# Exercise 1:

**Task 1: Introduction: Understanding HPDA Use Cases (60 min)**

1. Research one recent HPDA use case from industry in the literature, e.g., using Google or Google Scholar. Describe the use case briefly, put an emphasis on performance relevant characteristics, e.g., how much data do they process, how long does the workflow run? Address how the Big Data Challenges (5V) apply.
Make sure you cite the used literature properly! Your notes should cover about 1/4-page.

**Answer**:
A notable use case in High Performance Data Analytics (HPDA) is genome sequencing, which has become increasingly essential for both medical research and healthcare innovation. This process leverages HPDA to analyse massive datasets produced by sequencers-each generating up to 6 terabytes of data daily. Using HPDA allows researchers to significantly reduce the time required for analysis, with workflows that used to take years now producing results within a day. This increase in efficiency is vital for timely insights, such as identifying genetic markers or developing targeted therapies for diseases.

In this use case, several of the 5V Big Data challenges are present.

- Volume: The vast data produced daily (in petabytes) must be processed, requiring scalable storage and efficient data access systems.
- Velocity: Real-time data throughput is crucial, especially when processing high bandwidth data, which demands optimized storage and retrieval speeds.
- Variety: The data spans structured genomic sequences and unstructured metadata, making unified management systems essential.
- Veracity: Reliable, accurate data handling is critical for clinical relevance, demanding low error rates and data integrity.
- Value: Timely genomic delivers immense values, transforming healthcare through personalized treatments and rapid diagnostics.

Rerence: Wang H, Wang X, Ren H, Wang X, Lu Z.2020.3-Hydroxypyridine Dehydrogenase HpdA Is Encoded by a Novel Four-Component Gene Cluster and Catalyzes the First Step of 3-Hydroxypyridine Catabolism in Ensifer adhaerens HP1. Appl Environ Microbiol86:e01313-20.https://doi.org/10.1128/AEM.01313-20

2. Research a scientific HPDA use case following the same instructions as above.

**Answer:**
One prominent scientific use case for High Performance Data Analytics (HPDA) is in earth system modeling, where HPDA techniques are used to simulate complex environmental and climate processes. This application involves running extensive simulations that demand high-resolution data from multiple sources, such as satellite imagery and environmental sensors. HPDA tools enable researchers to analyze petabytes of data to model weather patterns, predict natural disasters, and study climate change with improved accuracy. By integrating deep learning models as

“surrogates,” the time required for these simulations has been reduced significantly, allowing researchers to explore various scenarios rapidly and efficiently.

This application directly addresses the Big Data Challenges (5V):

- Volume: The system processes petabytes of data generated from various global sources, which must be integrated and analyzed continuously.
- Velocity: Real-time data processing is crucial to predict rapid changes in weather and environmental conditions.
- Variety: Data sources include structured meteorological data, unstructured satellite imagery, and heterogeneous environmental datasets.
- Veracity: Data accuracy is essential to ensure reliable climate predictions, as errors can lead to significant predictive inaccuracies.
- Value: Timely and accurate climate models provide substantial value in environmental protection, disaster preparedness, and policymaking.

Reference: Lewandowski, Natalie and Koller, Bastian. ‘Transforming Medical Sciences with High-performance Computing, High-performance Data Analytics and AI’. 1 Jan. 2023 : 1505 – 1507.

**Task 2: Bash Exercises (60 min)**

1. $ echo "Hello World" → Prints out hello world. Everything within semicolon is treated as one string.

2. $ echo Hello, World → Also prints hello world. However, echo treats “Hello,” and “World” as two different strings. By default, the string for echo is separated by space.

3. $ echo Hello, world; Foo bar → The bash prints “Hello,” and “world”. However, it could not identify “foo”.
4. $ echo Hello, world! → Prints Hello, world!
5. $ echo "line one";echo "line two" → Prints following. “;” separates the line.
line one
line two

6. $ echo "Hello, world > readme" → prints out “Hello, world > readme “.

7. $ echo "Hello, world" > readme → creates a new file “readme” and prints “Hello, world” in it.

8. $ cat readme → read the contents of the folder “readme”.

9. $ example="Hello, World" → sets the string “Hello, World” in the variable “example”.

10. $ echo $example → prints out the content of the variable “example”.

11. $ echo ’$example’ → prints out “$example” as it treats as a string.

12. $ echo "$example" → prints out the contents of the string. So, single colon (‘’) is treated at string whereas double colon (“”) is treated as variable.

13. $ echo "Please enter your name."; read example →Suggest entering the user’s name. Then the entered user’s name is read and stored in the variable “example”.

14. $ echo "Hello $example" → Prints out “Hello Shyam”. i.e. hello + value of “example” variable.

15. $ three=1+1+1;echo $three → prints out “1+1+1”.

16. $ bc → basic calculator.

17. $ echo 1+1+1 | bc → output string “1+1+1” is sent through pipe (|) to the basic calculator. Thus, prints out 3.

18. $ let three=1+1+1;echo $three → prints 3. i.e. the value of the variable three.

19. $ echo date → In Macobook, only “date” prints the date.

20. $ cal → prints calendar.

21. $ which cal → gives the location of the calculator.

22. $ /bin/cal → prints the calendar.

23. $ $(which cal) → prints calendar

24. $ ‘which cal‘ → invalid command.

25. $ echo "The date is $(date)" → prints out “The date is Tue Nov 19 19:51:41 CET 2024”.

26. $ seq 0 9 → prints the sequence 0 to 9 vertically.

27. $ seq 0 9 | wc -l → prints out 10. The sequence is 10 and are passed through pipe for counting. –l means lines and wc means word count. Thus, 10 lines are counted.

28. $ seq 0 9 > sequence → creates the sequences of 0 to 9 and stores it in the variable “sequence”.

29. $ wc -l < sequence → the value of “sequence” variable is fed to “wc -l” which does the word linewise.

30. $ for I in $(seq 1 9) ; do echo $I ; done → prints the sequence form 1 to 9 using for-loop.

31. $ (echo -n 0 ; for I in $(seq 1 9) ; do echo -n +$I ; done ; echo) | bc → Starts with the output string , making sure that no new line is appended. Variables i takes the value from 1 to 9. For each value of I, appends (+) followed by the current number of to the output string without new line. Outputs a new line at the end of the string. Pipes the constructed string to the basic calculator, which evaluates the arithmetic expression.

**Task 2: Bash Research (60 min)**

This is a more difficult optional task which can be done instead of Task 2

Core Concepts of Bash: A Brief Overview
Bash (Bourne Again Shell) is a powerful command-line interpreter that enables users to interact with the operating system. It offers robust features, such as scripting capabilities, process control, and input/output management. Below are some core concepts of Bash with practical examples and tips to avoid common errors.

**1. Special Characters and Escaping**
Special characters like ;, &, |, and $ have specific meanings in Bash. To use them literally, they must be escaped using a backslash (\) or enclosed in quotes.

- echo "This is \$HOME"   # Displays "This is $HOME" without expanding the variable.
- echo "File path: /path/to/file; another/path"  # Works with the semicolon as part of the string.

Common Error: Forgetting to escape special characters results in unexpected behavior.
Avoidance: Always use quotes or backslashes where needed.

**2. Redirection**
Redirection manages where command output or input goes. Use > for writing to a file, >> for appending, and < to read from a file.

```
ls > output.txt        # Redirects standard output to a file.
echo "New Line" >> output.txt  # Appends to the file.
wc -l < output.txt      # Counts lines in the file using input redirection.
```

Common Error: Overwriting important files unintentionally with >.
Avoidance: Use >> when appending or back up important files.

**3. Piping**
Piping (|) passes the output of one command as input to another.

```
ls -l | grep "txt"      # Lists files and filters only those containing "txt".
cat file.txt | wc -w    # Counts words in a file.
```

Common Error: Piping with commands that don't handle input correctly.
Avoidance: Verify compatibility of commands in a pipeline.

**4. Variables**
Variables store data, which can be reused in commands.

```
MY_VAR="Hello, Bash!"
echo $MY_VAR          # Outputs: Hello, Bash!
```

Common Error: Forgetting the $ to reference variables.
Avoidance: Always prefix variable names with $ when referencing.

**5. Arithmetic**
Bash supports integer arithmetic using the $((expression)) syntax.

```
NUM1=10
NUM2=5
RESULT=$((NUM1 + NUM2))
echo $RESULT          # Outputs: 15
```

Common Error: Using floating-point numbers directly (unsupported in native Bash arithmetic).
Avoidance: Use external tools like bc for floating-point calculations.

**6. Common Commands**
Some essential commands in Bash include:

```
mkdir new_dir          # Creates a directory.
cd new_dir            # Changes to the new directory.
ls -al               # Lists all files in long format.

top                  # Displays running processes.
df -h                # Shows disk space usage.

cat file.txt          # Displays file content.
head -n 5 file.txt      # Shows the first 5 lines of the file.
```

**Avoiding Errors**

1. **Quoting Variables**: Use quotes to handle spaces in variable values.

```
'FILE="My File.txt"
cat "$FILE"          # Avoids "No such file" error due to spaces.
```

2. **Checking Commands**: Test commands like rm on sample files before running them on important data.

By mastering these core concepts and using careful syntax, you can effectively utilize Bash for everyday tasks and scripting.

**Task 3: Setup of the software environment in a Virtual Machine (60 min)**

1. I created the virutal machine in the Openstack of GWDG and and connected to it from my local machine through ssh connection. The private key was generated on the openstack and it was saved in the .ssh folder.
2. The image of Ubuntu 22.04.4 verison was created.
3. The instance for the Ubuntu was created. Also, I allocated the volume, CPUs, and, RAM for that instance.
4. Git, Openssh-server, Openjdk-11-jdk were installed.
5. New SSH key was connected and was used to connect to the virtual machine using the ssh. The set up was tested using  $ssh localhost, fingerprint was confirmed, and I was logged out from the virtual machine using ‘exit’.