



**Project Title:**

**AI-Powered Personalized Study Planner**

**Group Members:**

- Rehan Asif - 70147344
- Khudija Kashif - 70147016

**Submitted to: DR. HAMEDOON**

## **Introduction:**

The **AI-Powered Personalized Study Planner** is a Jupyter Notebook application designed to help students organize and optimize their study schedules using data-driven predictions and automated planning. The system trains a **Linear Regression model** on historical or synthetic data—including subject difficulty, target marks, and hours studied—to estimate how much time is required for each subject.

Users interact with the planner through an interface built using **ipywidgets**, where they can add subjects, set goal marks, select difficulty levels, and define their available study hours for each day of the week. Once the inputs are provided, the application breaks the predicted study time into **one-hour blocks** and distributes them across a **14-day schedule** based on the user's availability.

To support real-world usage, the planner includes **Google Calendar integration**, allowing users to export the generated study schedule directly into their calendar as individual events.

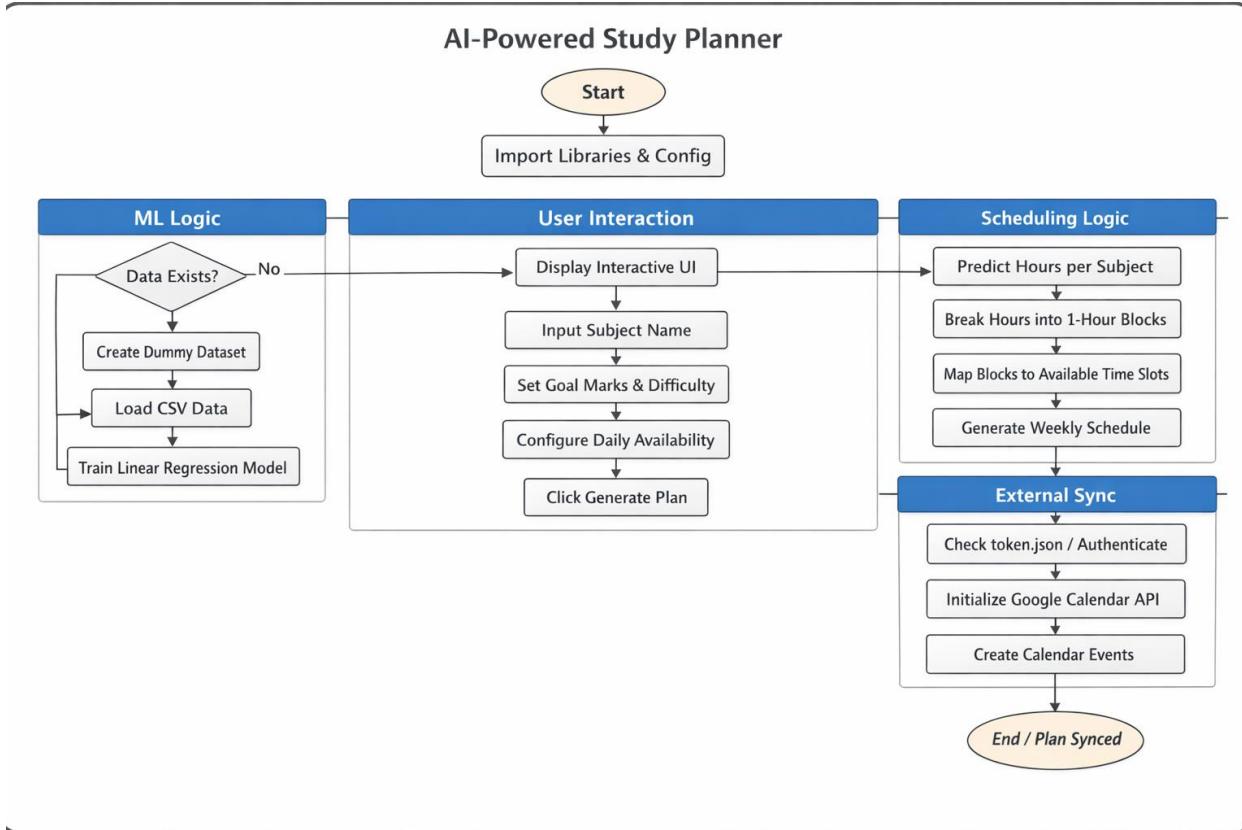
## **Technologies Used**

The project is built using a Python-based data science stack. **Scikit-learn** is used to implement the Linear Regression model, while **Pandas** and **NumPy** handle data processing and synthetic dataset generation. The interactive interface is created using **ipywidgets**, enabling sliders, buttons, and input fields within the notebook.

For calendar synchronization, the project uses the **Google APIs Client Library** along with **Google Auth** to manage OAuth2 authentication and event creation. Scheduling logic is handled using Python's **datetime** module, and **Matplotlib** and **Seaborn** are included for optional data visualization.

- **AI-Based Study Time Prediction**  
Estimates required study hours based on subject difficulty and performance goals.
- **Personalized Weekly Scheduling**  
Allows users to define daily study start times and available hours.
- **Interactive Notebook Dashboard**  
Enables subject management and schedule generation without manual coding.
- **Google Calendar Synchronization**  
Automatically exports study sessions as calendar events.
- **Automatic Data Generation**  
Creates dummy training data when no prior dataset is available.

## Flow Diagram:



## Project Code:

```

# 1. Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import ipywidgets as widgets
from IPython.display import display, clear_output
import datetime
import os, path

# 2. Load Data
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google.auth.transport import OAuth2Transport
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError

# Configuration
DATA_PATH = "data/study_data.csv"
SCOPES = ["https://www.googleapis.com/auth/calendar"]
print("Libraries imported successfully!")
# ✓ 11s
Libraries imported successfully!
  
```

**2. Train the Model**

```

def train_model():
    try:
        if not path.exists(DATA_PATH):
            print("Data file not found. Creating dummy data...")
            # Create dummy data
            df_dummy = pd.DataFrame({
                "Difficulty": np.random.randint(1, 60),
                "Hours_Required": np.random.randint(50, 90),
                "Hours_Studied": np.random.randint(1, 100)
            })
            df_dummy.to_csv(DATA_PATH, index=False)
        df = pd.read_csv(DATA_PATH)
        x = df[["Difficulty", "Hours_Obtained"]]
        y = df["Hours_Studied"]

        model = LinearRegression()
        model.fit(x, y)

        print("Model trained successfully!")
        clear_output()
    except Exception as e:
        print(f"Error training model: {e}")
        clear_output()

model = train_model()
# ✓ 6s
  
```

### 3. Google Calendar Authentication Logic

```
def get_credentials(token_json):
    creds = None
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            try:
                creds.refresh(Request())
            except Exception:
                print("Token expired and refresh failed. Please re-authenticate.")
                creds = None
        else:
            pass
    creds = None

    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            try:
                creds.refresh(Request())
            except Exception:
                print("Token expired and refresh failed. Please re-authenticate.")
                creds = None
        else:
            pass

    if not creds:
        if os.path.exists('credentials.json'):
            print("ERROR: credentials.json not found. Please place it in the project root.")
            return None
        try:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        except Exception as e:
            print(f"Authentication Error: {e}")
            return None

    # Save the credentials file for the next run
    with open('token.json', 'w') as token:
        token.write(creds_to_json())
    return creds

return build('calendar', 'v3', credentials=creds)

def create_calendar_event(service, summary, start_time, end_time):
    event = {
        'summary': summary,
        'start': {
            'dateTime': start_time.isoformat(),
            'timeZone': 'UTC',
        },
        'end': {
            'dateTime': end_time.isoformat(),
            'timeZone': 'UTC',
        },
    }
    event = service.events().insert(calendarId='primary', body=event).execute()
    print(f'Event created: {event.get("htmlLink")}'
```

#### 4. Interactive Planner Application

```

# State
subjects_list = []
generated_schedule = []

# --- WIDGETS ---

# 1. Subject Navigation
self.subjects_widget = widgets.Select(
    options=[{"label": "Subject", "value": "Subject"}, {"label": "Goal Holder", "value": "GH"}, {"label": "All"}],
    description="Select Subject", style=style
)
self.goal_widget = widgets.Text(
    value="Goal", style=style
)
self.duration_widget = widgets.Slider(
    min=0, max=100, step=1, value=50, description="Duration", style=style
)
self.difficulty_widget = widgets.Slider(
    min=1, max=5, step=1, value=3, description="Difficulty", style=style
)

# 2. Weekly Schedule Settings (Per Day)
days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
self.days_widget = widgets.Dropdown(
    options=days_of_week, value="Monday", description="Day", style=style
)

# 3. Schedule Rows
schedule_rows = []
for day in days_of_week:
    schedule_rows.append(widgets.VBox())
    for hour in range(7, 24):
        start_drop = widgets.Dropdown(
            options=[("7:00 AM", 0), ("8:00 AM", 1), ("9:00 AM", 2), ("10:00 AM", 3), ("11:00 AM", 4), ("12:00 PM", 5), ("1:00 PM", 6), ("2:00 PM", 7), ("3:00 PM", 8), ("4:00 PM", 9), ("5:00 PM", 10), ("6:00 PM", 11), ("7:00 PM", 12)], value=0, description="Start Time", style=style
        )
        duration_slider = widgets.Slider(
            min=0, max=100, step=1, value=50, description="Duration", style=style
        )
        schedule_rows[-1].append(start_drop, duration_slider)

# 4. Accordion
accordions_schedule = widget.Accordion(children=[self.schedule_widget])
accordions_schedule.set_title(0, "Configure weekly Availability (click to expand)")

# 5. Buttons
self.add_button = widgets.Button(description="Add Subject", button_style="info")
self.generate_button = widgets.Button(description="Generate Plan", button_style="primary")
self.close_button = widgets.Button(description="Close", button_style="success")
self.clear_button = widgets.Button(description="Clear Subjects", button_style="warning")
self.output_area = widgets.Output()

# --- LOGIC ---

def refresh_if_message():
    with output_area:
        clear_output()
        if message:
            print(f"({message})")
            print("")

# --- Main Logic ---

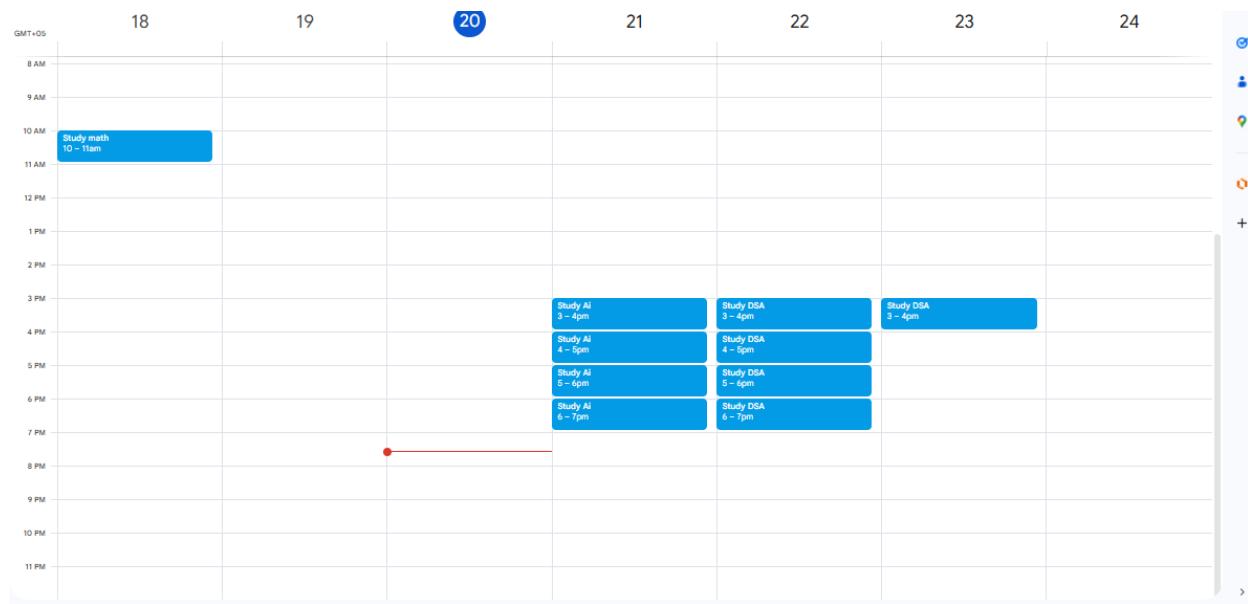
print("Current Subjects: ", len(subjects_list))
for i, s in enumerate(subjects_list):
    print(f"({i+1}), {s['name']} | Goal: {s['marks']} | Diff: {s['difficulty']}")

if new_subject:
    if generated_schedule:

```

## Output panel:

The screenshot shows the AI Study Planner interface. At the top, there's a header with 'AI Study Planner', a search bar for 'Subject' (set to 'n.g. AI'), and buttons for 'Add Subject', 'Clear Subjects', 'Goal Marks' (set to 80), and 'Difficulty' (set to 'Medium'). Below this is a section titled 'Available Study Hours' with a heading 'Configure Weekly Availability (Click to Expand)'. It lists daily start times from 10:00 to 10:00 and hours from 4 to 4. Underneath is a list of events: 'Synced 2 events!'. It shows two subjects: '1. AI | goal: 80 | diff: 1' and '2. DSA | goal: 80 | diff: 1'. The AI events are: 'Monday 10:00 : Study AI', 'Tuesday 10:00 : Study AI', 'Wednesday 10:00 : Study AI', 'Thursday 10:00 : Study AI', 'Friday 10:00 : Study AI', 'Saturday 10:00 : Study AI', and 'Sunday 10:00 : Study AI'. The DSA events are: 'Monday 10:00 : Study DSA', 'Tuesday 10:00 : Study DSA', 'Wednesday 10:00 : Study DSA', 'Thursday 10:00 : Study DSA', 'Friday 10:00 : Study DSA', and 'Saturday 10:00 : Study DSA'. At the bottom, there are buttons for 'Generate Plan' and 'Sync To Calendar'.



## Scope of Work

The scope of this project focuses on designing and implementing an AI-assisted study planning system within a Jupyter Notebook environment. The system aims to convert academic goals into a structured and realistic study schedule using machine learning and automation.

The project includes the following components:

- Development of a **Linear Regression-based machine learning model** to predict required study hours based on subject difficulty, target marks, and past study data.

- Automatic **generation of synthetic training data** when historical data is unavailable to ensure the system remains functional.
- Creation of an **interactive user interface using ipywidgets** for subject input, goal setting, and weekly availability configuration.
- Implementation of **study schedule generation logic** that breaks predicted study hours into one-hour blocks and allocates them across a 14-day period based on user availability.
- Integration with **Google Calendar** using OAuth2 authentication to export generated study sessions as calendar events.
- Handling of **capacity limitations**, including alerts when the total study load exceeds the user's available time.

**GitHub repository Link:**

**<https://github.com/RehanAsif02/AI-Powered-Personalized-Study-Planner.git>**