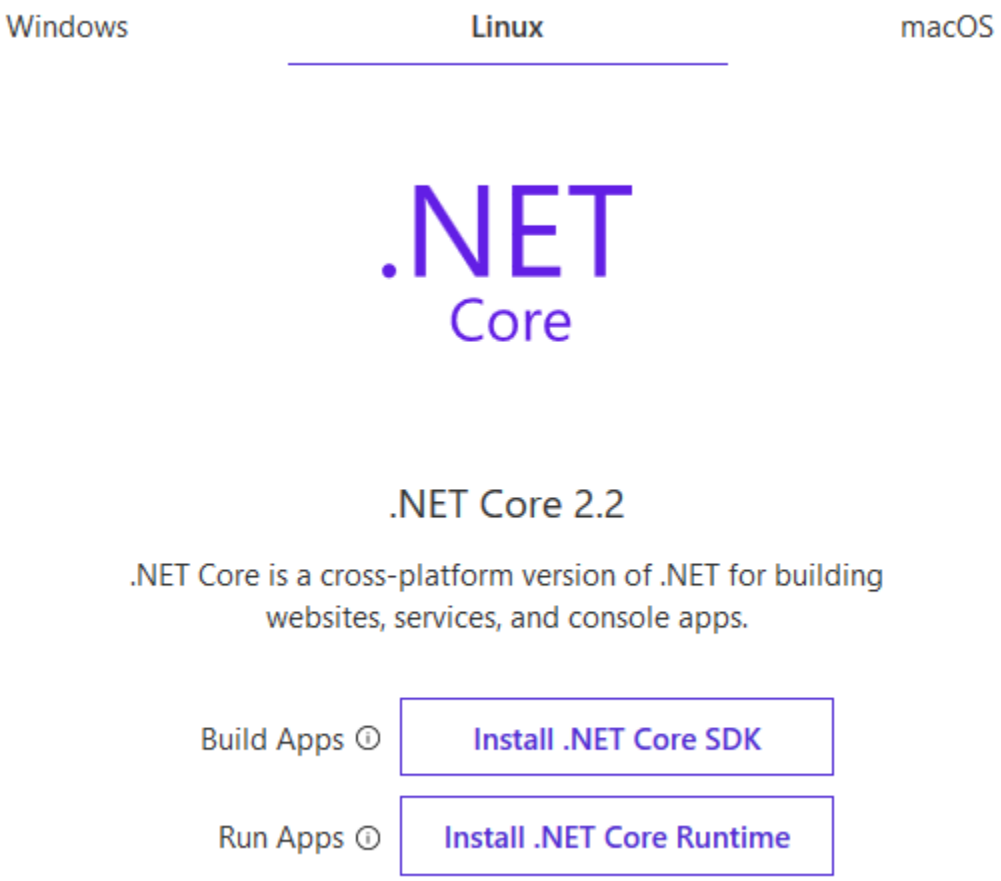
1. Preparing environment

In order to build multi-platform applications using .NET Core on Linux, the first steps are for installing in our Linux machine the required tools.

We need to install .NET Core SDK from Microsoft and to allow us to develop easier, we will install an advance editor with a lot of features, Visual Studio Code from Microsoft.

Both installations are very easy and the detailed description can be found by these two links:

Install .NET Core SDK for Linux.



Install VS Code for Linux.

Once installed VS Code, you need to install a C# extension to facilitate us to code and debugging:

Install C# extension.

1.1. Check the installed Fonts availability

Check that the directory with fonts "/usr/share/fonts/truetype" is exist.

Also check that it contains *.ttf files.

If you don't see this folder, make these steps:

1. Download the archive with *.ttf fonts: https://sautinsoft.com/components/fonts.tar
2. Uncompress the downloaded font's archive to a directory and add it to the font path, a list of directories containing fonts:

   ```
   # tar xvzf
   ```

3. Create a directory for new fonts

   ```
   # mkdir /usr/share/fonts/truetype
   ```

4. Move the uncompressed font files to the new font directory

   ```
   # mv *.ttf /usr/share/fonts/truetype
   ```

5. Navigate to the font directory

   ```
   # cd /usr/share/fonts/truetype
   ```

6. Create fonts.scale and fonts.dir

   ```
   # mkfontscale && mkfontdir
   # fc-cache
   ```

7. Add the new font directory to the X11 font path

   ```
   # chkfontpath --add /usr/share/fonts/truetype
   ```

8. Restart X font server

   ```
   # /etc/rc.d/init.d/xfs restart
   ```

You can verify the successful addition of the new path by running `chkfontpath` command or by listing X font server's /etc/X11/XF86Config file.

If you do not have root access, copy the *.ttf to ~/.fonts directory instead.

With these steps, we will ready to start developing.

In next paragraphs we will explain in detail how to create simple console application. All of them are based on this VS Code guide:

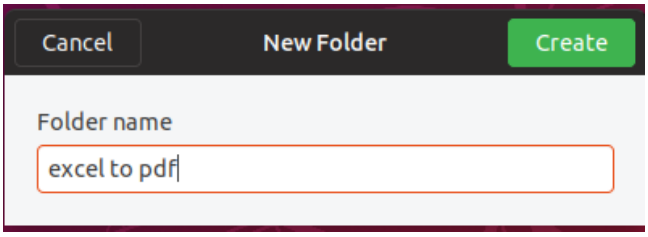
Get Started with C# and Visual Studio Code

Not only is possible to create .NET Core applications that will run on Linux using Linux as a developing platform. It is also possible to create it using a Windows machine and any modern Visual Studio version, as Microsoft Visual Studio Community 2017.

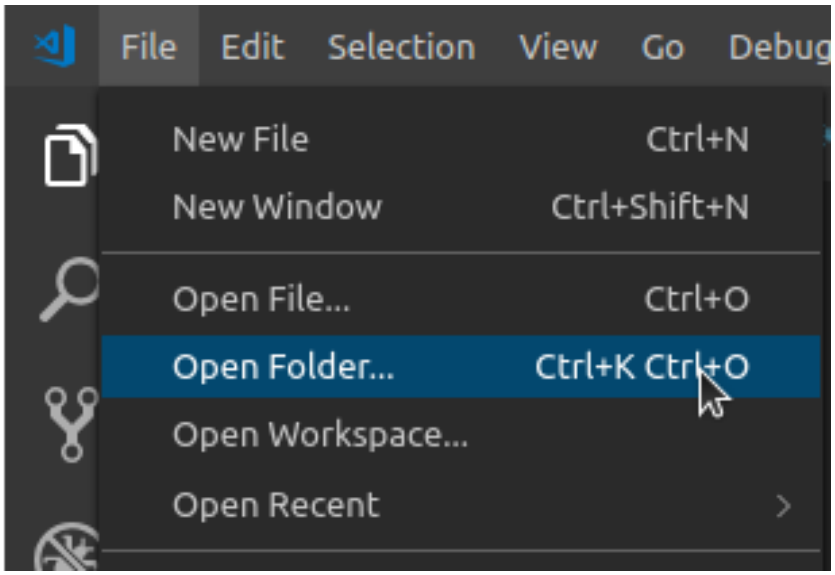
2. Creating “Convert XLSX to PDF” application

Create a new folder in your Linux machine with the name ***excel to pdf.***

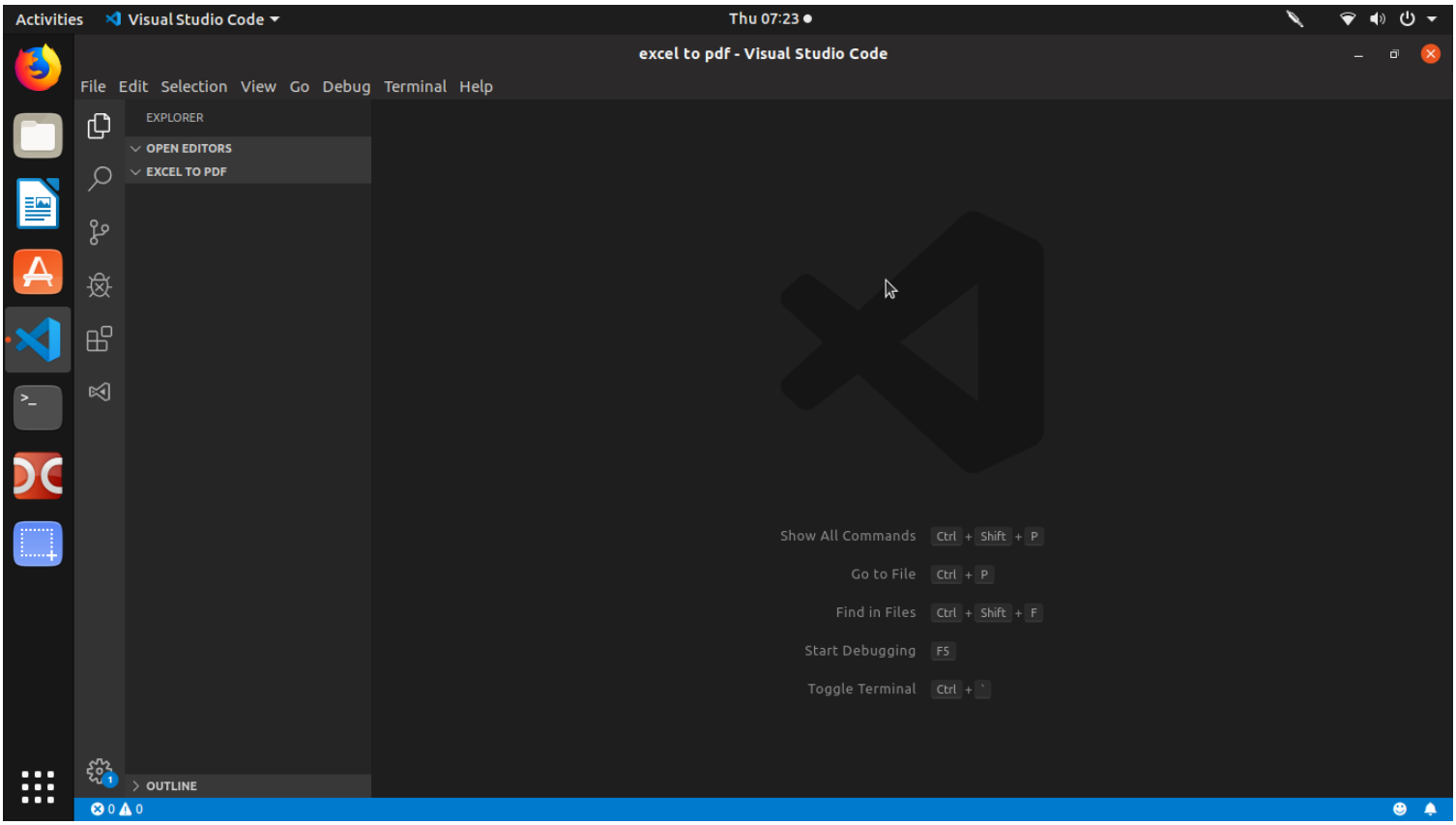
For example, let’s create the folder “***excel to pdf***” on Desktop (Right click-> New Folder):



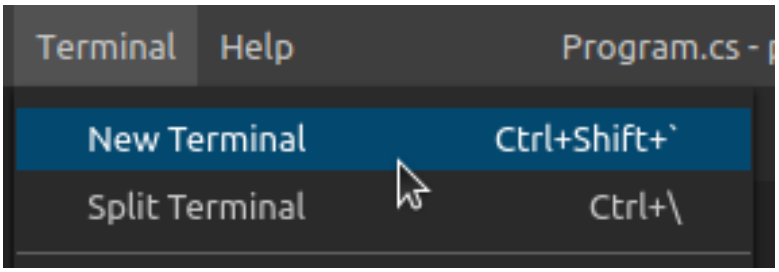
Open VS Code and click in the menu ***File->Open Folder***. From the dialog, open the folder you’ve created previously:



Next you will see the similar screen:



Now, open the integrated console – the Terminal: follow to the menu ***Terminal -> New Terminal*** (or press Ctrl+Shift+'):



Create a new console application, using ***dotnet*** command.

Type this command in the Terminal console: ***dotnet new console***

```
  Restore completed in 445.88 ms for /home/jorgen/Desktop/excel to pdf/excel to pdf.csproj.

Restore succeeded.

jorgen@jorgen-linux:~/Desktop/excel to pdf$
```

A new simple ***Hello world!*** console application has been created. To execute it, type this command: ***dotnet run***

```
Hello World!
jorgen@jorgen-linux:~/Desktop/excel to pdf$
```

You can see the typical “Hello world!” message.

Now we are going to convert this simple application into something more interesting.

We’ll transform it into an application that will convert a xlsx file to a pdf file.

First of all, we need to add the package reference to the ***sautinsoft.exceltopdf*** assembly using Nuget.

In order to do it, follow to the ***Explorer*** and open project file “***excel to pdf.csproj”***

Add these lines into the file ”***excel to pdf.csproj”***:

```
<PackageReference Include="Pkcs11Interop" Version="5.1.2" />
<PackageReference Include="Portable.BouncyCastle" Version="1.9.0" />
<PackageReference Include="SkiaSharp" Version="2.88.7" />
<PackageReference Include="Svg.Skia" Version="1.0.0.18" />
<PackageReference Include="System.IO.Packaging" Version="4.4.0" />
<PackageReference Include="System.Text.Encoding.CodePages" Version="4.5.0" />
```

It’s the reference to ***sautinsoft.exceltopdf*** package from Nuget.

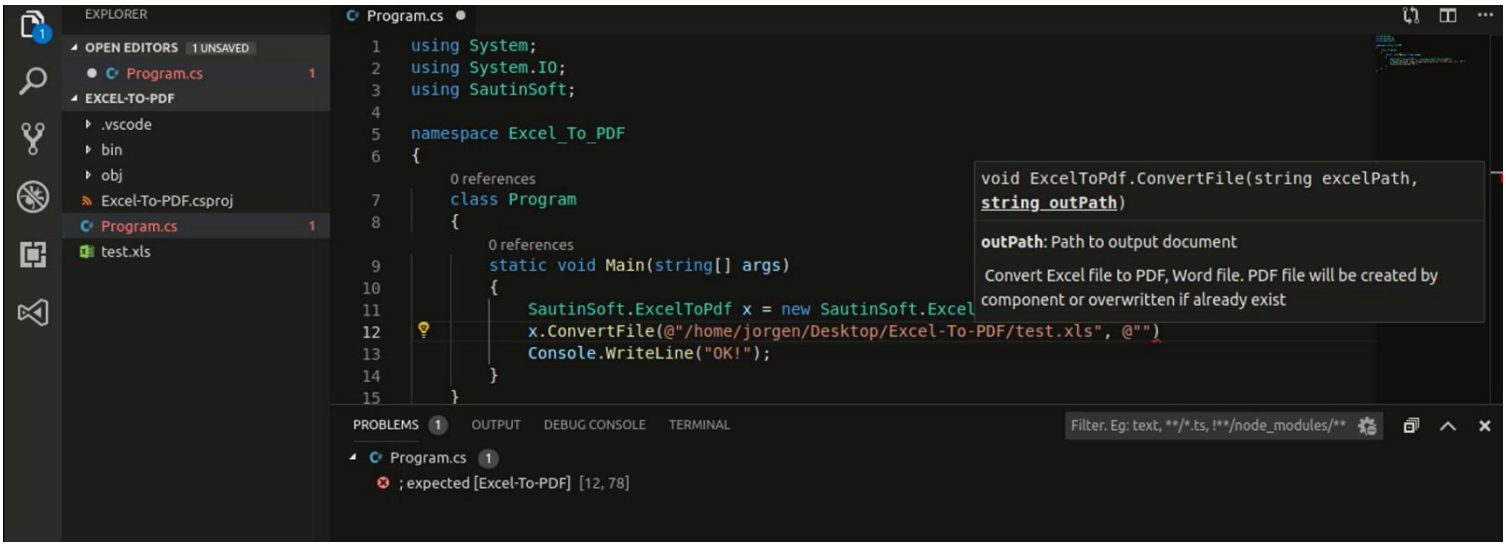
At the moment of writing this manual, the latest version of ***sautinsoft.exceltopdf*** was 5.2.3.5. But you may specify the latest version, to know what is the latest, follow:

https://www.nuget.org/packages/sautinsoft.exceltopdf/

At once as we’ve added the package reference, we have to save the “***excel to pdf.csproj”*** and restore the added package.

Good, now our application has the reference to ***sautinsoft.exceltopdf*** package and we can write the code to convert xlsx to pdf and other formats.

Follow to the ***Explorer***, open the ***Program.cs***, remove all the code and type the new:

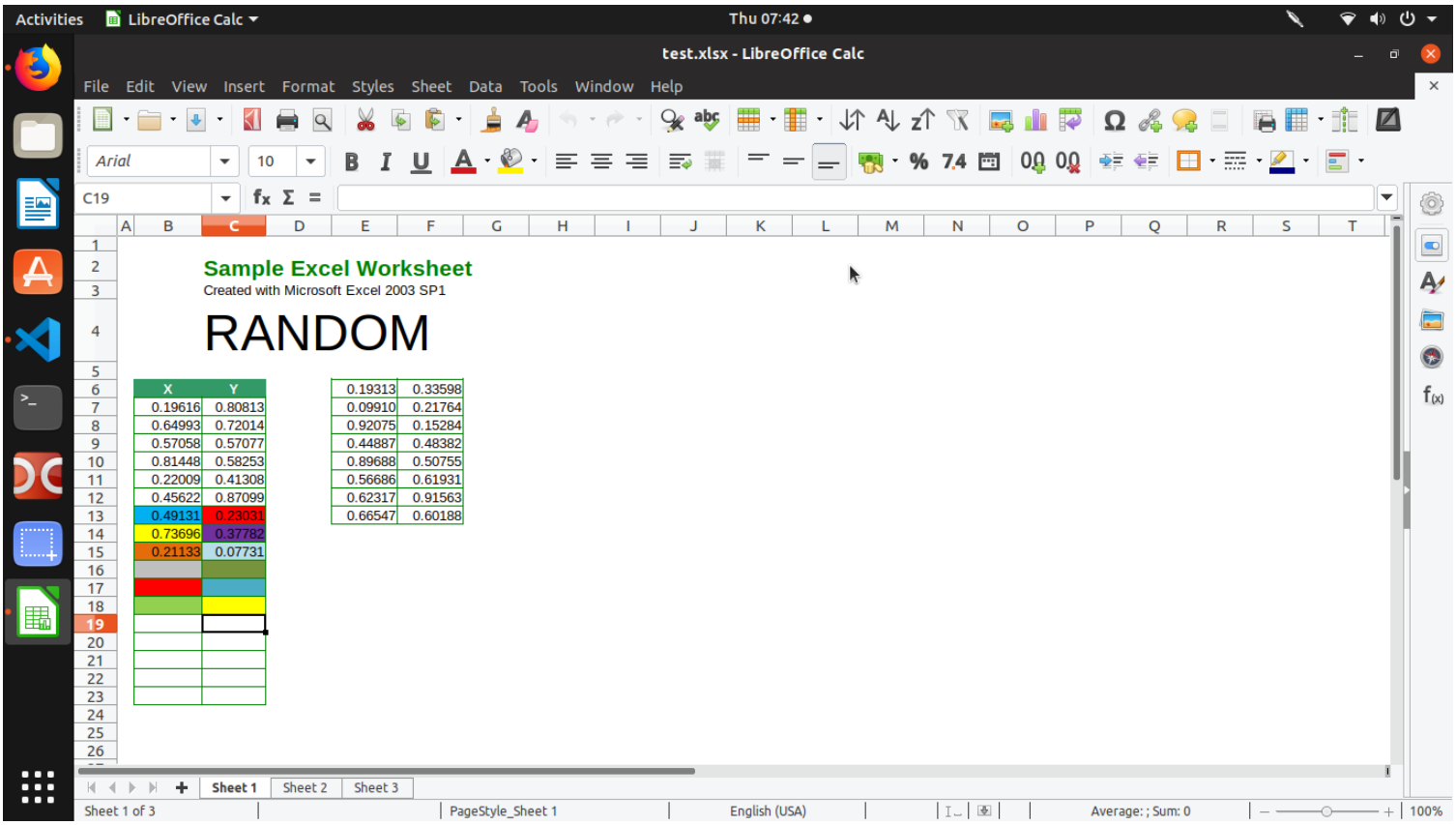


The new code:

```
using System;
using System.IO;
using SautinSoft;

namespace Excel_to_PDF
{
	class Program
	{
		static void Main(string[] args)
		{
		ExcelToPdf x = new ExcelToPdf();
		x.PageStyle.PageSize.Letter();

		// Set PDF as output format.
		x.OutputFormat = SautinSoft.ExcelToPdf.eOutputFormat.Pdf;

		// Let's convert only 1st sheet.
		x.Sheets.Custom(new int[] { 1 });

		string excelFile = @"/home/jorgen/Desktop/test.xlsx";
		string pdfFile = Path.ChangeExtension(excelFile, ".pdf");

		try
		{
		    x.ConvertFile(excelFile, pdfFile);
		    System.Diagnostics.Process.Start(pdfFile);
		}
		catch (Exception ex)
		{
		    Console.WriteLine(ex.Message);
		    Console.ReadLine();
		}
	}
}
```

To make tests, we need an input XLSX document. For our tests, let’s place a PDF file with the name “test.pdf” at the Desktop.

If we open this file in the default Excel Viewer, we'll its contents:



Launch our application and convert the "test.xslx" into "test.pdf", type the command:

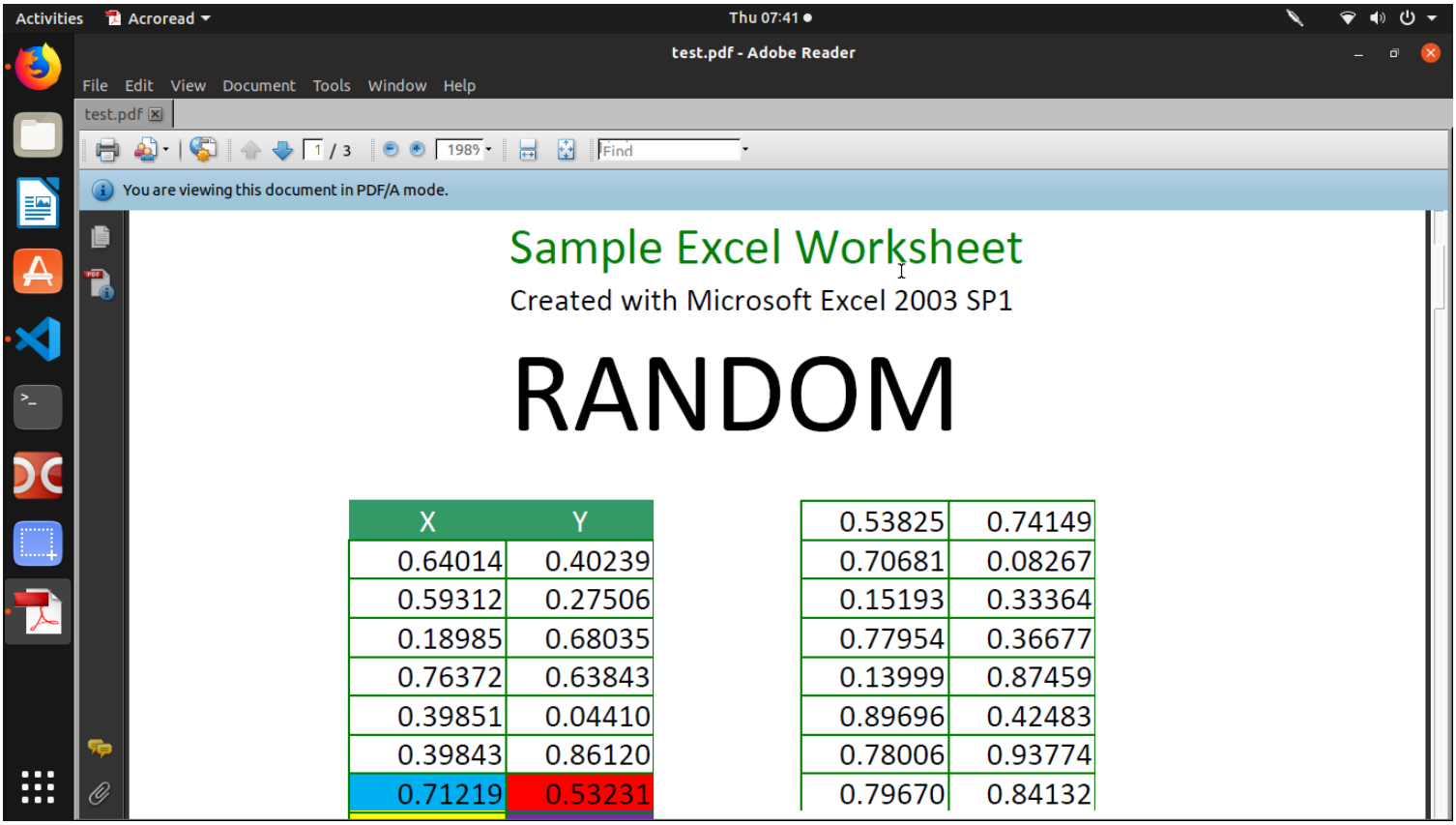
dotnet run

```
  Restore completed in 445.88 ms for /home/jorgen/Desktop/excel to pdf/excel to pdf.csproj.

Restore succeeded.

jorgen@jorgen-linux:~/Desktop/excel to pdf$
```

If you see the message "Converting successfully!", everything is fine and we can check the result produced by the Excel to PDF .Net library.

The new file “test.pdf” has to appear on the Desktop:

Open the result in PDF Viewer:



Well done! You have created the “Excel to PDF” application under Linux!

If you have any troubles or need extra code, or help, don’t hesitate to ask our SautinSoft Team at support@sautinsoft.com.