

Excel .Net

(Multi-platform .Net library)

[SautinSoft](#)

Linux development manual

Table of Contents

1. Preparing environment	2
1.1. Check the installed Fonts availability.....	3
2. Creating "Convert XLSX to PDF" application.....	5

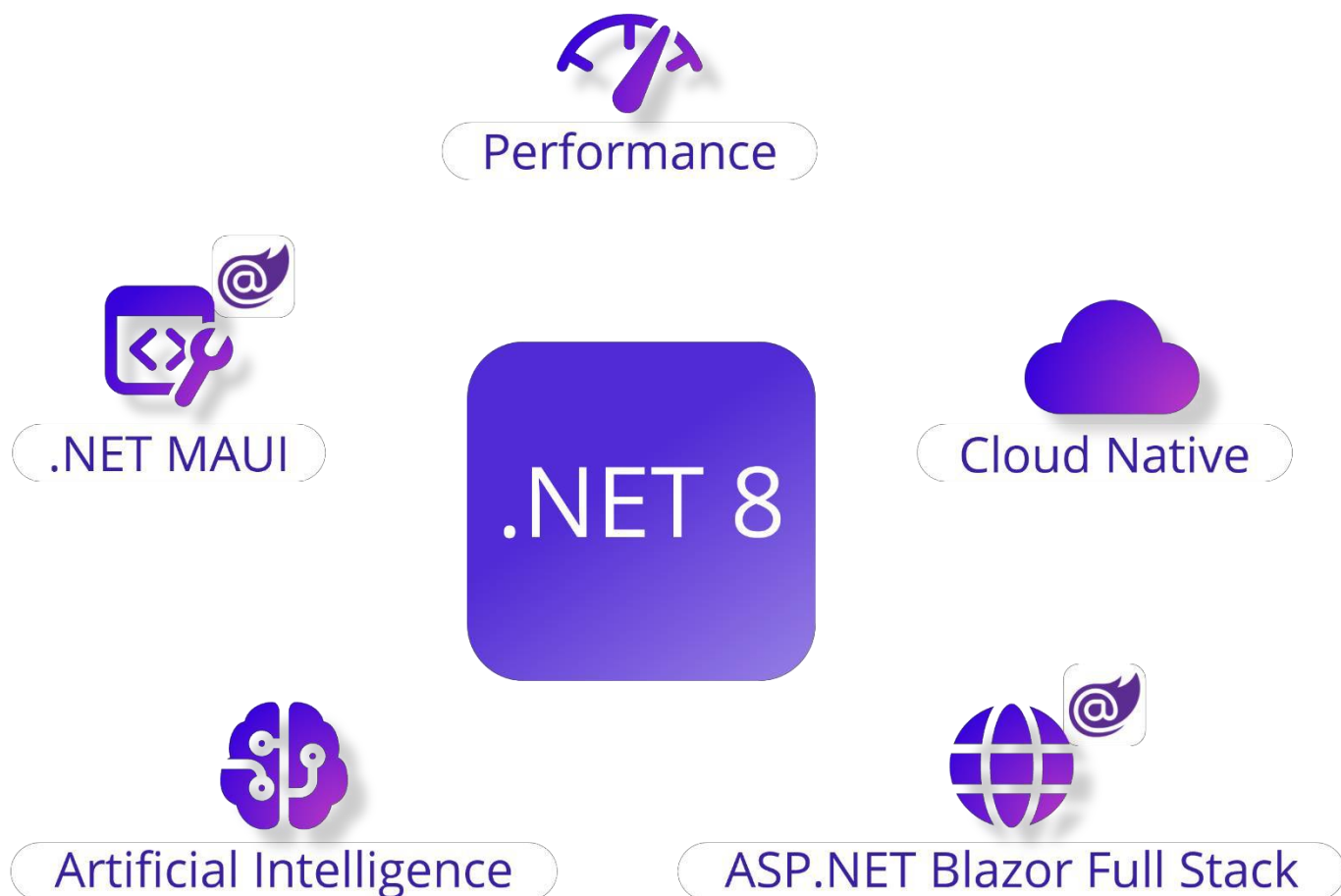
1. Preparing environment

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

We need to install .NET 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:

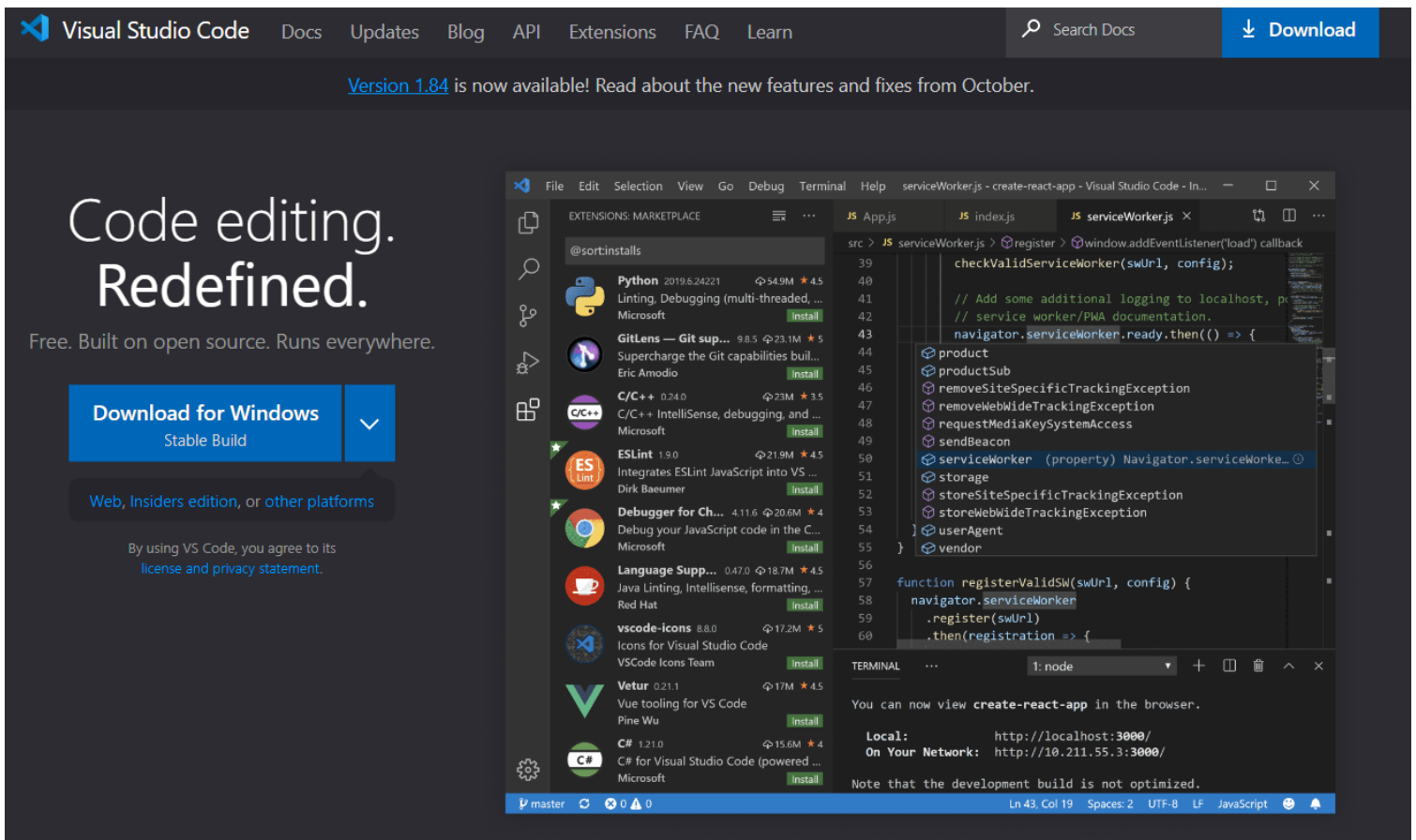
[Install .NET SDK for Linux.](#)



[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, you may install "Microsoft TrueType core fonts" using terminal and command:

```
$ sudo apt install ttf-mscorefonts-installer
```

```
linuxconfig@linuxconfig-org: ~  
All done, no errors.  
Extracting cabinet: /var/lib/update-notifier/package-data-downloads/partial/verdan32.exe  
  extracting fontinst.exe  
  extracting fontinst.inf  
  extracting Verdanab.TTF  
  extracting Verdanai.TTF  
  extracting Verdanz.TTF  
  extracting Verdana.TTF  
  
All done, no errors.  
Extracting cabinet: /var/lib/update-notifier/package-data-downloads/partial/webdin32.exe  
  extracting fontinst.exe  
  extracting Webdings.TTF  
  extracting fontinst.inf  
  extracting Licen.TXT  
  
All done, no errors.  
All fonts downloaded and installed.  
Processing triggers for man-db (2.9.0-2) ...  
Processing triggers for fontconfig (2.13.1-2ubuntu2) ...  
linuxconfig@linuxconfig-org:~$
```

Read more about [TrueType Fonts and "How to install Microsoft fonts, How to update fonts cache files, How to confirm new fonts installation"](#).

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

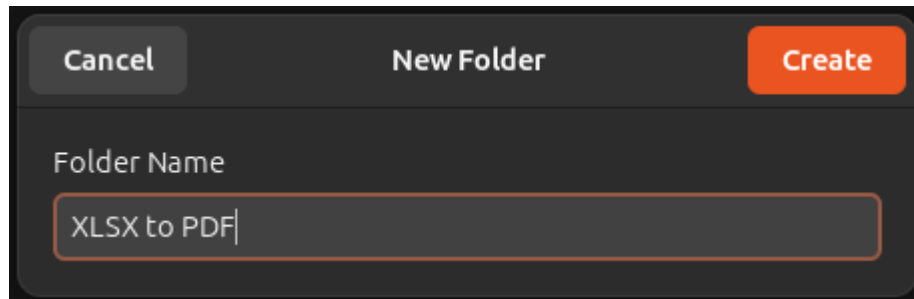
[Get Started with C# and Visual Studio Code](#)

Not only is possible to create .NET 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 2022.

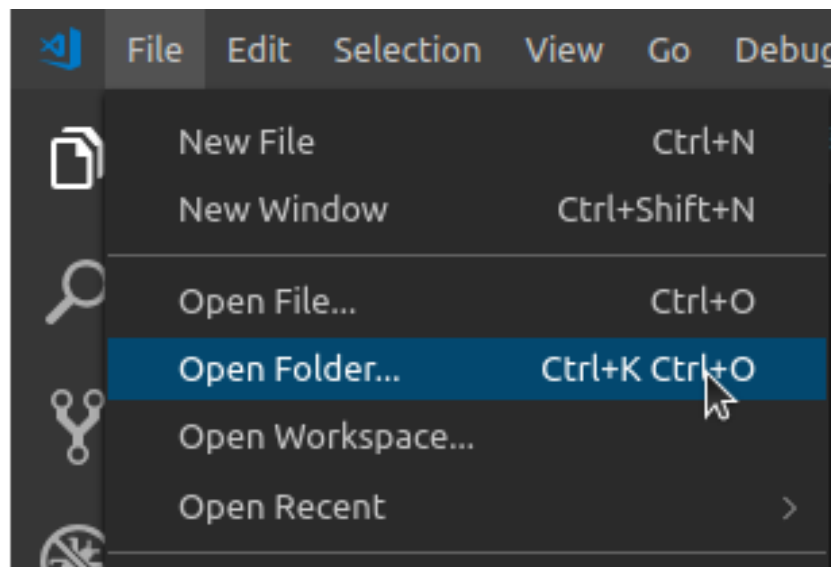
2. Creating “Convert XLSX to PDF” application

Create a new folder in your Linux machine with the name ***XLSX to PDF***.

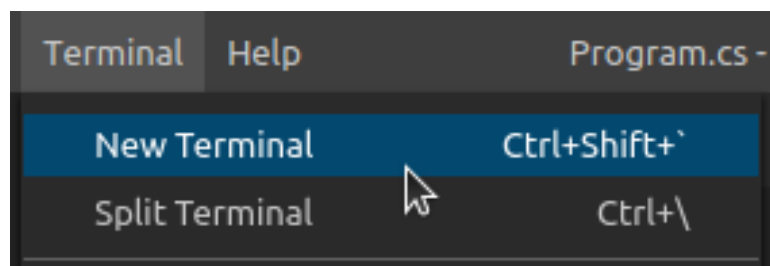
For example, let’s create the folder “***XLSX to PDF***” on Desktop (Right click-> New Folder):



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



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**

```
● alex@Linux2:~/Desktop/XLSX to PDF$ dotnet new console
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring /home/alex/Desktop/XLSX to PDF/XLSX to PDF.csproj:
  Determining projects to restore...
  Restored /home/alex/Desktop/XLSX to PDF/XLSX to PDF.csproj (in 1.55 sec).
Restore succeeded.

○ alex@Linux2:~/Desktop/XLSX to PDF$
```

Now we are going to modify this simple application 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 **"XLSX to PDF.csproj"**:

```
1  <Project Sdk="Microsoft.NET.Sdk">
2    <PropertyGroup>
3      <PackageId>Excel .Net</PackageId>
4      <Authors>Alex Alikin</Authors>
5      <Company>SautinSoft</Company>
6      <OutputType>Exe</OutputType>
7      <TargetFramework>net8.0</TargetFramework>
8    </PropertyGroup>
9    <ItemGroup>
10   <PackageReference Include="Sautinsoft.Excel" Version="*" />
11   <PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.7" />
12   <PackageReference Include="SkiaSharp.NativeAssets.macOS" Version="2.88.7" />
13   <PackageReference Include="SkiaSharp.HarfBuzz" Version="2.88.7" />
14   <PackageReference Include="HarfBuzzSharp.NativeAssets.Linux" Version="2.88.7" />
15   </ItemGroup>
16 </Project>
```

Add these lines into the file **"XLSX to PDF.csproj"**:

```
<ItemGroup>
<PackageReference Include="Sautinsoft.Excel" Version="*" />

<PackageReference Include="SkiaSharp.NativeAssets.Linux" Version="2.88.7" />

<PackageReference Include="SkiaSharp.NativeAssets.macOS" Version="2.88.7" />

<PackageReference Include="SkiaSharp.HarfBuzz" Version="2.88.7" />
```

```
<PackageReference Include="HarfBuzzSharp.NativeAssets.Linux" Version="2.88.7" />
```

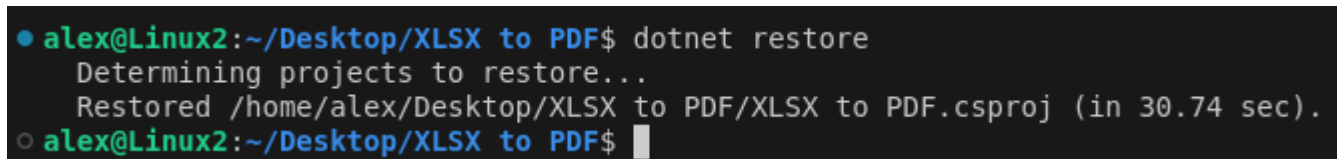
```
</ItemGroup>
```

The first reference installs the latest version **sautinsoft.excel** package from Nuget.

The second reference installs the **SkiaSharp.NativeAssets.Linux** package, which adds 2D graphics to .Net applications for Linux.

At once as we've added the package references, we have to save the "**XLSX to PDF.csproj**" and restore the added packages.

Follow to the **Terminal** and type the command: **dotnet restore**



```
● alex@Linux2:~/Desktop/XLSX to PDF$ dotnet restore
  Determining projects to restore...
  Restored /home/alex/Desktop/XLSX to PDF/XLSX to PDF.csproj (in 30.74 sec).
○ alex@Linux2:~/Desktop/XLSX to PDF$
```

Good, now our application has all the references and we can write the code to convert XLSX to PDF.

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

Sample.cs > Sample > Main

```
1  using System;
2  using System.IO;
3  using SautinSoft.Excel;
4
5  namespace Sample
6  {
7      0 references
8      class Sample
9      {
10         0 references
11         static void Main(string[] args)
12         {
13             // Before starting, we recommend to get a free 100-day key:
14             // https://sautinsoft.com/start-for-free/
15
16             // Apply the key here:
17             // SautinSoft.Excel.SetLicense("...");
18
19             string excelFile = @"/home/developer/Downloads/test.xlsx";
20             string pdfFile = @"/home/developer/Downloads/test_outputd.pdf";
21             ExcelDocument excelDocument = ExcelDocument.Load(excelFile);
22             excelDocument.Save(pdfFile, new PdfSaveOptions());
23         }
24     }
25 }
```

The code:

```
using System;
```

```
using System.IO;
```

```
using SautinSoft.Excel;
```

```
namespace Sample
```

```
{
```

```
    class Sample
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```



```
// Before starting, we recommend to get a free 100-day key:

// https://sautinsoft.com/start-for-free/

// Apply the key here:

// SautinSoft.Excel.SetLicense("...");


string excelFile = @"/home/developer/Downloads/test.xlsx";

string pdfFile = @"/home/developer/Downloads/test_outputc.pdf";

ExcelDocument excelDocument = ExcelDocument.Load(excelFile);

excelDocument.Save(pdfFile, new PdfSaveOptions());

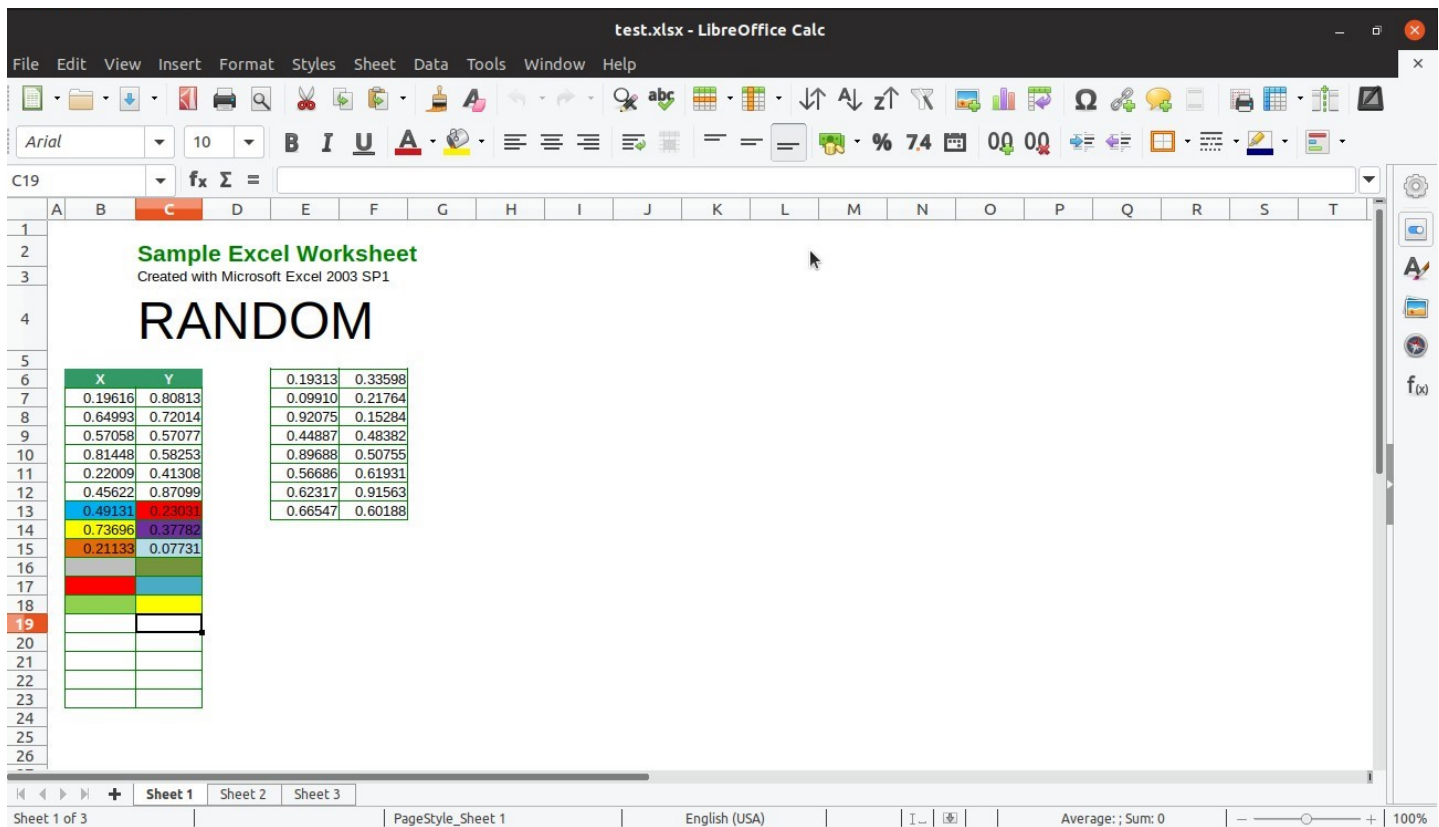
}

}
```

To make tests, we need an input XLSX document. For our tests, let's place a XLSX file with the name "test.xlsx" at the Desktop.



If we open this file, we'll see its contents:



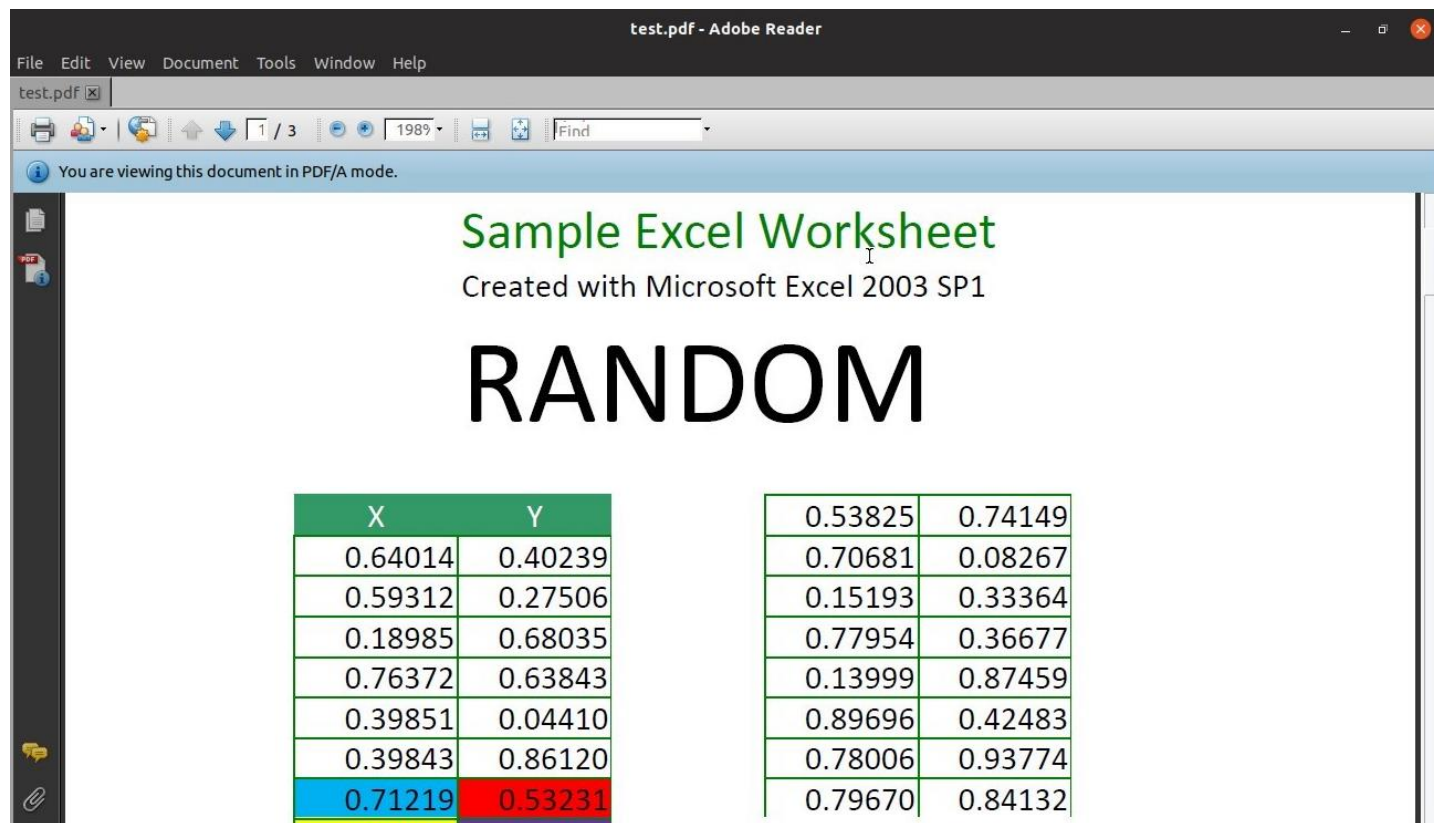
Launch our application and convert the "test.xlsx" into "test.pdf", type the command: ***dotnet run***

If you don't see any exceptions, everything is fine and we can check the result produced by The [Excel .Net](#) library.

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



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



Well done! You have created the "XLSX 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!