Wanted to understand - As I installed just released version of python 3.14, Should uninstall and reinstall all wheels using pip uninstall followed by pip install --pre <package>? Is --pre necessary to install latest wheels of 3.14?

Great question! Let me clarify this for you:

**You don't need to reinstall all packages**, but you will need to reinstall packages that have C extensions or are compiled binaries (wheels). Pure Python packages should continue working fine.

## About the --pre flag:

The --pre flag is **not related to Python versions** - it's for installing pre-release versions of packages (alpha, beta, rc versions). You don't need it just because you're using Python 3.14.

You would only use --pre if:

* The package doesn't have a stable release compatible with Python 3.14 yet
* You explicitly want to test pre-release versions

## What you should do:

1. **Try running your code first** - many packages might already work or have compatible wheels
2. **Only reinstall packages that fail** - when you encounter import errors or compatibility issues
3. **Standard reinstall command**:

bash

pip install --force-reinstall <package>

Or simply:

bash

pip install --upgrade <package>

1. **If no stable wheel exists for Python 3.14 yet**, then you might need:

bash

pip install --pre <package>

## Quick check:

You can see which packages are installed:

bash

pip list

Since Python 3.14 is very new, some packages might not have stable wheels yet. In that case, pip will either build from source (if you have build tools installed) or you'll need --pre to get pre-release versions.

**Bottom line**: Start using Python 3.14 normally, and only reinstall packages as needed when you encounter issues. The --pre flag is only necessary if stable releases aren't available for 3.14 yet.

Claude does not have the ability to run the code it generates yet.

D

Lets do the whole process using powershell script to automate everything. First we use pip freeze and then strip of versions followed by iterative loop to switch to next package is installation of a package fails due to missing dependencies.

Give me minimal script without any comments.

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

$failed = @()

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --force-reinstall --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with dependencies..." -ForegroundColor Yellow

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --force-reinstall $pkg

}

Write-Host "`nDone!" -ForegroundColor Green

Claude does not have the ability to run the code it generates yet.

D

But whether you used --pre switch?

You're absolutely right! I didn't include --pre. Here's the corrected version:

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

$failed = @()

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --pre --force-reinstall --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with dependencies..." -ForegroundColor Yellow

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --pre --force-reinstall $pkg

}

Write-Host "`nDone!" -ForegroundColor Green

Claude does not have the ability to run the code it generates yet.

D

But when packages are finalized will pip-review update them from development to their new final versions in future?

No, pip-review (or pip list --outdated) **won't automatically detect that a pre-release should be upgraded to a stable release**.

When you install with --pre, pip records the installed version but doesn't flag it as needing an update when stable releases come out. You'd need to manually reinstall without --pre or explicitly upgrade:

powershell

pip install --upgrade <package>

## Better approach for Python 3.14:

Try installing **without** --pre first, and only use it for packages that fail:

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

$failed = @()

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --force-reinstall --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with --pre..." -ForegroundColor Yellow

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --pre --force-reinstall $pkg

}

Write-Host "`nDone!" -ForegroundColor Green

This way, you get stable versions where available, and pre-release only when necessary. When stable versions are released later, normal pip install --upgrade or pip-review will detect and upgrade them.

Claude does not have the ability to run the code it generates yet.

D

Have you included uninstall command in the changed script?

No, I didn't include the uninstall command. Here's the corrected version:

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

foreach ($pkg in $packages) {

Write-Host "Uninstalling $pkg..." -ForegroundColor Yellow

pip uninstall -y $pkg

}

$failed = @()

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with --pre..." -ForegroundColor Yellow

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --pre $pkg

}

Write-Host "`nDone!" -ForegroundColor Green

Claude does not have the ability to run the code it generates yet.

D

Script is running and uninstalling packages . But have you taken care to store what was uninstalled. How script will keep track of what was uninstalled?

The script **does keep track** - look at the first line:

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

This captures all package names from pip freeze into the $packages array **before** uninstalling anything. The script then uses this same array to reinstall all packages that were uninstalled.

So the tracking is already there:

1. Line 1: Store all package names in $packages
2. Loop 1: Uninstall everything in $packages
3. Loop 2: Reinstall everything in $packages
4. Loop 3: Retry failed ones with --pre

The $packages variable persists throughout the script execution, so it knows exactly what to reinstall.

Claude does not have the ability to run the code it generates yet.

D

Some beginning packages like accesscontrol, acquisition, etc. are failing why? These are not required for some purpose?

Those packages (AccessControl, Acquisition, etc.) are **Zope framework components** - they're legacy Python web framework packages with C extensions.

They're likely failing because:

1. **No Python 3.14 wheels available yet** (pre-release or stable)
2. **Building from source is failing** (missing C compiler or dependencies)
3. **They may be incompatible** with Python 3.14

**Are they required?** Only if you're using Zope, Plone, or applications built on them. If you're not working with these frameworks, you probably don't need them.

**What to do:**

Let the script continue. For packages that fail even with --pre, you have options:

1. **Skip them** if not needed
2. **Wait** for 3.14-compatible releases
3. **Install build tools** (Visual Studio Build Tools on Windows) to compile from source
4. **Check if those packages are actually needed** in your workflow

The script will track which ones failed. After it completes, check the failed list and decide which ones you actually need for your work.

D

But packages like pyarrow are extremely difficult or even near impossible to build from source on windows even with visual studio buildtools

You're absolutely right! Packages like pyarrow, numpy, pandas, scipy, pillow, etc. with complex C/C++ dependencies are practically impossible to build from source on Windows.

**For Python 3.14 (just released), this is a major issue because:**

* Most scientific/data packages don't have wheels yet
* Building from source requires not just VS Build Tools, but also specific libraries (Arrow C++, BLAS, etc.)

**Better approach:**

1. **Wait a few weeks/months** for maintainers to release Python 3.14 wheels
2. **Use Python 3.13** instead for production work (stable ecosystem)
3. **Check compatibility first** before mass reinstalling:

powershell

pip list --outdated

1. **For critical packages without wheels**, look for pre-built wheels at:
   * <https://www.lfd.uci.edu/~gohlke/pythonlibs/> (Unofficial Windows binaries)

**Recommendation:**

Since you're hitting these issues, **consider staying on Python 3.13** until the ecosystem catches up (typically 2-4 months after a Python release). Python 3.14 is great for testing, but for daily work with data science/ML packages, the stable version is more practical.

Should I help you revert to 3.13, or do you want to continue and just skip the problematic packages for now?

Claude does not have the ability to run the code it generates yet.

D

Just check whether your script falls back to --pre if unsuccessful because I finding failed mesage and then --pre is not tried or am I missing?

You're right to question this! Looking at my script again:

powershell

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with --pre..." -ForegroundColor Yellow

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --pre $pkg

}

The script **does retry with** --pre, but there's a problem: the retry happens **after all packages are processed in the first loop**.

So you'll see:

1. All installs attempted (with "Failed" messages)
2. **Then** a message "Retrying failed packages with --pre..."
3. Then retries with --pre

If you're seeing "Failed" messages but not seeing the "Retrying failed packages with --pre..." message yet, **the script is still in the first loop**. The retries come after.

Are you seeing the yellow "Retrying failed packages with --pre..." message at all? Or is the script still running through the initial installation loop?

Claude does not have the ability to run the code it generates yet.

D

Oh! I see So after first loop those failed will be retried?

**Yes, exactly!**

The script works in phases:

1. **First loop**: Try to install all packages without --pre (you're seeing "Failed" messages here)
2. **Second loop**: After the first loop completes, it retries all failed packages with --pre

So right now you're still in the first loop. Once it finishes going through all packages, you'll see:

Retrying failed packages with --pre...

Then it will retry each failed package with the --pre flag.

Just let it continue running - the retries are coming after the current loop completes!

Claude does not have the ability to run the code it generates yet.

D

Is there any log for failed packages? How will I know which packages had failed in future? We missed that aspect.

You're absolutely right! The script doesn't save the failed packages to a file. Here's the updated version with logging:

powershell

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

foreach ($pkg in $packages) {

Write-Host "Uninstalling $pkg..." -ForegroundColor Yellow

pip uninstall -y $pkg

}

$failed = @()

foreach ($pkg in $packages) {

Write-Host "Installing $pkg..." -ForegroundColor Cyan

pip install --no-deps $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$failed += $pkg

Write-Host "Failed: $pkg" -ForegroundColor Red

}

}

Write-Host "`nRetrying failed packages with --pre..." -ForegroundColor Yellow

$stillFailed = @()

foreach ($pkg in $failed) {

Write-Host "Retrying $pkg..." -ForegroundColor Cyan

pip install --pre $pkg 2>$null

if ($LASTEXITCODE -ne 0) {

$stillFailed += $pkg

}

}

$failed | Out-File "failed\_packages.txt"

$stillFailed | Out-File "still\_failed\_after\_pre.txt"

Write-Host "`nLogs saved to failed\_packages.txt and still\_failed\_after\_pre.txt" -ForegroundColor Green

Write-Host "Done!" -ForegroundColor Green

This will create two log files for you to review later.

Claude does not have the ability to run the code it generates yet.

D

But script is already in run phase. Cannot change now.

You're right - the script is already running.

**After it completes**, you can check which packages failed by running:

powershell

pip list

Then compare it against your original packages. Or create a quick check script:

powershell

$original = Get-Content original\_packages.txt

$current = pip freeze | ForEach-Object { ($\_ -split '==')[0] }

$missing = $original | Where-Object { $\_ -notin $current }

$missing | Out-File "missing\_packages.txt"

But you'd need your original package list saved somewhere first.

**For now**: Just let the current script finish. The $failed array exists in memory during execution, so the retry loop will work. You just won't have a persistent log after the PowerShell session ends.

**Next time**: Run the updated script with logging from the start.

Claude does not have the ability to run the code it generates yet.

D

But somewhere in python any track is maintained?

**No**, Python/pip doesn't automatically maintain a log of failed installations.

However, you might have some traces:

1. **pip cache logs** (limited info):

powershell

pip cache info

1. **Check what's currently installed vs what you had**: If you had run pip freeze > requirements.txt before starting, you could compare:

powershell

pip freeze > current\_packages.txt

1. **PowerShell history** (if you're lucky): The console output might still be in your terminal's scrollback buffer - you could scroll up to see the "Failed" messages.
2. **Windows Event Viewer** - No relevant logs there for pip

**But realistically**: Once your PowerShell session ends, the $failed array is gone with no persistent record.

**Best you can do now**:

* Let the script finish
* Before closing PowerShell, manually copy the $failed array:

powershell

$failed

$stillFailed

This will print them to console so you can copy them.

**Lesson learned**: Always add logging for long-running scripts! 😅

Claude does not have the ability to run the code it generates yet.

D

Which of these are native to python (already installed with python)

from bs4 import BeautifulSoup from cffi import FFI from concurrent.futures import ThreadPoolExecutor from ctypes import c\_char\_p, c\_int, POINTER, byref from dask.distributed import Client from datetime import datetime from datetime import datetime as dt from datetime import datetime, timedelta from decimal import Decimal from io import StringIO from javascript import require from pathlib import Path from py4j.java\_gateway import JavaGateway from randomgen import RDRAND from rdrand import RdRandom from rdrand import RdSeedom #from scipy.optimize import newton from sympy import sin, cos, log, sqrt from sympy import symbols, integrate from ta.trend import sma\_indicator from ta.trend import wma\_indicator from typing import Optional, Dict, List, Any import calendar import cffi import csv import ctypes import dask.dataframe as dd import datetime import execjs import json import math import numpy as np import os import pandas as pd import pcre2 #import pyarrow as pa #import pyarrow.compute as pc #import pyarrow.csv as csv import rdrand import re import regex import regex as re1 import requests import sqlite3 import string import subprocess import sympy as sp import sys import threading import time import warnings import xlwings as xw import yfinance as y

Here are the **native Python modules** (included with Python installation):

**Native (no pip install needed):**

* concurrent.futures (ThreadPoolExecutor)
* ctypes
* datetime
* decimal (Decimal)
* io (StringIO)
* pathlib (Path)
* typing (Optional, Dict, List, Any)
* calendar
* csv
* json
* math
* os
* re
* sqlite3
* string
* subprocess
* sys
* threading
* time
* warnings

**Requires pip install:**

* bs4 (BeautifulSoup) - beautifulsoup4
* cffi / cffi.FFI
* dask
* javascript / execjs
* py4j
* randomgen / rdrand
* scipy
* sympy
* ta
* numpy
* pandas
* pcre2
* pyarrow
* regex
* requests
* xlwings
* yfinance

So roughly **40%** of your imports are native, **60%** require external packages.

D

Your list did not include execjs

You're right! I missed it in the summary list.

**execjs** is **NOT native** - it requires pip install.

So the corrected categorization:

**Requires pip install:**

* bs4 (BeautifulSoup) - beautifulsoup4
* cffi / cffi.FFI
* dask
* execjs ✓
* javascript
* py4j
* randomgen / rdrand
* scipy
* sympy
* ta
* numpy
* pandas
* pcre2
* pyarrow
* regex
* requests
* xlwings
* yfinance

I had mentioned execjs in the original list but forgot to repeat it in the summary. Thanks for catching that!

D

Again the list. Check each and tell me again:

from bs4 import BeautifulSoup from cffi import FFI from concurrent.futures import ThreadPoolExecutor from ctypes import c\_char\_p, c\_int, POINTER, byref from dask.distributed import Client from datetime import datetime from datetime import datetime as dt from datetime import datetime, timedelta from decimal import Decimal from io import StringIO from javascript import require from pathlib import Path from py4j.java\_gateway import JavaGateway from randomgen import RDRAND from rdrand import RdRandom from rdrand import RdSeedom #from scipy.optimize import newton from sympy import sin, cos, log, sqrt from sympy import symbols, integrate from ta.trend import sma\_indicator from ta.trend import wma\_indicator from typing import Optional, Dict, List, Any import calendar import cffi import csv import ctypes import dask.dataframe as dd import datetime import execjs import json import math import numpy as np import os import pandas as pd import pcre2 #import pyarrow as pa #import pyarrow.compute as pc #import pyarrow.csv as csv import rdrand import re import regex import regex as re1 import requests import sqlite3 import string import subprocess import sympy as sp import sys import threading import time import warnings import xlwings as xw import yfinance as yf

Let me go through each import carefully:

**NATIVE (included with Python):**

* concurrent.futures (ThreadPoolExecutor)
* ctypes (c\_char\_p, c\_int, POINTER, byref)
* datetime (datetime, timedelta)
* decimal (Decimal)
* io (StringIO)
* pathlib (Path)
* typing (Optional, Dict, List, Any)
* calendar
* csv
* json
* math
* os
* re
* sqlite3
* string
* subprocess
* sys
* threading
* time
* warnings

**REQUIRE PIP INSTALL:**

* bs4 (BeautifulSoup4)
* cffi
* dask (dask.distributed, dask.dataframe)
* execjs
* javascript
* numpy
* pandas
* pcre2
* pyarrow
* py4j
* randomgen
* rdrand
* regex (different from standard re)
* requests
* scipy
* sympy
* ta (technical analysis library)
* xlwings
* yfinance

**Summary:** 21 native, 18 require pip install.

D

PS D:\Programs\Powershell> pip install execjs --upgrade WARNING: Ignoring invalid distribution ~edi (D:\Programs\Python\Lib\site-packages) ERROR: Could not find a version that satisfies the requirement execjs (from versions: none) ERROR: No matching distribution found for execjs

The package name is PyExecJS, not execjs!

Try:

powershell

pip install PyExecJS --upgrade

or with --pre for Python 3.14:

powershell

pip install --pre PyExecJS

```

The import statement is `import execjs`, but the pip package name is `PyExecJS`.

---

\*\*Also note the warning:\*\*

```

WARNING: Ignoring invalid distribution ~edi

This suggests a corrupted package installation (possibly redis, pexpect, or similar). You might want to check:

powershell

pip list | Select-String "~"

And uninstall any corrupted packages shown.

Claude does not have the ability to run the code it generates yet.

### consolidated\_udfs.py

5,002 lines

py

D

See the attached file. On top, I have marked unavailable 3.14 packages with comments in imports. In below UDFs, you comment out all such functions and give me update py file.

I'll analyze the imports and comment out all functions that depend on unavailable Python 3.14 packages.

consolidated\_udfs\_py314.py

Code

""" Consolidated xlwings UDF Module - Python 3.14 Compatible Auto-generated by merge script Modified to comment out functions requiring unavailable packages """ from bs4 import BeautifulSoup from cffi import FFI from concurrent.futures import Thread

D

Continue

consolidated\_udfs\_py314.py

Code ∙ Version 2

output\_list = [ffi.string(output\_array\_c[0][i]).decode('utf-8') for i in range(match\_count)] final\_output = [[item] if item in output\_list else [""] for item in input\_list] dll.free\_matches(output\_array\_c[0], match\_count)

D

Give me similar script like following powershell script for bash. I wan to run in python virtual environment in msys64. I am accessing virtual env created vis python 3.14 in msys64:

$packages = pip freeze | ForEach-Object { ($\_ -split '==')[0] } foreach ($pkg in $packages) { Write-Host "Uninstalling $pkg..." -ForegroundColor Yellow pip uninstall -y $pkg } $failed = @() foreach ($pkg in $packages) { Write-Host "Installing $pkg..." -ForegroundColor Cyan pip install --no-deps $pkg 2>$null if ($LASTEXITCODE -ne 0) { $failed += $pkg Write-Host "Failed: $pkg" -ForegroundColor Red } } Write-Host "nRetrying failed packages with --pre..." -ForegroundColor Yellow $stillFailed = @() foreach ($pkg in $failed) { Write-Host "Retrying $pkg..." -ForegroundColor Cyan pip install --pre $pkg 2>$null if ($LASTEXITCODE -ne 0) { $stillFailed += $pkg } } $failed | Out-File "failed\_packages.txt" $stillFailed | Out-File "still\_failed\_after\_pre.txt" Write-Host "nLogs saved to failed\_packages.txt and still\_failed\_after\_pre.txt" -ForegroundColor Green Write-Host "Done!" -ForegroundColor Green

Here's the equivalent bash script for MSYS2 environment:

bash

#!/bin/bash

*# ANSI color codes*

YELLOW='\033[1;33m'

CYAN='\033[1;36m'

RED='\033[1;31m'

GREEN='\033[1;32m'

NC='\033[0m' *# No Color*

*# Get list of installed packages (without versions)*

packages=($(pip freeze | cut -d'=' -f1))

*# Uninstall all packages*

for pkg in "${packages[@]}"; do

echo -e "${YELLOW}Uninstalling $pkg...${NC}"

pip uninstall -y "$pkg"

done

*# Arrays to track failed installations*

failed=()

*# Try to install packages without dependencies*

for pkg in "${packages[@]}"; do

echo -e "${CYAN}Installing $pkg...${NC}"

if ! pip install --no-deps "$pkg" 2>/dev/null; then

failed+=("$pkg")

echo -e "${RED}Failed: $pkg${NC}"

fi

done

*# Retry failed packages with --pre flag*

echo -e "\n${YELLOW}Retrying failed packages with --pre...${NC}"

still\_failed=()

for pkg in "${failed[@]}"; do

echo -e "${CYAN}Retrying $pkg...${NC}"

if ! pip install --pre "$pkg" 2>/dev/null; then

still\_failed+=("$pkg")

fi

done

*# Save logs to files*

printf "%s\n" "${failed[@]}" > failed\_packages.txt

printf "%s\n" "${still\_failed[@]}" > still\_failed\_after\_pre.txt

echo -e "\n${GREEN}Logs saved to failed\_packages.txt and still\_failed\_after\_pre.txt${NC}"

echo -e "${GREEN}Done!${NC}"

*# Print summary*

echo -e "\n${CYAN}Summary:${NC}"

echo "Total packages: ${#packages[@]}"

echo "Failed initially: ${#failed[@]}"

echo "Still failed after --pre: ${#still\_failed[@]}"

**To use this script in MSYS2:**

1. **Save the script** (e.g., as reinstall\_packages.sh):

bash

nano reinstall\_packages.sh

1. **Make it executable**:

bash

chmod +x reinstall\_packages.sh

1. **Activate your Python 3.14 virtual environment first**:

bash

source /path/to/your/venv/bin/activate

1. **Run the script**:

bash

./reinstall\_packages.sh

**Alternative: One-liner version** (if you prefer not to create a file):

bash

packages=($(pip freeze | cut -d'=' -f1)); for pkg in "${packages[@]}"; do echo "Uninstalling $pkg..."; pip uninstall -y "$pkg"; done; failed=(); for pkg in "${packages[@]}"; do echo "Installing $pkg..."; pip install --no-deps "$pkg" 2>/dev/null || failed+=("$pkg"); done; echo "Retrying failed packages with --pre..."; still\_failed=(); for pkg in "${failed[@]}"; do echo "Retrying $pkg..."; pip install --pre "$pkg" 2>/dev/null || still\_failed+=("$pkg"); done; printf "%s\n" "${failed[@]}" > failed\_packages.txt; printf "%s\n" "${still\_failed[@]}" > still\_failed\_after\_pre.txt; echo "Done! Logs saved."

**Key differences from PowerShell:**

* Uses cut -d'=' -f1 instead of PowerShell's -split '=='
* Uses bash arrays () instead of PowerShell arrays @()
* Uses $? or ! for exit code checking instead of $LASTEXITCODE
* Uses printf "%s\n" to write arrays to files instead of Out-File
* ANSI color codes for colored output instead of -ForegroundColor

The script will create the same two log files showing which packages failed during installation.