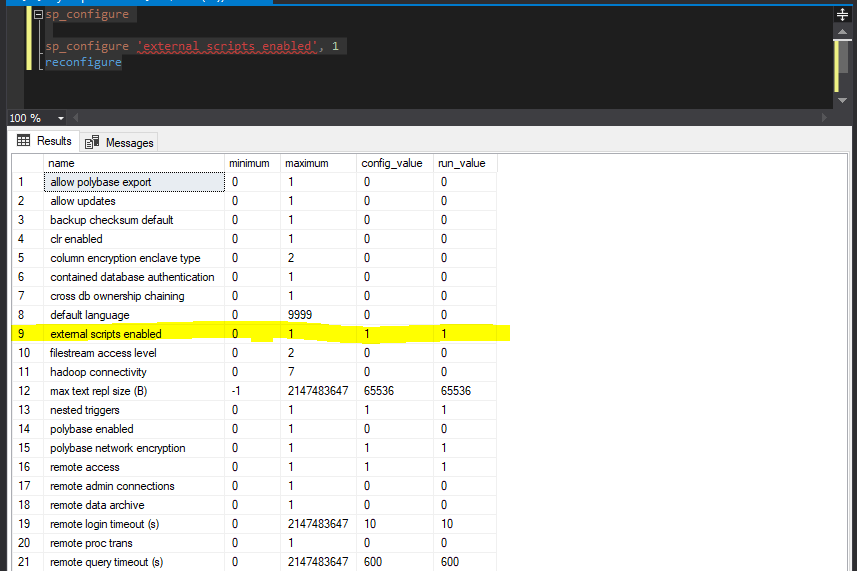
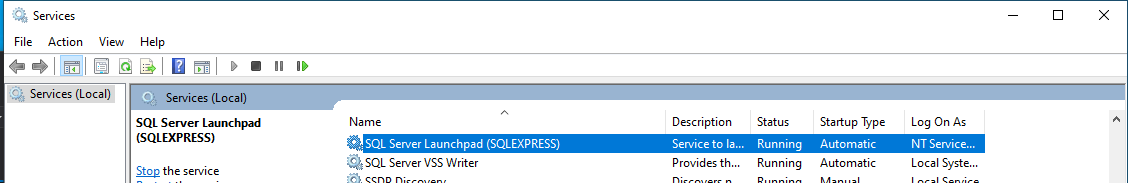
## Configure Sql Server for running (Python) scripts



|  |
| --- |
| sp\_configure  sp\_configure 'external scripts enabled', 1  reconfigure |

Check that the service is running:

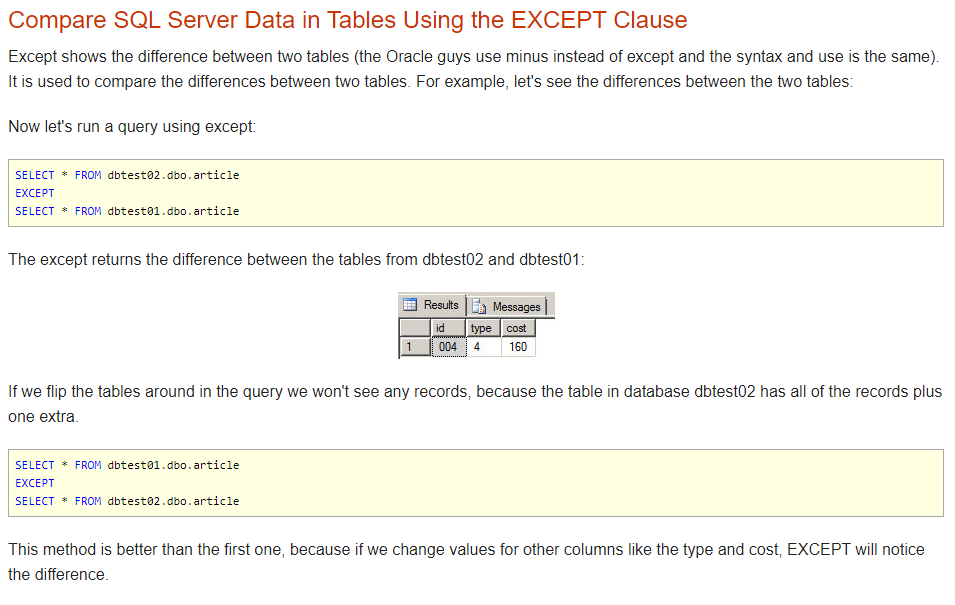


## Comparision of SQL Server Tables

Link: <https://www.mssqltips.com/sqlservertip/2779/ways-to-compare-and-find-differences-for-sql-server-tables-and-data/>

Article on FileSystem Object FSO and OLE Automation:

<https://www.red-gate.com/simple-talk/sql/t-sql-programming/reading-and-writing-files-in-sql-server-using-t-sql/>



## The Curse and Blessings of Dynamic SQL

<http://www.sommarskog.se/dynamic_sql.html>

A trap with long SQL Strings

The **@sql** variable itself can serve for the task, and in many cases it comes naturally, because you add to the **@sql** string piece by piece. In this example it could be like this:

DECLARE @sql nvarchar(MAX) = '',

@dbname sysname = 'tempdb'

SELECT @sql = @sql + N'

But there are two new traps hiding here. The first is one you would notice quickly. If you fail to initialise **@sql** to the empty string, the entire string will be NULL, and nothing would be executed. The other trap is that you (or someone else) may be tempted to introduce the shortcut operator +=:

DECLARE @sql nvarchar(MAX) = '',

@dbname sysname = 'NorthDynamic'

SELECT @sql += N'

This will bring back the syntax error, because on the right-hand side of the += operator, there are now only short strings, and thus there is no implicit conversion to **nvarchar(MAX)**. For this reason, adding

cast('' AS nvarchar(MAX)) + N'

is a good safety precaution.

## Dynamic Database and Server Names

Many people do as in this example:

DECLARE @dbname sysname = 'NorthDynamic',

@sql nvarchar(MAX)

SELECT @sql = 'USE ' + quotename(@dbname) + '

SELECT \* FROM dbo.Customers WHERE CustomerID = @custid'

PRINT @sql

EXEC sp\_executesql @sql, N'@custid nchar(5)', N'ALFKI'

That works, but there is a different way that I find cleaner:

DECLARE @dbname sysname = 'NorthDynamic',

@sql nvarchar(MAX),

@sp\_executesql nvarchar(200)

SELECT @sp\_executesql = quotename(@dbname) + '.sys.sp\_executesql'

SELECT @sql = N'SELECT \* FROM dbo.Customers WHERE CustomerID = @custid'

PRINT @sql

EXEC @sp\_executesql @sql, N'@custid nchar(5)', N'ALFKI'

I'm making use of two things here:

1. EXEC accepts a variable for the procedure name.
2. A system procedure executes in the context from the database it was invoked, also with three-part notation.

You could argue that an advantage with the USE statement is that the debug print makes it clear in which database the SQL batch is executed. That is certainly true, but you could address this by adding a comment to the SQL string.

We learnt in the section on [EXEC() AT](http://www.sommarskog.se/dynamic_sql.html#EXECAT) that we can use **sp\_executesql** to run commands on a linked server that is another SQL Server instance. If you want the server name to be dynamic, you can use the same technique:

DECLARE @servername sysname = 'YOURSERVER',

@dbname sysname = 'NorthDynamic',

@sql nvarchar(MAX),

@sp\_executesql nvarchar(200)

SELECT @sp\_executesql = quotename(@servername) + '.' +

quotename(@dbname) + '.sys.sp\_executesql'

SELECT @sql = N'SELECT \* FROM dbo.Customers WHERE CustomerID = @custid'

PRINT @sql

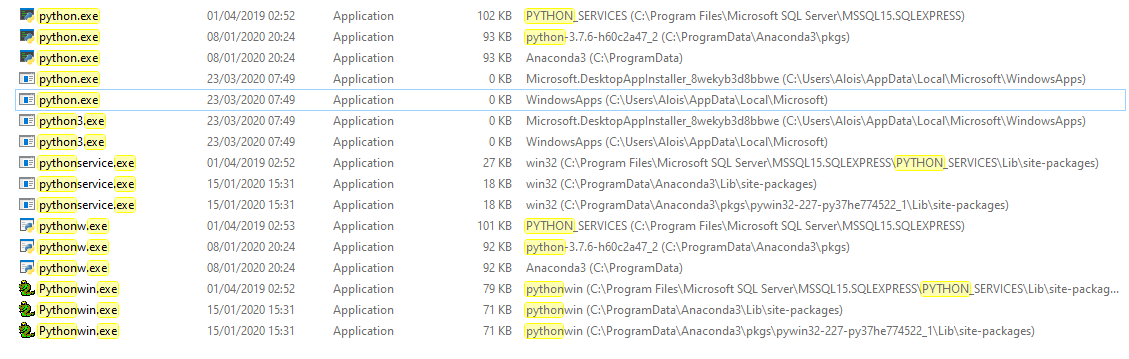
EXEC @sp\_executesql @sql, N'@custid nchar(5)', N'ALFKI'

Since USE is not easy to apply in this case, this solution comes out as really powerful here.

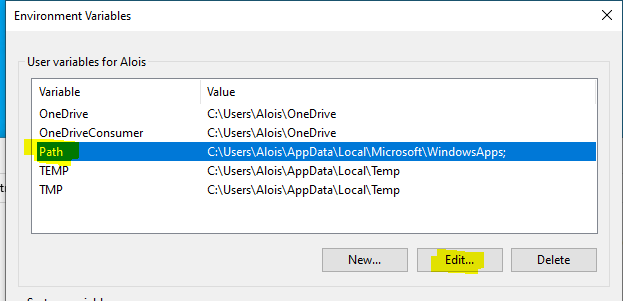
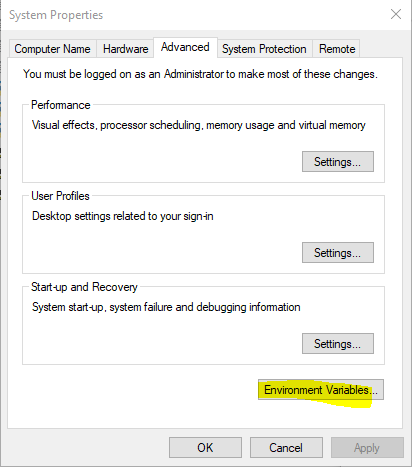
## Python installation

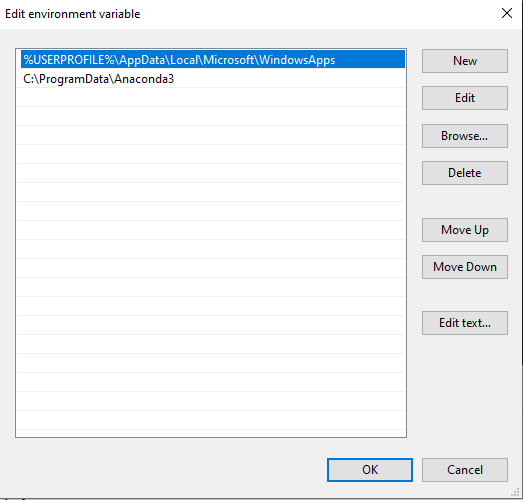
<https://www.youtube.com/watch?v=OdIHeg4jj2c>

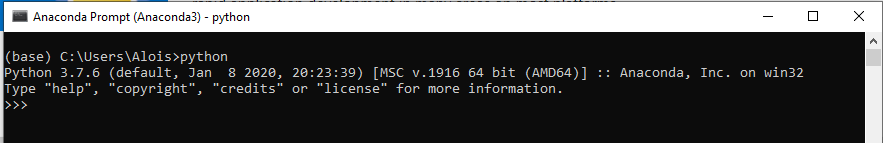
Where is the executable



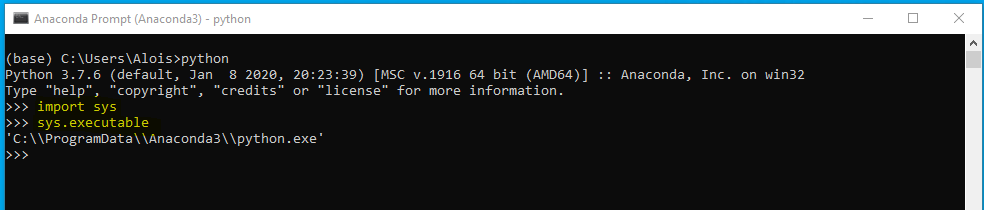
In cmd enter **view advanced system settings**

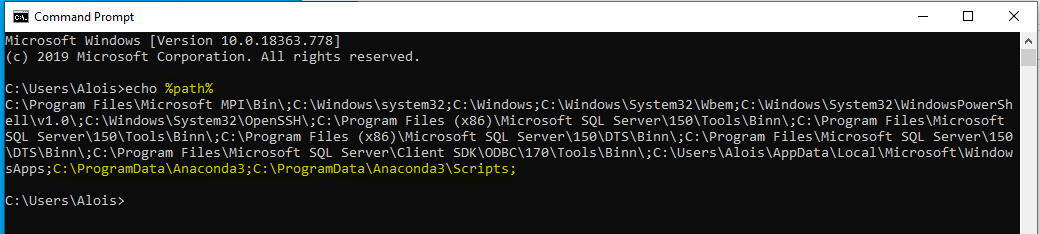




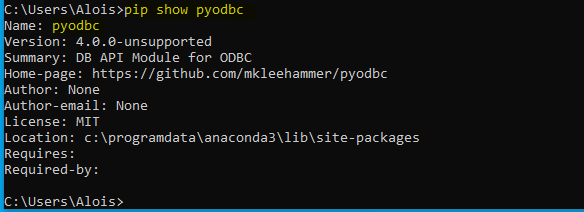
Now the Python executable is found

Find the exact location of the executable





Find installed packages in this case pyodbc



## Python virtualenv

Separate environments per project for further distribution and isolate these projects from future general updates and different versions on PCs.

**Using CONDA**

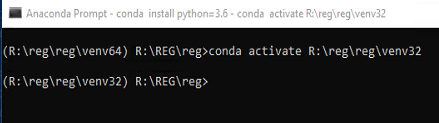
Using Conda is describted under the following link: <https://medium.com/pankajmathur/what-is-anaconda-and-why-should-i-bother-about-it-4744915bf3e6>

https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Conda allows to create a particular version:

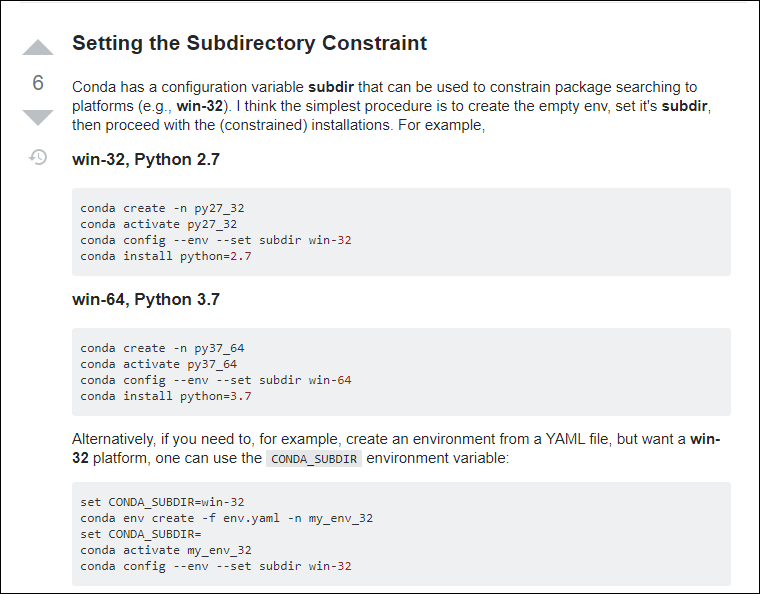
**conda create --prefix /tmp/test-env python=2.7**

1. **conda create --prefix /reg/reg/venv32**
2. **conda activate R:\reg\reg\venv32**

act

1. **conda config --env --set subdir win-32 #for 64 bit use win-64**
2. **conda install python=3.6 #choose the version you want**





An alternative way is to use pip.

**Using PIP**

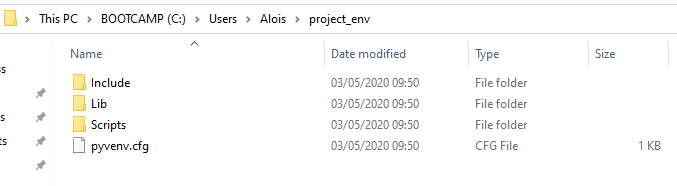
To check for installed python packages use: **pip list**

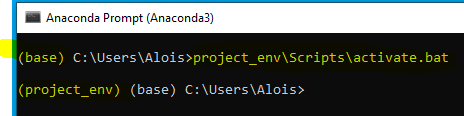
2. Create a folder in which the virtualenv will be placed: **python -m venv project\_env**

-m stands for make  
venv stands for virtual environment the package  
project\_env is the name of the new directory

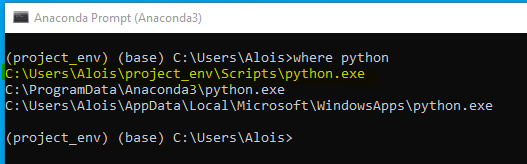
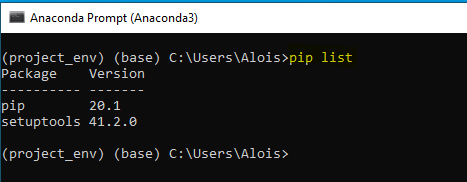


The following has now been created:

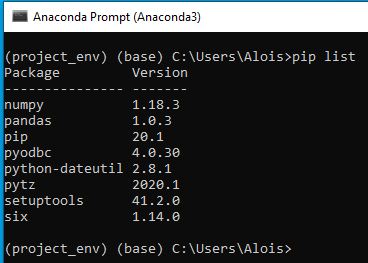


1. Activate the virtual environment: **project\_env\Scripts\activate.bat**

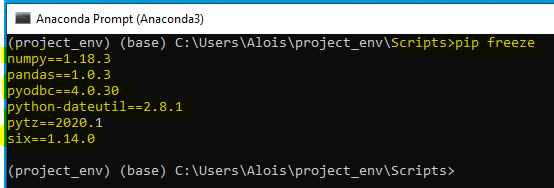
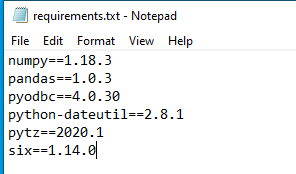
call venv/scripts/activate.bat

The (project\_env) is showing that we are now in the project environment. Typing **where python** will return as the path of the new virtual environment and pip list shows a minimal installation of packages only:  
 

We can now install additional packages in the virtual environment using pip install. After running pip install pandas and pip install pyodbc the list of packages looks now as follows:



It is now possible to share this environment with others using by using pip freeze and pasting the displayed information to a text file called requirements.txt:

Let’s now assume you would like to install the same environment somewhere else (either on your or someone else’s computer. This can be done with **pip install -r requirements.txt**

1. First again make a new directory

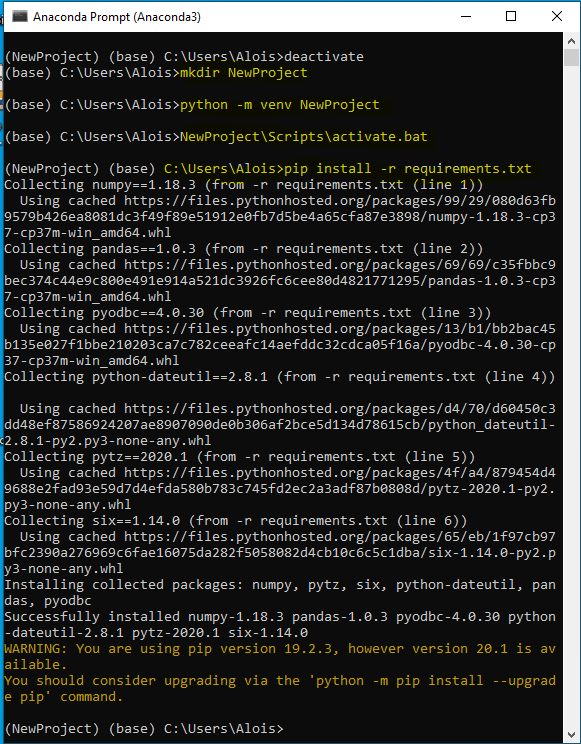
**Mkdir NewProject**

1. Then create the new virtual environment in that new directory

**Pyhton -m venv NewProject**

1. Activate the newly created environment  
   **NewProject\Scripts\activate.bat**
2. Install the packages using the requirements.txt file

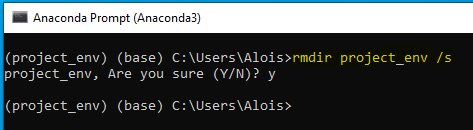
**pip install -r requirements.txt**



To deactivate simply enter **deactivate.**

To remove a virtual environment either delete the folder in the explorer or use the following cmd:

**Rmdir project\_env /s**



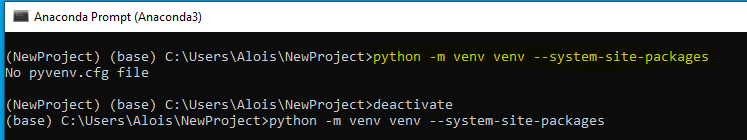
The venv folder is usually located within the project folder but will not contain any other project related files.

The venv should be something that can be destroyed and rebuilt without impacting the program files

The venv folder will not be part of git version control while the requirements.txt file should be part of version control.

It is also possible to create a virtual environment containing all of the packages of the system (anaconda). This can be done (ca. 115MB) by using the following cmd:

**Python -m venv venv –system-site-packages**



For a lean setup it would make sense to only install the actually needed packages.

**pip list –local** will only show the packages in the active environment.

## Pyinstaller

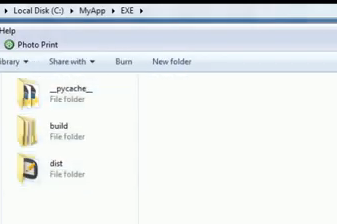
Pyinstaller.exe is located here: C:\ProgramData\Anaconda3\Scripts\pyinstaller.exe

Place the .py file and the icon in the same folder. Then enter the following cmd:

**Pyinstaller –onefile –windowed –icon=name.ico MyApp.py**

C:\ProgramData\Anaconda3\Scripts\build the new app will be published

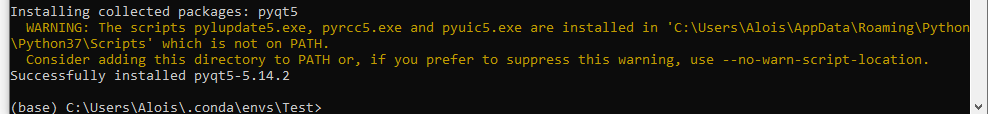
Link To Download InnoSetup Complier [http://www.jrsoftware.org/isinfo.php](https://www.youtube.com/redirect?v=a_PLU6eLLe0&event=video_description&redir_token=8QYlJSH7vMmINjey-riXPIvvTfp8MTU4ODYwOTAwN0AxNTg4NTIyNjA3&q=http%3A%2F%2Fwww.jrsoftware.org%2Fisinfo.php)



C:\ProgramData\Anaconda3\Scripts\dist there will now be a MyApp.exe

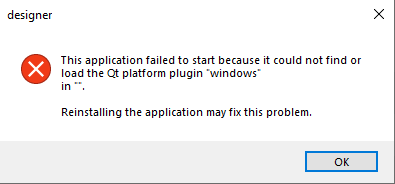
## PyQt5

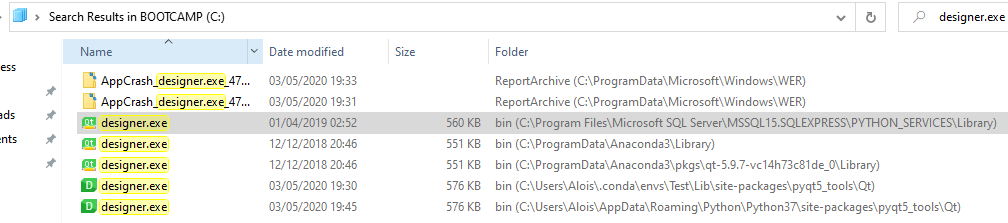
Pip install PyQt5



<https://pypi.org/project/pyqt5-tools/>

yourenv\Scripts\pip.exe install --pre pyqt5-tools~=5.11





5. open the project in VS Code and use its built-in terminal to cd to the script folder in you virtualenv.

6. type source activates (in vs code I use the git terminal).

7. deactivate to turn off the virtualenv.

## Why PyReg?

Simplicity

Concise readable, very well documented language

In house in Finance knowlege is available

Efficiency

Fast implementation

In memory calculation with pandas data frames

64 bit

Portability

Running on multiple platforms

Python code can be run on SQL Server 2017

Development can be done in Python and then portet to SQL Server

Cost

Open source, no cost

Governance and security aspects

Version control via git

We can go as far as freezing a release for example with pyinstaller in one executable file (.exe).

https://readthedocs.org/projects/pyinstaller/downloads/pdf/stable/

If needed, locate this file, which can be run on any platform on a dedicated virtual machine

where only very selective people have access to.

Pandas and vectorization:

<https://stackoverflow.com/questions/52673285/performance-of-pandas-apply-vs-np-vectorize-to-create-new-column-from-existing-c>

|  |
| --- |
| def divide(a, b):  if b == 0:  return 0.0  return float(a)/b  df['result'] = df.apply(lambda row: divide(row['A'], row['B']), axis=1)  df['result2'] = np.vectorize(divide)(df['A'], df['B'])  df.head()  # A B result result2  # 0 78 50 1.560000 1.560000  # 1 23 91 0.252747 0.252747  # 2 55 62 0.887097 0.887097  # 3 82 64 1.281250 1.281250  # 4 99 80 1.237500 1.237500 |

<https://towardsdatascience.com/data-science-with-python-turn-your-conditional-loops-to-numpy-vectors-9484ff9c622e>

|  |
| --- |
| import numpy as np from math import sin as sn import matplotlib.pyplot as plt import time**# Number of test points** N\_point = 1000**# Define a custom function with some if-else loops** def myfunc(x,y):  if (x>0.5\*y and y<0.3):  return (sn(x-y))  elif (x<0.5\*y):  return 0  elif (x>0.2\*y):  return (2\*sn(x+2\*y))  else:  return (sn(y+x))**# List of stored elements, generated from a Normal distribution** lst\_x = np.random.randn(N\_point) lst\_y = np.random.randn(N\_point) lst\_result = []**# Optional plots of the data** plt.hist(lst\_x,bins=20) plt.show() plt.hist(lst\_y,bins=20) plt.show()**# First, plain vanilla for-loop** t1=time.time() for i in range(len(lst\_x)):  x = lst\_x[i]  y= lst\_y[i]  if (x>0.5\*y and y<0.3):  lst\_result.append(sn(x-y))  elif (x<0.5\*y):  lst\_result.append(0)  elif (x>0.2\*y):  lst\_result.append(2\*sn(x+2\*y))  else:  lst\_result.append(sn(y+x)) t2=time.time()print("\nTime taken by the plain vanilla for-loop\n----------------------------------------------\n{} us".format(1000000\*(t2-t1)))**# List comprehension** print("\nTime taken by list comprehension and zip\n"+'-'\*40) %timeit lst\_result = [myfunc(x,y) for x,y in zip(lst\_x,lst\_y)]**# Map() function** print("\nTime taken by map function\n"+'-'\*40) %timeit list(map(myfunc,lst\_x,lst\_y))**# Numpy.vectorize method** print("\nTime taken by numpy.vectorize method\n"+'-'\*40) vectfunc = np.vectorize(myfunc,otypes=[np.float],cache=False) %timeit list(vectfunc(lst\_x,lst\_y))**# Results** Time taken by the plain vanilla for-loop ---------------------------------------------- **2000.0934600830078** us  Time taken by list comprehension and zip ---------------------------------------- 1000 loops, best of 3: **810 µs** per loop  Time taken by map function ---------------------------------------- 1000 loops, best of 3: **726 µs** per loop  Time taken by numpy.vectorize method ---------------------------------------- 1000 loops, best of 3: **516 µs** per loop |