**ACADEMIC TASK - 2 CSE316**

(Operating Systems)

**COMPUTER SCIENCE AND ENGINEERING**

**Section: K23DW Submitted by:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Reg. No.** | **Roll No.** |
| Priyanshu Kumar | 12303345 | 01 |
| S. Sai Vivek | 12300999 | 02 |
| Chandhan Kumar | 12313636 | 03 |

**Submitted to:**

**Dr. Gurbinder Singh Brar**



**Project Report :**

***AI-Enhanced Data Management System for Data Science in an Operating System***

# Project Overview

This project focuses on integrating artificial intelligence techniques with data management systems to enhance data handling and analysis within an operating system environment. The system optimizes data storage, retrieval, and processing to support data science tasks such as machine learning and statistical analysis.

Goals and Expected Outcomes

* Efficient Storage and Retrieval of Data: Optimize data access processes to make use of AI-enabled methods for swifter access to data.
* Efficient Data Processing: Enhance computational efficiency regarding machine learning models or statistical evaluation of data.
* Automation and Optimization: Reduce manual intervention by adopting data management techniques enabled by AI.
* scalability and performance: approach towards designing a system that efficiently manages changing volumes of data.

Scope

* Automated Data Management: AI can organize, classify, and retrieve data in real-time.
* Real-Time Processing: Algorithms based on AI can make data processing more efficient.
* Predictive Analytics: Machine learning models can help identify trends and improve data-driven decisions.
* Security and Integrity: AI can help detect anomalies and ensure safe trading of data.

# Module-Wise Breakdown

1. Data Storage Module

AI-optimized methods for data retrieval and indexing.

Efficient handling of both structured and unstructured data: Data in different formats can be stored effectively.

Overhead: Cloud-based solutions for scaling out to meet ever-changing data demands.

2. AI-Driven Processing Module

Improve data and categorization, and predictive insights for users.

Optimized automation: Indexing applies AI technology to ensure the quickest queries and retrieval uses artificial intelligence techniques.

Memory use: Intelligently allocate and clean up memory to ensure efficient use.

3. Data Visualization & Analysis Module

Charts, and real-time dashboards, depict data processing and statistics users visualize and utilizing data is processed. Interactive filtering and custom sorting: Users can apply filters to understand previous data. Data Trend: AI-optimized insights to understand your data processing and trends longer time frame.

# Functionalities

* Data Storage Optimization: The effective allocation and retrieval of structured and unstructured data.
* AI-Enabled Processing: Techniques for machine learning are employed in automating and optimizing data processing.
* Real-Time Insights: Dynamic dashboards delivered relevant key performance indicators of users.
* Security and Anomaly Detection: AI identifies anomalous data patterns and tracks authorized access.
* Predictive Modeling: AI-based forecasting for data utilization and trends.

# Technology Used

* Programming Language

Python (for AI and data processing)

JavaScript (for frontend visualization)

SQL & NoSQL Databases (PostgreSQL, MongoDB)

* Libraries & Tools

AI & Machine Learning: TensorFlow, PyTorch, Scikit-learn

Data Visualization: Matplotlib, Plotly, D3.js

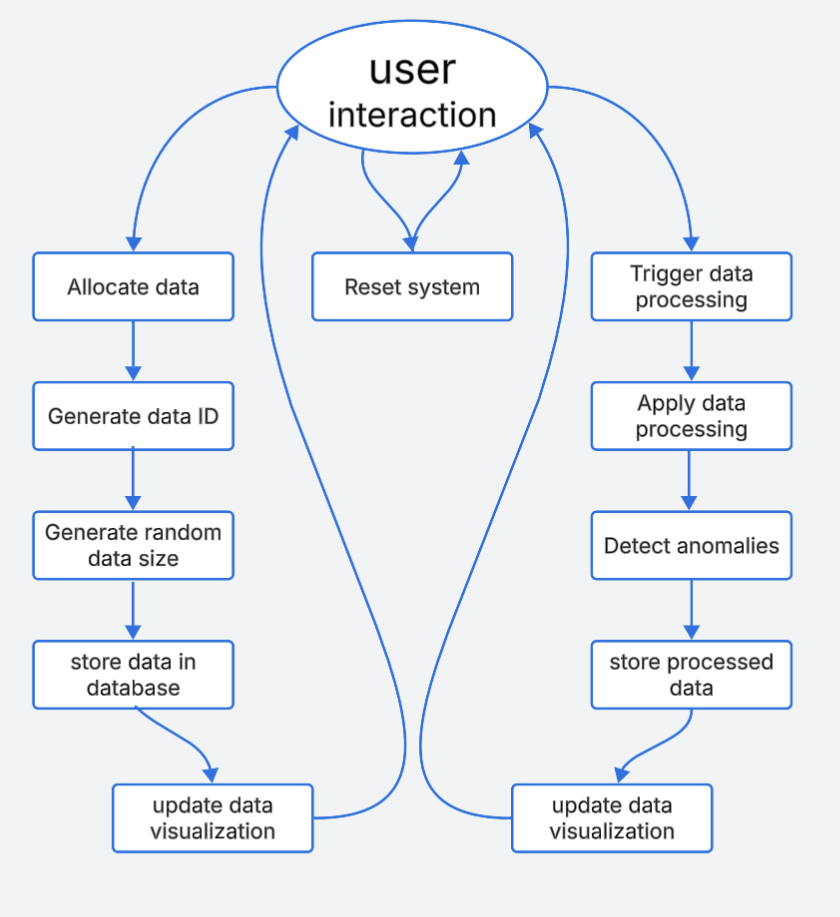
Cloud & Big Data: AWS, Azure, Google Cloud

Security & Access Control: Blockchain integration for data integrity

# System Architecture

* Data Storage Layer: AI-enhanced databases with optimized indexing and retrieval.
* Processing Layer: Machine learning models applied for data categorization and predictive analytics.
* Application Layer: Interactive dashboards and user interfaces for seamless access.
* Security Layer: AI-driven anomaly detection and secure access protocols.

# Flow Diagram



# Revision Tracking on GitHub

* Repository Name: AI-Powered Data Management System
* GitHub Link: <https://github.com/Priyanshu2835/AI-Enhanced-Data-Management-System-in-OS/upload/main>
* Commit History: Initialized project with UI setup, data loading, cleaning, analysis, system monitoring, save/export features, bug fixes, and final enhancements.

Conclusion and Future Scope

* This undertaking augments data management systems by harnessing the power of artificial intelligence methods and tools to improve data storage, retrieval, and analysis. This supports the efficient use of large datasets in an operating systems setting.
* Advanced AI Models: Use deep learning methods to enhance data classification and clustering.
* Real-Time Processing: Increase efficiency by managing data in a streaming and real-time fashion.
* Scalability: Expand and scale the system to support big data applications in a distributed computing environment.
* Cloud-Based Implementation: Implement the system in cloud environments for remote access and enhanced capability and performance.
* Enhanced Security: Utilize advanced encryption and anomaly detection methods for better data management security.

# References

* "Artificial Intelligence: A Guide for Thinking Humans" - Melanie Mitchell
* "Machine Learning with Python Cookbook" - Chris Albo
* "Database Management Systems" - Raghu Ramakrishnan

# Appendix

A. AI-Generated Project Breakdown Report

1. Project Overview

Develop an AI-driven data management system that optimizes data storage, retrieval, and processing, enhancing efficiency in machine learning and statistical analysis.

2. Module-Wise Breakdown

Module 1: GUI (Frontend Interface)

Purpose: Create a user-friendly and interactive interface for data management and AI-based analytics.

Role: Provides an intuitive dashboard to display data insights, upload datasets, and visualize AI-powered recommendations.

Module 2: AI-Based Data Processing

Purpose: Implement AI techniques for optimizing data storage, retrieval, and analysis.

Role: Handles data pre-processing, clustering, anomaly detection, and predictive analytics using machine learning models.

Module 3: Data Visualization

Purpose: Render analytical data as graphs, charts, and interactive reports.

Role: Helps users understand data trends, anomalies, and insights through visual representation.

3. Functionalities

Module 1: GUI (Frontend Interface)

* Interactive dashboard for uploading and managing datasets.
* Data preview and filtering options.
* AI-based insights display.

Module 2: AI-Based Data Processing

* Machine learning models for classification and clustering.
* Automated anomaly detection.
* Real-time data analysis and processing.

Module 3: Data Visualization

* Data distribution visualization (bar charts, histograms).
* Clustering insights (scatter plots).
* Anomaly detection results (heatmaps, graphs).

4. Technology Recommendations

The tech stack remains focused on HTML, CSS, and JavaScript, as specified:

Languages: Python (Flask for API, Pandas for data handling, TensorFlow/Scikit-learn for AI).

Module 1: GUI

Tools:

Flask, Bootstrap (for responsive design), JavaScript (for interactive elements)  
Example: Use Flask templates for dynamic content and Bootstrap for a structured layout.

Module 2: Logic

Tools:

Pandas (for data handling), Scikit-learn (for ML models), TensorFlow/Keras (for deep learning).

Example: Implement KMeans for clustering and Isolation Forest for anomaly detection.

Module 3: Data Visualization

Libraries:

Matplotlib, Seaborn, Plotly for interactive and insightful visualizations.

Example: Use Seaborn for heatmaps and Matplotlib for scatter plots.

5. Execution Plan

Here’s a step-by-step guide to implement the visualizer, inspired by the screenshot:

Step 1: Project Setup (1-2 hours)

Task: Set up the project structure.

Create app.py (backend), templates/ (HTML files), and static/ (CSS/JS files)

Step 2: Build GUI Layout

Task: Create an intuitive dashboard using Bootstrap.

Step 3: Implement AI-Based Processing (4-6 hours)

Develop functions for data pre-processing, feature scaling, and AI model execution.

Integrate machine learning algorithms for clustering and anomaly detection.

Step 4: Integrate Data Visualization (3-4 hours)

Generate dynamic plots and charts using Matplotlib/Seaborn.

Display real-time AI-based data insights.

Step 5: Testing & Deployment (2-3 hours)

Test all functionalities and optimize performance.

Deploy on a local/remote server (Heroku, AWS, or PythonAnywhere).

1. Problem Statement

Develop a system that integrates AI techniques with data management to optimize data storage, retrieval, and analysis. The system enhances the efficiency of data science tasks such as machine learning, statistical analysis, and real-time anomaly detection.

1. Solution/Code *Python Code*

import os

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from IPython.display import display, clear\_output

import ipywidgets as widgets

from datetime import datetime

import time

import psutil

import platform

class AIEnhancedDataManager:

def \_\_init\_\_(self, directory=None):

self.directory = directory if directory else os.getcwd()

self.current\_df = None

self.file\_history = []

self.setup\_ui()

def setup\_ui(self):

self.file\_dropdown = widgets.Dropdown(

options=self.list\_files(),

description='📂 Dataset:',

disabled=False,

style={'description\_width': 'initial'}

)

self.load\_btn = widgets.Button(description='Load Data', button\_style='success')

self.clean\_btn = widgets.Button(description='Clean Data', button\_style='info', disabled=True)

self.analyze\_btn = widgets.Button(description='Analyze', button\_style='warning', disabled=True)

self.system\_btn = widgets.Button(description='System Info', button\_style='primary')

self.save\_btn = widgets.Button(description='Save Data', button\_style='danger', disabled=True)

self.load\_btn.on\_click(self.on\_load\_click)

self.clean\_btn.on\_click(self.on\_clean\_click)

self.analyze\_btn.on\_click(self.on\_analyze\_click)

self.system\_btn.on\_click(self.on\_system\_click)

self.save\_btn.on\_click(self.on\_save\_click)

self.progress = widgets.FloatProgress(value=0, min=0, max=100, description='Progress:')

self.status\_output = widgets.Output()

self.data\_output = widgets.Output()

self.system\_output = widgets.Output()

self.tabs = widgets.Tab(children=[self.data\_output, self.system\_output])

self.tabs.set\_title(0, 'Data View')

self.tabs.set\_title(1, 'System Info')

self.ui = widgets.VBox([

widgets.HTML("<h1>📊 AI-Powered Data Manager</h1>"),

widgets.HBox([self.file\_dropdown, self.load\_btn]),

widgets.HTML("<h3>Actions:</h3>"),

widgets.HBox([self.clean\_btn, self.analyze\_btn, self.system\_btn, self.save\_btn]),

self.progress,

self.status\_output,

self.tabs

])

def show(self):

display(self.ui)

def update\_progress(self, value, description=None):

self.progress.value = value

if description:

self.progress.description = description

def log\_status(self, message, is\_error=False):

with self.status\_output:

timestamp = datetime.now().strftime("%H:%M:%S")

print(f"❌ [{timestamp}] {message}" if is\_error else f"✅ [{timestamp}] {message}")

def list\_files(self):

try:

files = [f for f in os.listdir(self.directory) if f.endswith('.csv')]

return files if files else ["No CSV files found"]

except Exception as e:

self.log\_status(f"Error listing files: {str(e)}", True)

return ["Error listing files"]

def refresh\_file\_list(self):

self.file\_dropdown.options = self.list\_files()

def on\_load\_click(self, btn):

filename = self.file\_dropdown.value

if filename == "No CSV files found":

self.log\_status("No CSV files available to load", True)

return

self.update\_progress(0, "Loading...")

self.current\_df = self.load\_data(filename)

if self.current\_df is not None:

self.update\_progress(100, "Ready")

self.clean\_btn.disabled = False

self.analyze\_btn.disabled = False

self.save\_btn.disabled = False

self.file\_history.append({'timestamp': datetime.now(), 'filename': filename, 'action': 'load'})

def load\_data(self, filename):

try:

with self.data\_output:

clear\_output()

print(f"⏳ Loading {filename}...")

for i in range(10):

time.sleep(0.05)

self.update\_progress(i\*10)

df = pd.read\_csv(os.path.join(self.directory, filename))

with self.data\_output:

clear\_output()

display(widgets.HTML(f"<h2>📋 {filename} Preview</h2>"))

display(df.head(5))

self.log\_status(f"Successfully loaded {filename} ({len(df)} rows)")

return df

except Exception as e:

self.log\_status(f"Error loading {filename}: {str(e)}", True)

with self.data\_output:

clear\_output()

display(widgets.HTML(f"<div style='color:red'>❌ Error loading file: {str(e)}</div>"))

return None

def on\_clean\_click(self, btn):

if self.current\_df is None:

self.log\_status("No data loaded to clean", True)

return

self.update\_progress(0, "Cleaning...")

try:

with self.data\_output:

clear\_output()

print("🧹 Cleaning data...")

steps = ["Handling missing values", "Standardizing formats", "Removing duplicates", "Fixing data types"]

for i, step in enumerate(steps):

time.sleep(0.5)

self.update\_progress((i+1)\*25, f"Cleaning: {step}")

with self.data\_output:

clear\_output()

print(f"🔧 {step}...")

self.current\_df = self.clean\_data(self.current\_df)

with self.data\_output:

clear\_output()

display(widgets.HTML("<h2>✨ Cleaned Data Preview</h2>"))

display(self.current\_df.head(5))

self.log\_status("Data cleaning completed successfully")

self.update\_progress(100, "Ready")

except Exception as e:

self.log\_status(f"Error during cleaning: {str(e)}", True)

self.update\_progress(0, "Ready")

def clean\_data(self, df):

try:

num\_cols = df.select\_dtypes(include=[np.number]).columns

cat\_cols = df.select\_dtypes(include=['object']).columns

if not num\_cols.empty:

df[num\_cols] = df[num\_cols].fillna(df[num\_cols].mean())

if not cat\_cols.empty:

df[cat\_cols] = df[cat\_cols].fillna(df[cat\_cols].mode().iloc[0])

initial\_rows = len(df)

df = df.drop\_duplicates()

removed = initial\_rows - len(df)

if removed > 0:

self.log\_status(f"Removed {removed} duplicate rows")

return df

except Exception as e:

raise Exception(f"Data cleaning failed: {str(e)}")

def on\_analyze\_click(self, btn):

if self.current\_df is None:

self.log\_status("No data loaded to analyze", True)

return

self.update\_progress(0, "Analyzing...")

try:

with self.data\_output:

clear\_output()

print("🔍 Analyzing data...")

self.generate\_summary(self.current\_df)

self.update\_progress(100, "Ready")

except Exception as e:

self.log\_status(f"Error during analysis: {str(e)}", True)

self.update\_progress(0, "Ready")

def generate\_summary(self, df):

with self.data\_output:

clear\_output()

display(widgets.HTML("<h2>📊 Data Analysis Report</h2>"))

display(widgets.HTML("<h3>📋 Basic Statistics</h3>"))

display(df.describe())

missing = df.isnull().sum()

display(widgets.HTML("<h3>🔍 Missing Values</h3>"))

display("No missing values found!" if missing.sum() == 0 else missing)

numeric\_df = df.select\_dtypes(include=[np.number])

if not numeric\_df.empty:

display(widgets.HTML("<h3>🌡️ Feature Correlation</h3>"))

plt.figure(figsize=(10, 8))

sns.heatmap(numeric\_df.corr(), annot=True, cmap="coolwarm", center=0)

plt.title("Feature Correlation Heatmap")

plt.show()

display(widgets.HTML("<h3>📈 Feature Distributions</h3>"))

for col in numeric\_df.columns:

plt.figure(figsize=(8, 4))

sns.histplot(numeric\_df[col], kde=True)

plt.title(f"Distribution of {col}")

plt.show()

cat\_cols = df.select\_dtypes(include=['object']).columns

if not cat\_cols.empty:

display(widgets.HTML("<h3>📊 Categorical Analysis</h3>"))

for col in cat\_cols:

plt.figure(figsize=(8, 4))

sns.countplot(y=col, data=df, order=df[col].value\_counts().index)

plt.title(f"Distribution of {col}")

plt.show()

def on\_system\_click(self, btn):

self.update\_progress(0, "Gathering system info...")

try:

with self.system\_output:

clear\_output()

print("🖥️ Gathering system information...")

for i in range(1, 6):

time.sleep(0.2)

self.update\_progress(i\*20)

self.display\_system\_info()

self.update\_progress(100, "Ready")

self.log\_status("System information displayed")

except Exception as e:

self.log\_status(f"Error gathering system info: {str(e)}", True)

self.update\_progress(0, "Ready")

def display\_system\_info(self):

with self.system\_output:

clear\_output()

display(widgets.HTML("<h2>🖥️ System Integration</h2>"))

sys\_info = [

("System", platform.system()), ("Node Name", platform.node()),

("Release", platform.release()), ("Version", platform.version()),

("Machine", platform.machine()), ("Processor", platform.processor()),

("Python Version", platform.python\_version())

]

display(pd.DataFrame(sys\_info, columns=["Property", "Value"]))

mem = psutil.virtual\_memory()

mem\_info = [

("Total", f"{mem.total/1e9:.2f} GB"), ("Available", f"{mem.available/1e9:.2f} GB"),

("Used", f"{mem.used/1e9:.2f} GB"), ("Percentage", f"{mem.percent}%")

]

display(pd.DataFrame(mem\_info, columns=["Metric", "Value"]))

cpu\_info = [

("Physical Cores", psutil.cpu\_count(logical=False)),

("Logical Cores", psutil.cpu\_count(logical=True)),

("Current Usage", f"{psutil.cpu\_percent()}%"),

("Current Frequency", f"{psutil.cpu\_freq().current:.2f} MHz")

]

display(pd.DataFrame(cpu\_info, columns=["Property", "Value"]))

disk = psutil.disk\_usage('/')

disk\_info = [

("Total", f"{disk.total/1e9:.2f} GB"), ("Used", f"{disk.used/1e9:.2f} GB"),

("Free", f"{disk.free/1e9:.2f} GB"), ("Percentage", f"{disk.percent}%")

]

display(pd.DataFrame(disk\_info, columns=["Metric", "Value"]))

process = psutil.Process()

proc\_info = [

("PID", process.pid), ("Name", process.name()),

("Status", process.status()),

("Memory Usage", f"{process.memory\_info().rss/1e6:.2f} MB"),

("CPU Percent", f"{process.cpu\_percent()}%")

]

display(pd.DataFrame(proc\_info, columns=["Property", "Value"]))

fig, ax = plt.subplots(1, 2, figsize=(12, 4))

mem = psutil.virtual\_memory()

ax[0].pie([mem.used, mem.available], labels=['Used', 'Available'],

autopct='%1.1f%%', colors=['#ff9999','#66b3ff'])

ax[0].set\_title('Memory Usage')

cpu\_percent = psutil.cpu\_percent(percpu=True)

ax[1].bar(range(len(cpu\_percent)), cpu\_percent, color='#99ff99')

ax[1].set\_title('CPU Core Usage')

ax[1].set\_xlabel('Core')

ax[1].set\_ylabel('Usage %')

ax[1].set\_ylim(0, 100)

plt.tight\_layout()

plt.show()

def on\_save\_click(self, btn):

if self.current\_df is None:

self.log\_status("No data to save", True)

return

save\_name = widgets.Text(value=f"cleaned\_{self.file\_dropdown.value}", description='Save as:')

confirm\_btn = widgets.Button(description='Confirm Save', button\_style='success')

cancel\_btn = widgets.Button(description='Cancel')

save\_dialog = widgets.VBox([

widgets.HTML("<h3>💾 Save Dataset</h3>"),

save\_name,

widgets.HBox([confirm\_btn, cancel\_btn])

])

def on\_confirm\_save(btn):

filename = save\_name.value

if not filename.endswith('.csv'):

filename += '.csv'

self.save\_data(self.current\_df, filename)

save\_dialog.close()

def on\_cancel\_save(btn):

save\_dialog.close()

confirm\_btn.on\_click(on\_confirm\_save)

cancel\_btn.on\_click(on\_cancel\_save)

with self.data\_output:

clear\_output()

display(save\_dialog)

def save\_data(self, df, filename):

filepath = os.path.join(self.directory, filename)

try:

self.update\_progress(0, "Saving...")

if os.path.exists(filepath):

confirm = input(f"⚠️ {filename} exists. Overwrite? (y/n): ")

if confirm.lower() != 'y':

self.log\_status("Save cancelled")

self.update\_progress(0, "Ready")

return

df.to\_csv(filepath, index=False)

self.update\_progress(100, "Ready")

self.log\_status(f"Data saved successfully to {filename}")

self.refresh\_file\_list()

self.file\_history.append({

'timestamp': datetime.now(),

'filename': filename,

'action': 'save'

})

except Exception as e:

self.log\_status(f"Error saving file: {str(e)}", True)

self.update\_progress(0, "Ready")

if \_\_name\_\_ == "\_\_main\_\_":

try:

get\_ipython()

manager = AIEnhancedDataManager()

manager.show()

except:

print("Running in basic mode...")

manager = AIEnhancedDataManager()

print("\nAvailable datasets:", manager.list\_files())

filename = input("Enter dataset filename to load: ")

df = manager.load\_data(filename)

if df is not None:

df = manager.clean\_data(df)

manager.generate\_summary(df)

if input("Save cleaned dataset? (yes/no): ").strip().lower() == 'yes':

manager.save\_data(df, "cleaned\_" + filename)