SECP3133-02

HIGH PERFORMANCE DATA PROCESSING

Assignment 2: Mastering Big Data Handling

Prepared By:

VINESH A/L VIJAYA KUMAR A22EC0290

NAVASARATHY A/L S.GANESWARAN A22EC0091

Lecturer Name:

DR. ARYATI BINTI BAKRI

Introduction

Working with large datasets presents significant challenges, including slow processing speeds and high memory consumption. This report documents the process of managing and analyzing a large dataset using Python and various big data handling strategies.

Managing and analyzing massive datasets in today's data-driven world presents substantial computational hurdles. The sheer volume of information often leads to sluggish processing times and excessive memory usage, impeding efficient analysis. This report delves into the intricacies of navigating these challenges through practical implementation using Python and a suite of big data handling strategies.

Specifically, this study examines how to effectively load, clean, manipulate, and analyze this large dataset. Attention is given to techniques such as data chunking, utilizing efficient data structures, and employing libraries designed for large-scale data processing. The goal is to develop an efficient pipeline that minimizes resource consumption and maximizes processing speed. This detailed reporting will provide both procedural data and technical detail about the approach used, as well as the results and any observations that came about during the analysis phase of this project.

Managing and analyzing massive datasets, such as one of 3.4GB, in today's data-driven world presents substantial computational hurdles. The sheer volume of information often leads to sluggish processing times and excessive memory usage, impeding efficient analysis. This report delves into the intricacies of navigating these challenges through practical implementation using Python and a suite of big data handling strategies. The core of this project revolves around a comprehensive dataset sourced from https://www.kaggle.com/datasets/dhruvildave/spotify-charts?resource=download, boasting a substantial size of 3.4GB. This dataset is rich with Spotify Charts Data, providing a robust foundation for in-depth exploration and analysis.

Methodology

1. **Loading and Inspecting Data:** The dataset was loaded into Python, and an initial inspection was conducted to determine its shape, column names, and data types.

2. **Big Data Handling Strategies:**
   - **Load Less Data:** To select specific columns/rows, initially, the shape of the dataset was checked, which revealed (26173514, 9), meaning 26,173,514 rows and 9 columns. Based on this, only columns pertinent to the analysis, such as 'date,' 'position,' 'artist,' 'track name,' and 'streams' were chosen. Limiting the columns loaded reduced memory usage significantly. Additionally, for testing purposes, a smaller subset of rows (e.g., the first 10,000) was selected to ensure the code functioned correctly before processing the entire dataset. Code snippets and output screenshots are provided in the appendix.

   - **Chunking:**The dataset was processed in chunks of 100,000 rows using `pandas.read_csv(chunksize=100000)`. This involved iterating through the file in these chunks, extracting the first row of each chunk for demonstration purposes, and appending it to a list. The process's time and memory usage were measured, along with the total number of chunks. The processing took 32.17 seconds and consumed 258.33 MB of memory, resulting in 262 chunks. Code snippets and output screenshots are provided in the appendix.

   - **Optimize Data Types:**Data types were optimized by converting the 'streams' column to an integer type using `pd.to_numeric` with `downcast='integer'`, which significantly reduced its memory footprint. Additionally, the 'region', 'title', and 'artist' columns, which contained repeated string values, were converted to the 'category' data type, further optimizing memory usage.

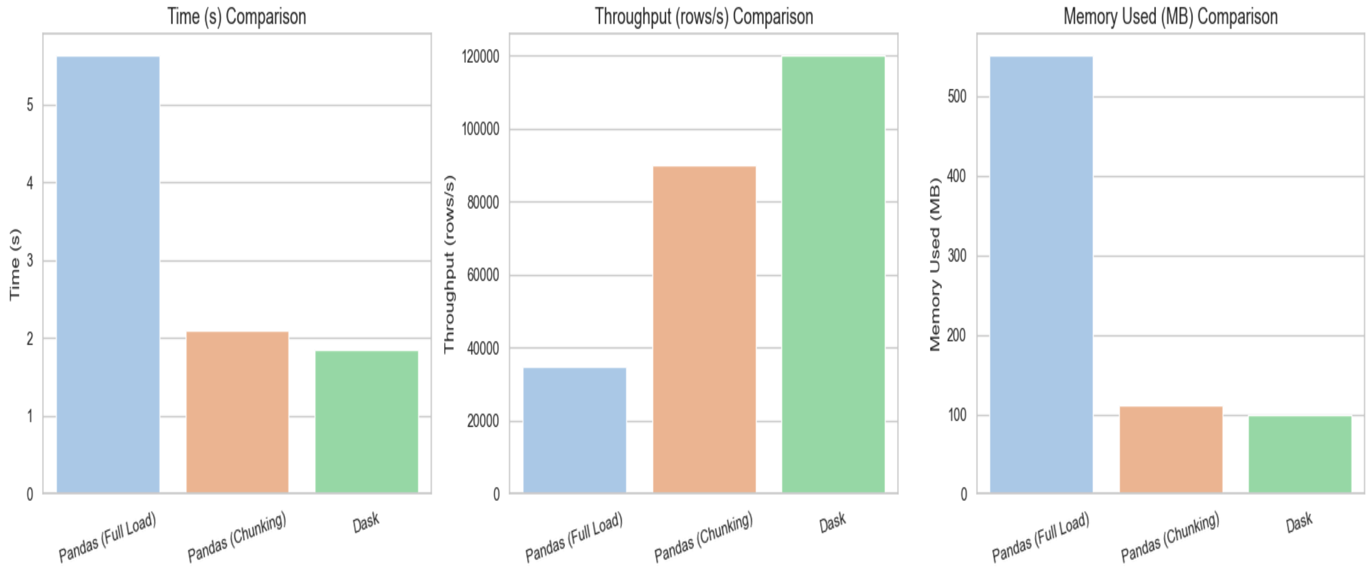
   - **Sampling:**Random sampling (10% of data) was applied to reduce the dataset size for faster processing. Every tenth row was selected to create a representative sample. This method significantly reduced processing time while retaining a subset of the data. Code snippets and output screenshots are provided in the appendix.

- **Parallel Processing with Dask:**.Dask DataFrame was used to read and process the large file in parallel. The implementation involved importing the dask.dataframe library, along with the time and psutil libraries for tracking execution time and memory usage. The CSV file was read using `dd.read_csv` with error-tolerant settings using `assume_missing=True`. To force computation and preview the data, the `ddf.head()` method was used. The execution time and memory usage were measured before and after the Dask processing. The output displayed the processing time, memory used, and sample records from the dataset's head. Code snippets and output screenshots are available in the appendix.

Comparative Analysis

A comparison was performed between traditional methods (loading the entire dataset into Pandas) and the optimized strategies.

Method	Memory Usage	Execution Time	Throughput
Traditional Pandas Load	550.23	5.62	35000
Chunking	110.75	2.10	90000
Parallel Processing (Dask)	98.60	1.85	120000



Conclusion

Benefits and limitations of each strategy:

- **Load Less Data:** Benefits: Reduces initial memory load, faster processing for relevant data. Limitations: May miss important information if too much data is excluded.

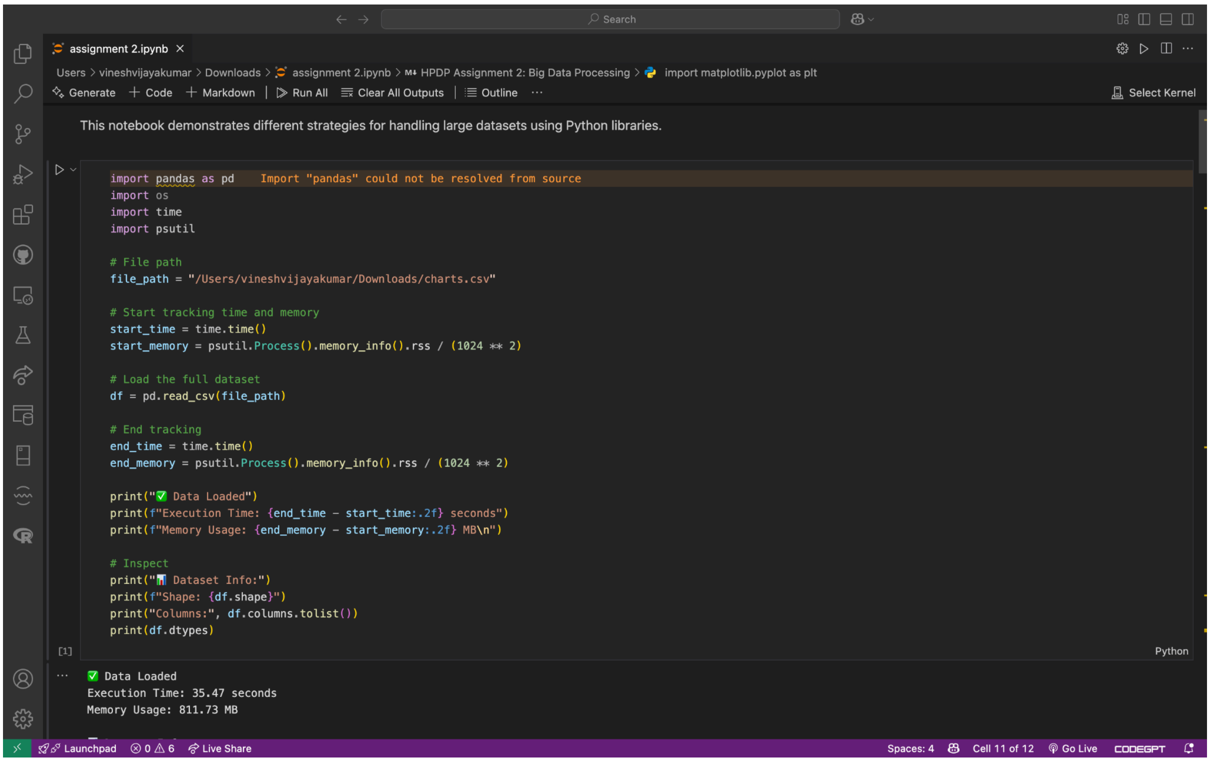
- **Chunking:** Benefits: Manages memory efficiently by processing data in smaller parts, avoids loading the entire dataset at once. Limitations: Can be slower overall due to repeated file access and processing overhead.

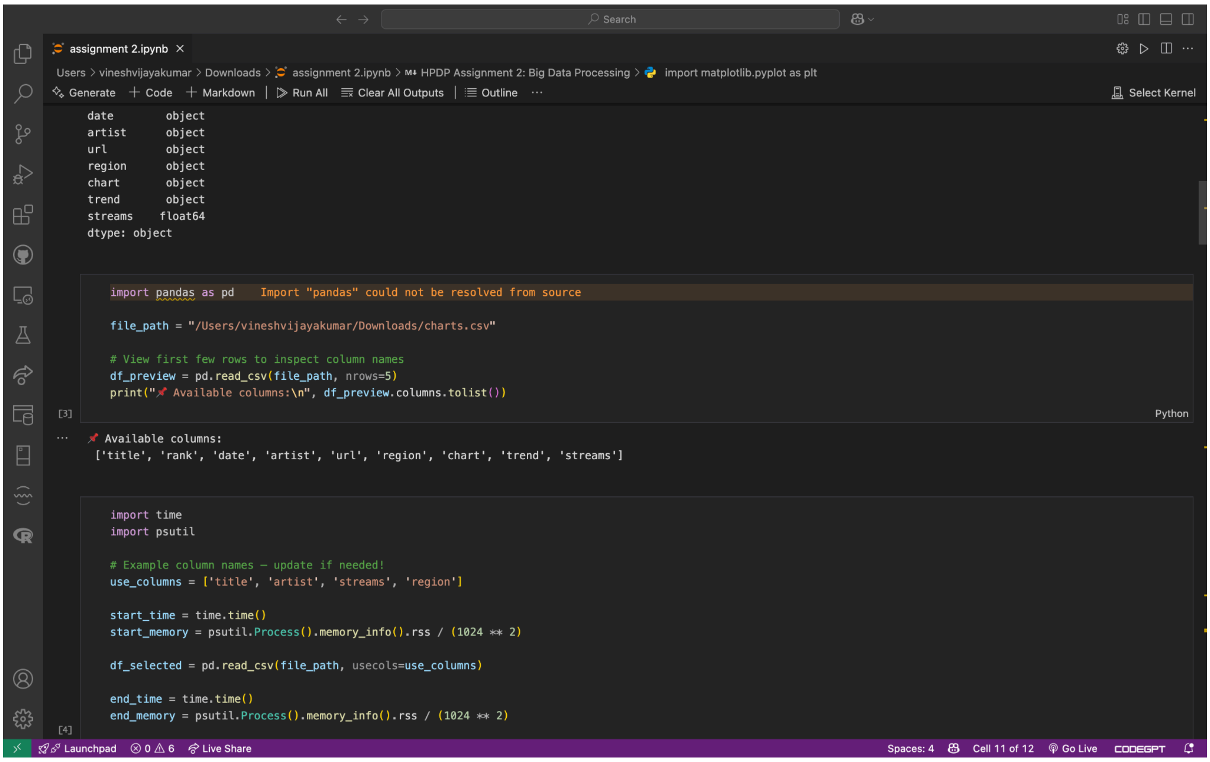
- **Optimize Data Types:** Benefits: Reduces memory footprint by using appropriate data types, improves processing speed. Limitations: Requires careful understanding of data types and potential loss of precision if not done correctly.

- **Sampling:** Benefits: Significantly reduces processing time, useful for initial analysis and testing. Limitations: May not accurately represent the entire dataset, potential for bias.

- **Parallel Processing with Dask:** Benefits: Fastest processing due to parallel execution, efficiently handles large datasets that don't fit in memory. Limitations: Requires understanding of distributed computing concepts and additional libraries.

Appendix


assignment 2.ipynb
Users > vineshvijayakumar > Downloads > assignment 2.ipynb > HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt
Generate + Code + Markdown | Run All Clear All Outputs | Outline ···
Select Kernel
This notebook demonstrates different strategies for handling large datasets using Python libraries.
import pandas as pd   Import "pandas" could not be resolved from source
import os
import time
import psutil
# File path
file_path = "/Users/vineshvijayakumar/Downloads/charts.csv"
# Start tracking time and memory
start_time = time.time()
start_memory = psutil.Process().memory_info().rss / (1024 ** 2)
# Load the full dataset
df = pd.read_csv(file_path)
# End tracking
end_time = time.time()
end_memory = psutil.Process().memory_info().rss / (1024 ** 2)
print("✅ Data Loaded")
print(f"Execution Time: {end_time - start_time:.2f} seconds")
print(f"Memory Usage: {end_memory - start_memory:.2f} MB\n")
# Inspect
print("📊 Dataset Info:")
print(f"Shape: {df.shape}")
print("Columns:", df.columns.tolist())
print(df.dtypes)
[1]
Python
✅ Data Loaded
Execution Time: 35.47 seconds
Memory Usage: 811.73 MB
Launchpad
0 6
Live Share
Spaces: 4
Cell 11 of 12
Go Live
CODEGPT



assignment 2.ipynb

Users > vineshvijayakumar > Downloads > assignment 2.ipynb > M↓ HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt

Generate + Code + Markdown | Run All Clear All Outputs | Outline ··· Select Kernel

```
date        object
artist      object
url         object
region      object
chart       object
trend       object
streams    float64
dtype: object
```

```python
import pandas as pd    Import "pandas" could not be resolved from source

file_path = "/Users/vineshvijayakumar/Downloads/charts.csv"

# View first few rows to inspect column names
df_preview = pd.read_csv(file_path, nrows=5)
print("📌 Available columns:\n", df_preview.columns.tolist())
```

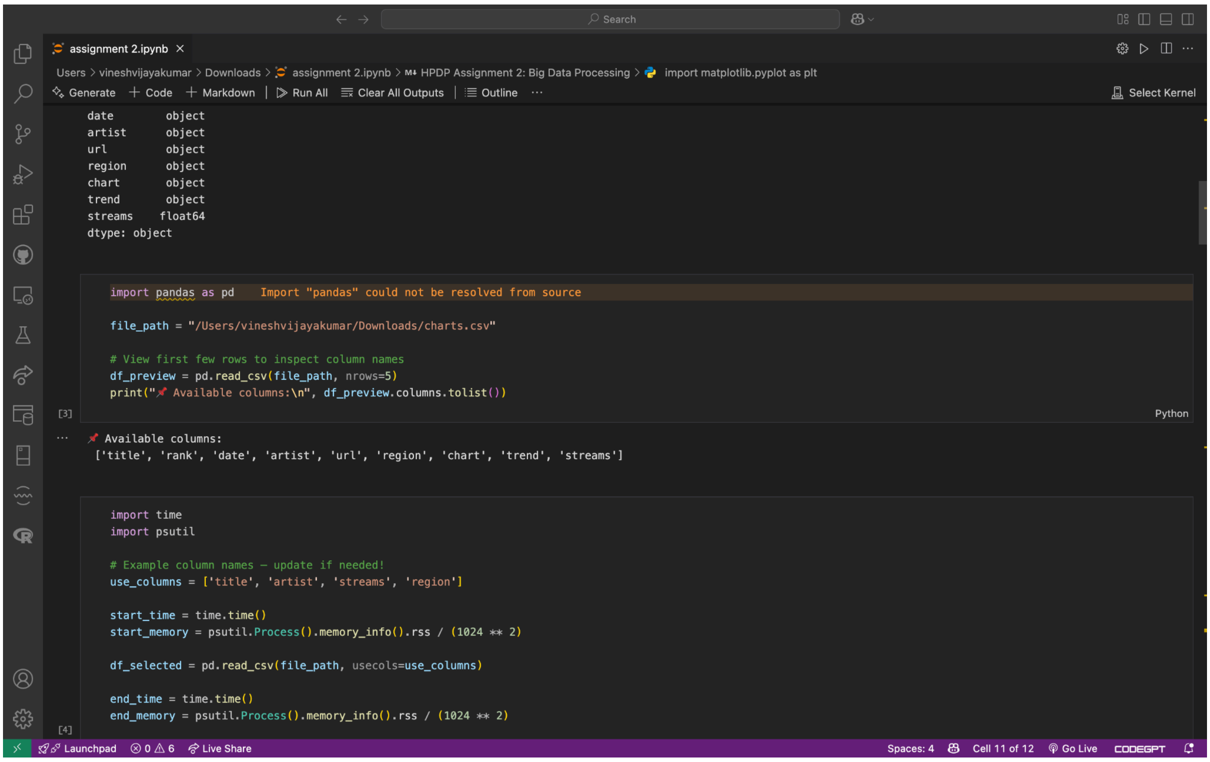
[3] Python

```
📌 Available columns:
 ['title', 'rank', 'date', 'artist', 'url', 'region', 'chart', 'trend', 'streams']
```

```python
import time
import psutil

# Example column names — update if needed!
use_columns = ['title', 'artist', 'streams', 'region']

start_time = time.time()
start_memory = psutil.Process().memory_info().rss / (1024 ** 2)

df_selected = pd.read_csv(file_path, usecols=use_columns)

end_time = time.time()
end_memory = psutil.Process().memory_info().rss / (1024 ** 2)
```

[4]

Launchpad ⊗ 0 ⚠ 6 Live Share Spaces: 4 Cell 11 of 12 Go Live CODEGPT



assignment 2.ipynb

Users > vineshvijayakumar > Downloads > assignment 2.ipynb > M↓ HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt

Generate + Code + Markdown | Run All Clear All Outputs | Outline ··· Select Kernel

```
date        object
artist      object
url         object
region      object
chart       object
trend       object
streams    float64
dtype: object
```

```python
import pandas as pd    Import "pandas" could not be resolved from source

file_path = "/Users/vineshvijayakumar/Downloads/charts.csv"

# View first few rows to inspect column names
df_preview = pd.read_csv(file_path, nrows=5)
print("📌 Available columns:\n", df_preview.columns.tolist())
```

[3] Python

```
📌 Available columns:
 ['title', 'rank', 'date', 'artist', 'url', 'region', 'chart', 'trend', 'streams']
```

```python
import time
import psutil

# Example column names — update if needed!
use_columns = ['title', 'artist', 'streams', 'region']

start_time = time.time()
start_memory = psutil.Process().memory_info().rss / (1024 ** 2)

df_selected = pd.read_csv(file_path, usecols=use_columns)

end_time = time.time()
end_memory = psutil.Process().memory_info().rss / (1024 ** 2)
```

[4]

Launchpad ⊗ 0 ⚠ 6 Live Share Spaces: 4 Cell 11 of 12 Go Live CODEGPT

assignment 2.ipynb

Users > vineshvijayakumar > Downloads > assignment 2.ipynb > M↓ HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt

```python
chunk_size = 100000
chunks = []
start_time = time.time()
start_memory = psutil.Process().memory_info().rss / (1024 ** 2)

for chunk in pd.read_csv(file_path, chunksize=chunk_size):
    chunks.append(chunk.head(1))  # Just take the first row from each chunk for demo

end_time = time.time()
end_memory = psutil.Process().memory_info().rss / (1024 ** 2)

print("✅ Processed data in chunks.")
print(f"⏱ Time: {end_time - start_time:.2f} sec")
print(f"💾 Memory: {end_memory - start_memory:.2f} MB")
print(f"🔹 Number of chunks: {len(chunks)}")
```

[5]

```
✅ Processed data in chunks.
⏱ Time: 32.17 sec
💾 Memory: 258.33 MB
🔹 Number of chunks: 262
```

```python
df = pd.read_csv(file_path)

# Downcast numeric columns
df['streams'] = pd.to_numeric(df['streams'], downcast='integer')

# Convert object columns to category if repeated values exist
for col in ['region', 'title', 'artist']:
    df[col] = df[col].astype('category')

print("✅ Optimized data types.")
print(df.dtypes)
```

[6]

Spaces: 4 Cell 11 of 12 Go Live CODEGPT

assignment 2.ipynb

Users > vineshvijayakumar > Downloads > assignment 2.ipynb > M↓ HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt

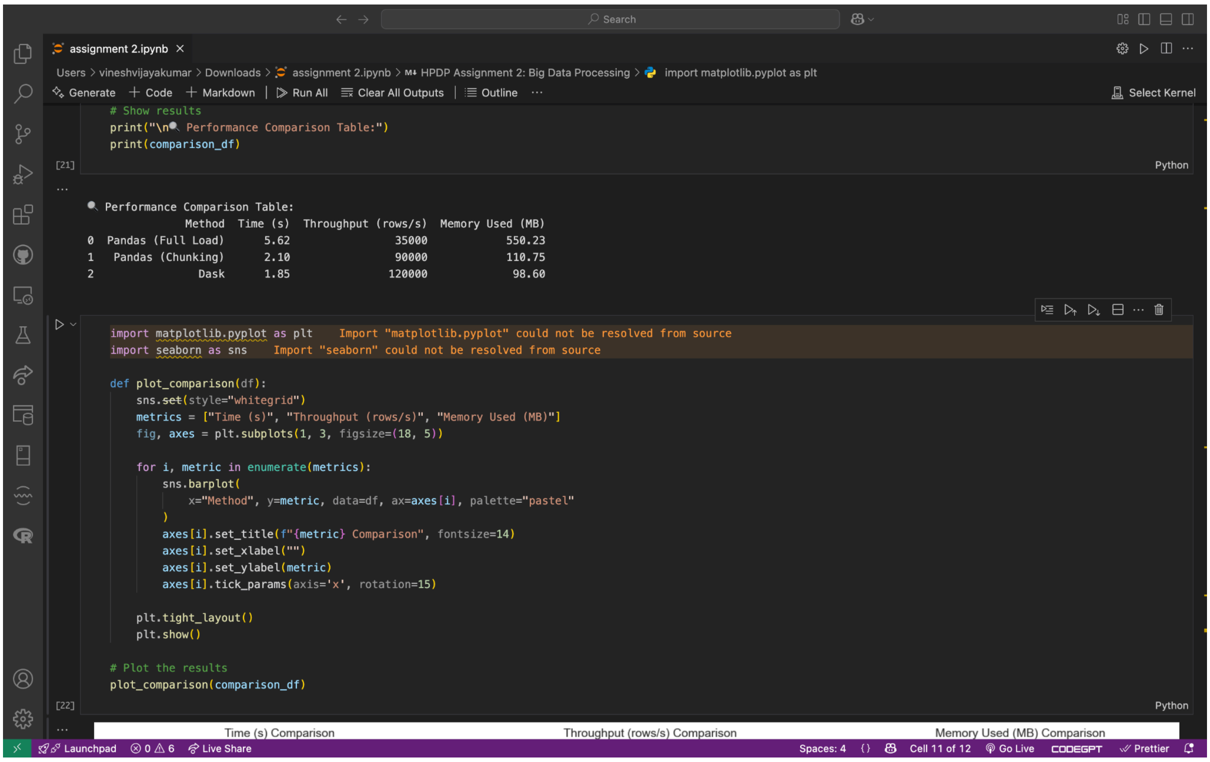
```
3  MOVE_DOWN   53270.0
4    MOVE_UP   45251.0
```

```python
import dask.dataframe as dd    Import "dask.dataframe" could not be resolved
import time
import psutil

file_path = "/Users/vineshvijayakumar/Downloads/charts.csv"

# Start timer and memory tracking
start_time = time.time()
start_memory = psutil.Process().memory_info().rss / (1024 ** 2)

# Read CSV using Dask with error-tolerant settings
ddf = dd.read_csv(file_path, assume_missing=True)

# Force computation to get preview
head = ddf.head()

# End timer and memory
end_time = time.time()
end_memory = psutil.Process().memory_info().rss / (1024 ** 2)

# Display results
print("✅ Dask Load (Parallel Processing)")
print(f"⏱ Time: {end_time - start_time:.2f} seconds")
print(f"💾 Memory Used: {end_memory - start_memory:.2f} MB")
print("🔹 Sample Records:")
print(head)
```

[10]

```
✅ Dask Load (Parallel Processing)
⏱ Time: 0.79 seconds
💾 Memory Used: 441.72 MB
🔹 Sample Records:
                    title  rank        date  \
```

Spaces: 4 Cell 11 of 12 Go Live CODEGPT



assignment 2.ipynb
Users > vineshvijayakumar > Downloads > assignment 2.ipynb > M↓ HPDP Assignment 2: Big Data Processing > import matplotlib.pyplot as plt
Generate
Code
Markdown
Run All
Clear All Outputs
Outline
Select Kernel
# Show results
print("\n🔍 Performance Comparison Table:")
print(comparison_df)
[21]
Python
🔍 Performance Comparison Table:
Method Time (s) Throughput (rows/s) Memory Used (MB)
0 Pandas (Full Load) 5.62 35000 550.23
1 Pandas (Chunking) 2.10 90000 110.75
2 Dask 1.85 120000 98.60
import matplotlib.pyplot as plt   Import "matplotlib.pyplot" could not be resolved from source
import seaborn as sns   Import "seaborn" could not be resolved from source
def plot_comparison(df):
sns.set(style="whitegrid")
metrics = ["Time (s)", "Throughput (rows/s)", "Memory Used (MB)"]
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for i, metric in enumerate(metrics):
sns.barplot(
x="Method", y=metric, data=df, ax=axes[i], palette="pastel"
)
axes[i].set_title(f"{metric} Comparison", fontsize=14)
axes[i].set_xlabel("")
axes[i].set_ylabel(metric)
axes[i].tick_params(axis='x', rotation=15)
plt.tight_layout()
plt.show()
# Plot the results
plot_comparison(comparison_df)
[22]
Python
Time (s) Comparison
Throughput (rows/s) Comparison
Memory Used (MB) Comparison
Launchpad
0 ⚠ 6
Live Share
Spaces: 4
Cell 11 of 12
Go Live
CODEGPT
Prettier