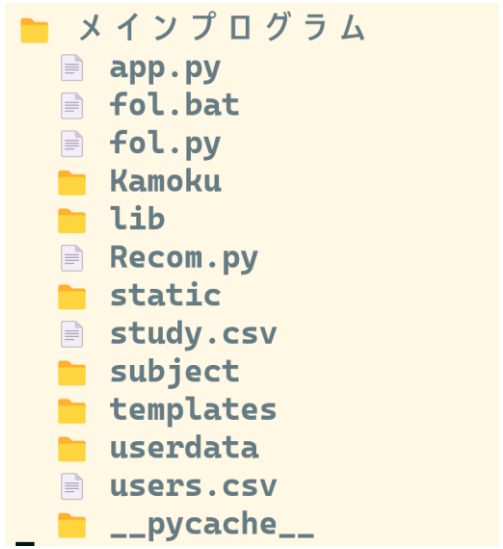
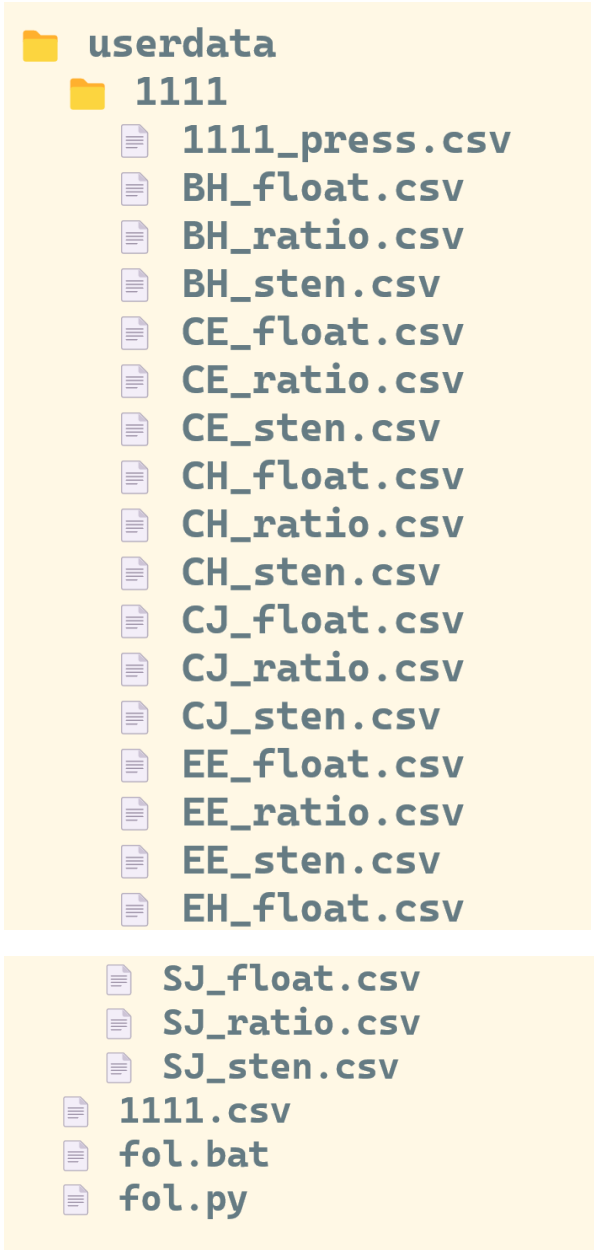
超パーフェクト学習システム解説
中市卒論クリアブック
（改）

システム全体



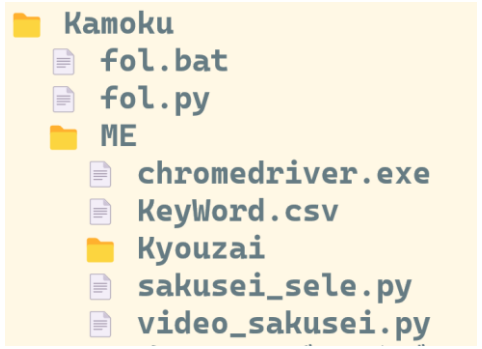
ファイル		フォルダ	
App.py	システム全体を制御	Kamoku	教材
Recom.py	清水さん作成教材選択モジュール	Static	静的データ
Study.csv	勉強時間保存	Subject	科目
Users.csv	IDとパスワード	Templates	Html
		Userdata	ユーザーごと

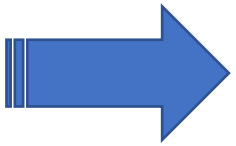
userdata

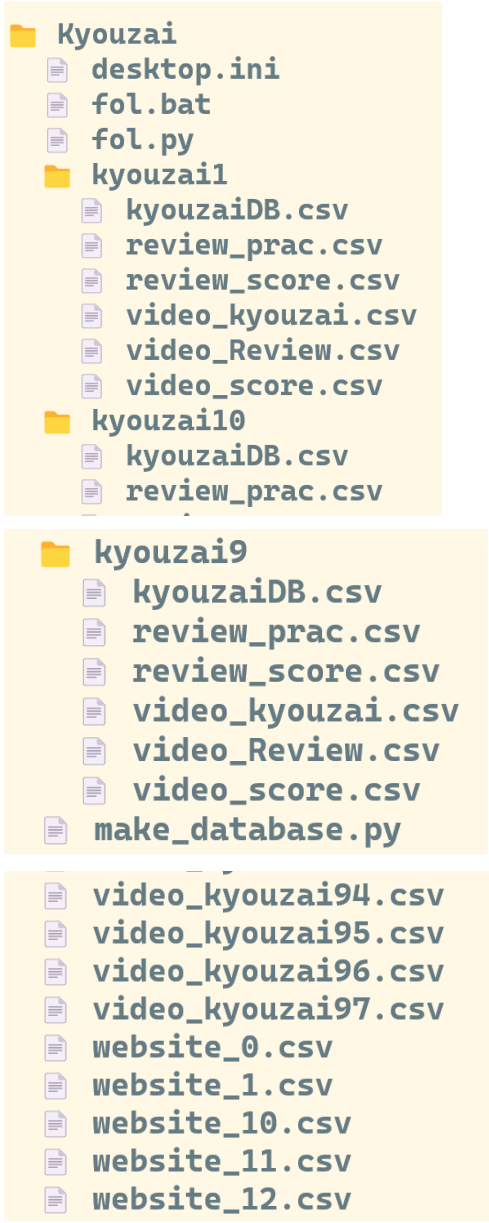


ファイル	
1111/	username
${subject}_sten.csv	Part1. 開始と終わりを指定した単元データ
${subject}_float.csv	Part2. Part1のCPM処理
${subject}_ratio.csv	Part3. Part2をスケジュール化
${username}_press.csv	Part4. 各科目の圧縮率

Kamoku

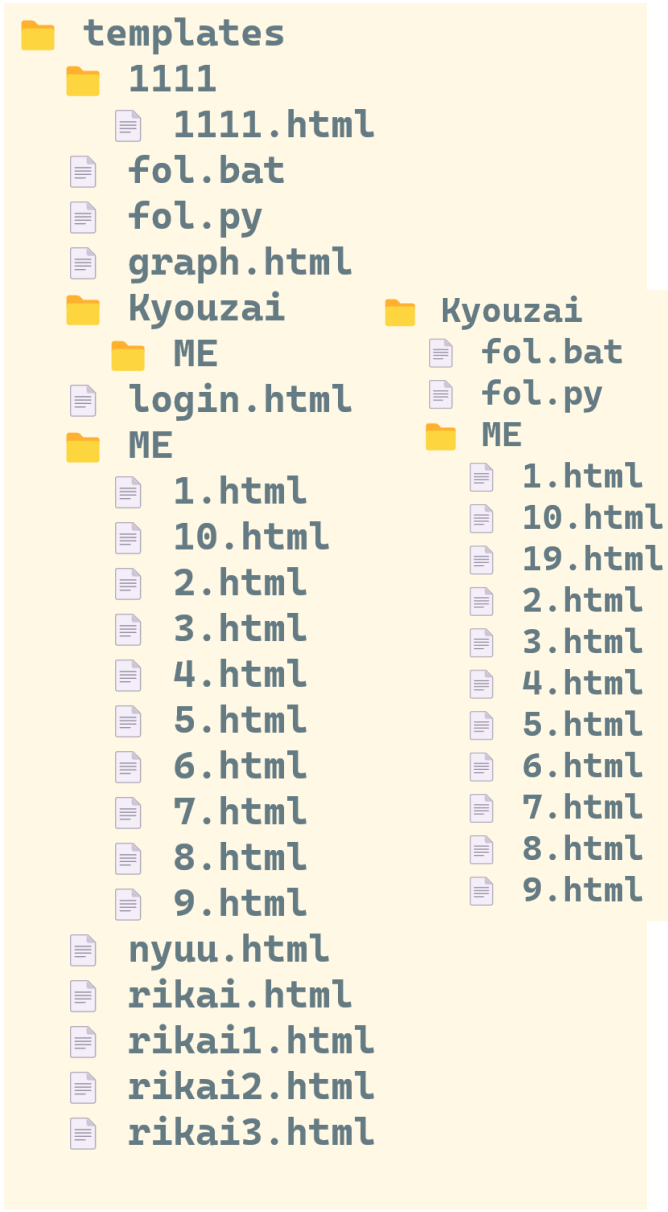






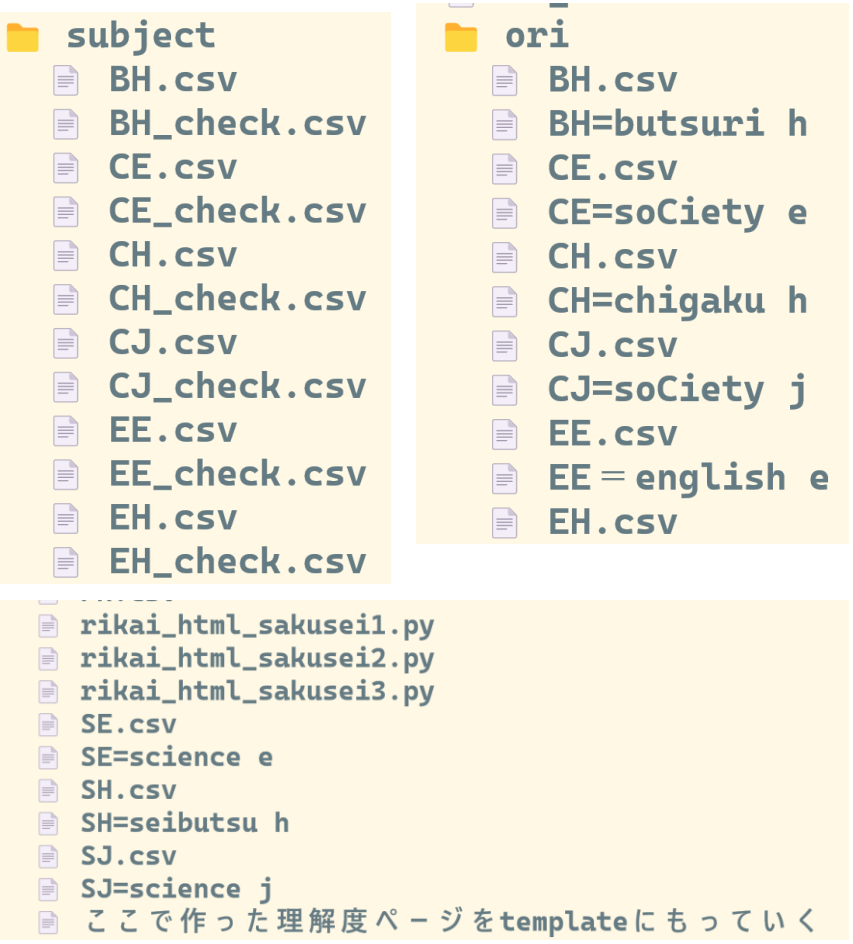
ファイル	
ME/	${Subject}
Keyword.csv	スクレイピング時のキーワード 教材画面のタイトルにも使われている
sakusei_sale.py	Seleniumでwebページをスクレイピング
Video_sakusei.py	Youtube api v3でYoutubeをスクレイピング
Website_${num}.csv	スクレイピング結果
Video_kyouzai_${num}.csv	
Make_database.py	スクレイピング結果をデータベース化
Kyouzai${num}	実際に使うデータ

templates



ファイル	
1111.html	スケジュール表示画面
Login.html	ログイン画面
Graph.html	グラフ画面の枠組みのみ
ME/${num}	教材画面
Kyouzai/ME/${num}	教材画面中間地点 キーワード選択するならここ
Nyuu.html	入力画面
Rikai1,2,3.html	理解度チェック画面小,中,高 Subject内のrikai_html_sakusei.py によって作成する
Rikai.html	ログアウトを促すhtml 別にログアウトする必要ない

subject



ファイル	
BH.csv	Butsuri Highの理解度
BH_check.csv	Butsuri Highの学習範囲
Rikai_html_sakusei1,2,3.py	Rikai1,2,3.htmlを作る
Kate.py	使用しないが，app.pyで使用されているカテゴリの表が確認できる
Ori/	年間指導計画と学習系統図から作成できる(人力)オリジナルデータを入れる

Static

ファイル	
Mainpage.css	入力画面を修飾
Style.css	いろんな場所を修飾
Subject.html	グラフページの初期値
1111/Bh.html	1111さんのButsuri Highのグラフ

App.py　必要なモジュール一覧

```
from flask import Flask, render_template, request, redirect, url_for, session,abort
import csv
import os
from os import name
import pandas as pd
import tkinter as tk
from tkinter import messagebox
import numpy as np
from datetime import datetime, timedelta, date
import networkx as nx
from pyvis.network import Network



#################################################################
####################多分自作モジュール
#################################################################

from Recom import Similarity, Informative, Concentration, Credibility, kousinn,\
                 video_kousinn, video_Credibility, Video_Concentration
```

App.py 開始時にログインページに

```python
app = Flask(__name__)
app.secret_key = 'your_secret_key'  # セッションの安全性のためのキーを設定

@app.route('/')
def home():
    return render_template('login.html')
```

ログイン

```python
##########################################################################################
#####################################ログイン
##########################################################################################
@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')

    if not username or not password:
        return render_template('login.html', login_message='ユーザー名とパスワードを入力してください')

    with open('users.csv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file)
        for row in reader:
            if row[0] == username and row[1] == password:
                # ログインが成功したらセッションにユーザー情報を保存
                session['username'] = username
                session['password'] = password
                session['name'] = row[2]
                return redirect(url_for('dashboard'))

    return render_template('login.html', login_message='ユーザー名またはパスワードが正しくありません')
```

新規作成

```python
##########################################################################################
#####################################新規登録
##########################################################################################
@app.route('/register', methods=['POST'])
def register():
    new_username = request.form.get('new_username')
    new_password = request.form.get('new_password')
    new_name = request.form.get('new_name')

    if not new_username or not new_password or not new_name:
        return render_template('login.html', registration_message='全ての項目を入力してください')

    # ユーザーIDが数字で構成されているか確認
    if not new_username.isdigit():
        return render_template('login.html', registration_message='ユーザーIDは数字である必要があります')

    with open('users.csv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file)
        for row in reader:
            if row[0] == new_username:
                return render_template('login.html', registration_message='このユーザー名はすでに使用されています')

    with open('users.csv', 'a', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow([new_username, new_password, new_name])

    return render_template('login.html', registration_message='新しいユーザーが登録されました')
```

App.py　カテゴリ作成

```
#########################################################################################################
#######################################カテゴリ選択
#########################################################################################################
# CSVファイルからデータを読み込む関数
def read_unit_data(file_path):
    df = pd.read_csv(file_path, sep=",", dtype={"単元番号": str, "単元名": str})
    df = df.dropna(subset=["単元番号", "単元名"])  # NaNを含む行を削除
    unit_numbers = df["単元番号"].tolist()
    unit_names = df["単元名"].tolist()
    return unit_numbers, unit_names

# カテゴリと略称のデータ
categories_data = {
    "category": ["小学校国語", "小学校算数", "小学校英語", "小学校理科", "小学校社会",
                 "中学国語", "中学数学", "中学英語", "中学理科", "中学社会",
                 "高校国語", "高校数学", "高校英語",
                 "高校物理", "高校科学", "高校生物", "高校地学",
                 "高校地理", "高校歴史", "高校公共"],
    "path": ["JE", "ME", "EE", "SE", "CE", "JJ", "MJ", "EJ", "SJ", "CJ", "JH", "MH", "EH", "BH", "KH", "SH", "CH", "GH", "HH", "PH"]
}

# カテゴリデータから単元データを取得
unit_data = {}
for i, category in enumerate(categories_data["category"]):
    unit_numbers, unit_names = read_unit_data(f"subject/ori/{categories_data['path'][i]}.csv")
    unit_data[category] = {"unit_numbers": unit_numbers, "unit_names": unit_names}

# DataFrameを作成
categories = pd.DataFrame({
    "category": categories_data["category"],
    "subjects": [unit_data[category]["unit_names"] for category in categories_data["category"]],
    "units": [unit_data[category]["unit_numbers"] for category in categories_data["category"]],
    "path": categories_data["path"]
})

# 結果を表示
# print(categories)
```

App.py 教材の画面に飛ぶ

```
#########################################################################################################
#########################################教材の画面に飛ぶ
#########################################################################################################


@app.route('/Kyouzai/<category>/<unit>')
def kyouzai_start(category, unit):

    path = categories[categories['category'] == category]['path'].values[0]
    # print(path)

    template_path = f'Kyouzai/{path}/{unit}.html'

    return render_template(template_path)


@app.route('/Kyouzai/<path>/<unit>')
def redirect_to_template(path, unit):
    # Redirect to the corresponding HTML template
    return redirect(url_for('kyouzai_start', category=path, unit=unit))
```

名前とID表示

```
#########################################################################################################
#########################################2ページ目左上
#########################################################################################################
@app.route('/dashboard')
def dashboard():
    # セッションからユーザー情報を取得
    username = session.get('username')
    password = session.get('password')
    name = session.get('name')

    # ユーザーがログインしていない場合はログインページにリダイレクト
    if not username or not password or not name:
        return redirect(url_for('home'))

    user_info = {'id': 1, 'username': username, 'password': password, 'name': name}
    return render_template('nyuu.html', user_info=user_info)
```

学習時間設定

```
#########################################################################################################
#########################################2ページ目右上
#########################################################################################################
@app.route('/save_total_hours', methods=['POST'])
def save_total_hours():
    total_hours_data = request.form.get('total_hours_data')

    if total_hours_data:
        with open('study.csv', 'a', newline='', encoding='utf-8') as file:
            file.write(total_hours_data)

        return 'Total hours data saved successfully', 200
    else:
        return 'Error saving total hours data', 400


@app.route('/check_duplicate_username', methods=['GET'])
def check_duplicate_username():
    username_to_check = request.args.get('username')

    with open('study.csv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file)
        for row in reader:
            if row[0] == username_to_check:
                return 'true'

    return 'false'

@app.route('/delete_and_save_total_hours', methods=['POST'])
def delete_and_save_total_hours():
    username = request.form.get('username')
    total_hours_data = request.form.get('total_hours_data')

    # ユーザー名が同じ行が存在するか確認
    with open('study.csv', 'r', encoding='utf-8') as file:
        reader = csv.reader(file)
        data = list(reader)

    with open('study.csv', 'w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        for row in data:
            if row[0] != username:
                writer.writerow(row)

        # 新しいデータを追加
        writer.writerow(total_hours_data.split(','))

    return 'Data deleted and saved successfully', 200
```

App.py 予定作成

```python
#########################################################################################################
#####################################2ページ目左下
#########################################################################################################
@app.route('/write_fullcalendar_data', methods=['POST'])
def write_fullcalendar_data():
    username = request.form.get('username')
    csv_data = request.form.get('csv_data')

    # userdataディレクトリが存在しない場合は作成
    user_data_dir = 'userdata'
    if not os.path.exists(user_data_dir):
        os.makedirs(user_data_dir)

    # CSVファイルにイベントデータを書き込み
    with open(f'{user_data_dir}/{username}.csv', 'w', newline='', encoding='utf-8') as file:
        file.write(csv_data)

    return 'FullCalendar data written to CSV successfully', 200


@app.route('/read_fullcalendar_data', methods=['GET'])
def read_fullcalendar_data():
    username = request.args.get('username')

    # userdataディレクトリが存在しない場合は、データがまだ存在しないとみなして空のCSVデータを返す
    user_data_dir = 'userdata'
    if not os.path.exists(user_data_dir):
        return '', 200

    # CSVファイルからFullCalendarデータを読み込み
    try:
        with open(f'{user_data_dir}/{username}.csv', 'r', encoding='utf-8') as file:
            csv_data = file.read()
        return csv_data, 200
    except FileNotFoundError:
        return '', 200
```

App.py ページ遷移

```
#########################################################################################################
#######################################2ページ目右下
#########################################################################################################


#main_page関数を押すと更新されてしまうのでやっつけ別ルート
@app.route('/mainpage')
def mainpage():
    username = session.get('username')

    # ユーザーのHTMLファイルが存在するか確認
    template_path = f'templates/{username}/{username}.html'
    if not os.path.exists(template_path):
        abort(404, description='先にスケジュールを作成してください')

    return render_template(f'{username}/{username}.html')

#ついで
@app.route('/graphpage')
def graphpage():
    username = session.get('username')
    template_path =f'static/subjects/{username}/ME.html'#MEじゃなくてもいいどうせ全部作るからなんでもいい
    if not os.path.exists(template_path):
        abort(404, description='グラフが作成されていません')
    return render_template('graph.html', username=username)


@app.route('/rikai_page1')
def rikai_page1():
    return render_template('rikai1.html')

@app.route('/rikai2_page')
def rikai_page2():
    return render_template('rikai2.html')

@app.route('/rikai3_page')
def rikai_page3():
    return render_template('rikai3.html')
```

```
@app.route('/main_page')
def main_page():
    # ユーザー名を取得
    username = session.get('username')

    # userdata/{username}.csvが存在するか確認
    user_csv_path = f"userdata/{username}.csv"
    if not os.path.exists(user_csv_path):
        abort(404, description='予定データが存在しません')


    for idx, row in categories.iterrows():
        category = row["category"]
        path = row['path']
        input_ori_filename = f"subject/ori/{path}.csv"
        input_schedule_filename = f"subject/{path}_check.csv"
        output_directory = f"userdata/{username}"
        output_filename = f"{output_directory}/{path}_sten.csv"

        # 出力ディレクトリが存在しない場合は作成する
        if not os.path.exists(output_directory):
            os.makedirs(output_directory)

        target_unit_numbers = get_target_unit_numbers(input_schedule_filename, username)
        # print(f"\n{category} カテゴリの処理中:")
        # print("対象ユニット番号:", target_unit_numbers)

        process_csv(category, input_ori_filename, output_filename, target_unit_numbers)

    for idx, row in categories.iterrows():
        category = row["category"]
        path = row["path"]
        process_category(category, path)

    events_data = event_close_read_csv(f"userdata/{username}.csv")

    closest_event, days_until_event = find_closest_event(events_data)

    if closest_event:
        print(f"最も近いイベント '{closest_event['Title']}' までの残り日数: {days_until_event}日")

    start_date = datetime.today().date()
    exam_days = days_until_event
    study_hours_file = "study.csv"

    generate_study_schedule(categories, start_date, exam_days, study_hours_file)

    # pathを使ってCSVファイルを取り込む
    schedule_csv_files = [f'userdata/{username}/{path}_ratio.csv' for path in categories['path']]
    df_combined = read_and_concat_csv_files(schedule_csv_files)

    # HTMLを生成
    generate_html(df_combined)

    # レンダリングするHTMLを指定
    return render_template(f'{username}/{username}.html')
```

App.py グラフページ

```
#########################################################################################################
#############################グラフページ#################################################
#########################################################################################################
@app.route('/graph_page')
def graph_page():
    username = session.get('username')

    for index, row in categories.iterrows():
        path = row['path']
        category = row['category']

        # ユーザーのcsvファイルが存在するか確認
        template_path = f"userdata/{username}/{path}_float.csv"
        if not os.path.exists(template_path):
            abort(404, description='先にスケジュールを作成してください')



        df_rikaim = pd.read_csv(f"subject/{path}.csv")
        df_graph = pd.read_csv(f"subject/ori/{path}.csv")
        df_check = pd.read_csv(f"subject/{path}_check.csv")
        df_float = pd.read_csv(f"userdata/{username}/{path}_float.csv")

        header_units =  df_graph['単元番号'].tolist()
        user_data = df_rikaim[df_rikaim["username"] == {username}].iloc[:, 1:].values.flatten().tolist()
        print(df_check)
        # usernameが存在するか確認
        if username in df_check["username"].values:
            group1_nodes = df_check[df_check["username"] == username].iloc[0, 1:].astype(str).tolist()
            for i in range(len(group1_nodes)):
                if group1_nodes[i] == 1:
                    group1_nodes[i] = df_check.columns[i + 1]
        #1を無理やり単元番号に変更するだけのバカみたいなシステム

        elif int(username) in df_check["username"].values:
            #ユーザー名がintがたになった時の処理(こっちメイン)
            int_username = int(username)
            user_row = df_check[df_check["username"] == int_username].iloc[0, 1:]
            group1_nodes = user_row[1:].astype(str).tolist()
            group1_nodes = [int(float(x)) for x in group1_nodes]
            # print(group1_nodes)

            for i in range(len(group1_nodes)):
                if group1_nodes[i] == 1:
                    group1_nodes[i] = df_check.columns[i + 1]

        else:
            # エラーメッセージを表示して処理を中断
            abort(404, description="理解度を入力してください")



        group2_nodes = df_float[df_float["CRITICAL"] == "YES"]["CODE"].astype(str).tolist()
        # print(group2_nodes)
```

```
        # '単元番号'列の値がheader_unitsに存在しない場合は-1で埋める
        missing_units = set(header_units) - set(df_rikaim.columns[1:])
        user_data += [-1] * len(missing_units)

        user_scores = {}
        for index, row in df_rikaim.iterrows():
            user_number = row.get("username", -1)
            user_scores[user_number] = {str(i): row.get(str(i), -1) for i in header_units}
        # print(user_scores)

        nodes_data = df_graph["単元名"].tolist()
        node_numbers = df_graph["単元番号"].tolist()
        conditions = df_graph["前単元"].tolist()

        G = nx.DiGraph()
        node_names = {node_numbers[i]: f"{node_numbers[i]} - {nodes_data[i]}" for i in range(len(node_numbers))}
        color_mapping = {
            0: 'crimson',
            1: 'orange',
            2: 'khaki',
            3: 'palegreen',
            4: 'forestgreen',
            -1: 'deepskyblue',
        }

        for i in range(len(nodes_data)):
            condition = conditions[i]

            if pd.isna(condition) or condition == 99999:
                continue
            else:
                if isinstance(condition, float):
                    condition = int(condition)

                dependent_nodes = [str(condition)] if isinstance(condition, int) else [str(int(c)) for c in condition.split(',')]

                for dependent_node in dependent_nodes:
                    dependent_node = int(dependent_node)
                    if dependent_node in node_numbers:
                        G.add_edge(node_names[dependent_node], node_names[node_numbers[i]])

        net = Network(notebook=True)
```

ノード色分け

App.py グラフページ

ノードの形変化

```
for node in G.nodes:
    node_number = int(node.split(" - ")[0])
    if node_number in node_numbers:
        index = node_numbers.index(node_number)
        if user_number in user_scores and str(node_number) in user_scores[user_number]:
            color = color_mapping.get(user_scores[user_number][str(node_number)])
        link = f'/Kyouzai/{category}/{node_number}'

        if str(node_number) in group2_nodes:
            net.add_node(node, color=color, originalColor=color, title=f'<a href="{link}" target="_blank">{node.split(" - ")[1]}</a>', shape='star', size=40)
        elif str(node_number) in group1_nodes:
            net.add_node(node, color=color, originalColor=color, title=f'<a href="{link}" target="_blank">{node.split(" - ")[1]}</a>', size=35)
        else:
            net.add_node(node, color=color, originalColor=color, title=f'<a href="{link}" target="_blank">{node.split(" - ")[1]}</a>', size=20)

for edge in G.edges:
    net.add_edge(edge[0], edge[1], arrows='to')

# ディレクトリのパス
directory_path = f'static/subjects/{username}/'

# ディレクトリが存在しない場合に作成
if not os.path.exists(directory_path):
    os.makedirs(directory_path)

# その後、net.show()を呼び出す
net.show(f'static/subjects/{username}/{path}.html')
```

検索機能埋め込み

```
# その後、net.show()を呼び出す
net.show(f'static/subjects/{username}/{path}.html')

search_script = """
<script>
function searchNodes() {
    var searchText = document.getElementById('searchBox').value;
    var nodes = network.body.data.nodes.get();
    var foundNodes = nodes.filter(function(node) {
        return node.label.includes(searchText);
    });
    if (foundNodes.length > 0) {
        var foundNodeIds = foundNodes.map(function(node) {
            return node.id;
        });
        network.selectNodes(foundNodeIds);
        network.focus(foundNodeIds[0], {animation: true});
        network.body.data.nodes.update(foundNodeIds.map(function(nodeId) {
            return {id: nodeId, color: 'gray'};
        }));
    } else {
        alert('ノードが見つかりませんでした。');
    }
}

function resetSearch() {
    document.getElementById('searchBox').value = '';
    network.unselectAll();
    network.body.data.nodes.update(nodes.map(function(node) {
        return {id: node.id, color: node.originalColor};
    }));
}
</script>

<div style="position: absolute; top: 10px; right: 10px;">
    <input type="text" id="searchBox" placeholder="単元を検索">
    <button onclick="searchNodes()">検索</button>
    <button onclick="resetSearch()">リセット</button>
</div>
"""



with open(f'static/subjects/{username}/{path}.html', 'a', encoding='utf-8') as html_file:
    html_file.write(search_script)

return render_template('graph.html', username=username)
```

App.py テスト範囲選択

```
##########################################################################################################
##################################テスト範囲を対象にして対象単元を指定#####################################
##########################################################################################################

def get_target_unit_numbers(input_filename, user_number):
    target_unit_numbers = set()

    with open(input_filename, 'r', encoding='utf-8') as input_file:
        reader = csv.reader(input_file)
        headers = next(reader)


        for row in reader:
            if row[0] == user_number:  # ユーザー番号の列に対応するインデックス
                for header, value in zip(headers[1:], row[1:]):
                    if value == '1':
                        target_unit_numbers.add(header)

    return target_unit_numbers
```

```
def process_csv(category, input_filename, output_filename, target_unit_numbers):
    with open(input_filename, 'r', encoding='utf-8') as input_file, \
        open(output_filename, 'w', encoding='utf-8', newline='') as output_file:

        # print(f"Input File: {input_filename}")
        # print(f"Output File: {output_filename}")
        # print(f"Target Unit Numbers: {target_unit_numbers}")

        reader = csv.DictReader(input_file)

        fieldnames = ['単元名', '単元番号', '指導時数', '前単元']
        writer = csv.DictWriter(output_file, fieldnames=fieldnames)
        writer.writeheader()
        previous_unit_numbers = set()
        code_contents = target_unit_numbers
        all_unit_numbers = set(code_contents)
        # print(all_unit_numbers)

        writer.writerow({
            '単元名': 'Start',
            '単元番号': '0',
            '指導時数': '0',
            '前単元': ''
        })

        for row in reader:
            unit_number = row['単元番号']

            # target_unit_numbersに含まれない単元番号の行は無視する
            if unit_number not in target_unit_numbers:
                continue

            unit_name = row['単元名']
            teaching_hours = row['指導時数']
            previous_units = row['前単元']

            # NaNの場合は '0' を挿入
            if previous_units.lower() == 'nan':
                previous_units = '0'
            else:
                # カンマで分割し、かつ存在しない単元番号に対応する前単元の要素も削除
                previous_units = ','.join(num for num in previous_units.split(',') if num in target_unit_numbers)

            # 前単元が一つも存在しない場合（NaNまたはすべてが含まれない場合）のみ '0' を挿入
            if not previous_units:
                previous_units = '0'

            # print(f"Used: {unit_number}: {unit_name}, Teaching Hours: {teaching_hours}, Previous Units: {previous_units}")

            writer.writerow({
                '単元名': unit_name,
                '単元番号': unit_number,
                '指導時数': teaching_hours,
                '前単元': previous_units
            })

            previous_unit_numbers.update(previous_units.split(','))

        valid_previous_units = previous_unit_numbers.intersection(target_unit_numbers)
        end_previous_units = ','.join(sorted(target_unit_numbers - valid_previous_units))

        # print(f"End Unit: Previous Units: {end_previous_units}")

        writer.writerow({
            '単元名': 'End',
            '単元番号': '999',
            '指導時数': '0',
            '前単元': end_previous_units
        })
```

App.py CPM計算

```
#####################################################
#########################################CPM計算(youtu
#####################################################



#print stars
def stars(number):
    for i in range(number):
        print("*", end = "")
    print("")

#error messages
def errorCodeMsg():
    print("Error in input file : CODE ")
    quit()


def errorPredMsg():
    print("Error in input file : PREDECESSORS ")
    quit()

def errorDaysMsg():
    print("Error in input file : DAYS ")
    quit()

# Scans if the code in predecessors and successors
# in the list of task codes:
def getTaskCode(mydata, code):
    x = 0
    flag = 0
    for i in mydata['CODE']:
        if(i == code):
            flag = 1
            break

        x+=1

    if(flag == 1):
        return x
    else:
        errorCodeMsg()
```

```
def forwardPass(mydata):
    ntask = mydata.shape[0]
    ES = np.zeros(ntask, dtype=np.int32)
    EF = np.zeros(ntask, dtype=np.int32)
    temp = []  # hold temporary codes

    for i in range(ntask):
        if mydata['PREDECESSORS'][i] is None:
            ES[i] = 0
            try:
                EF[i] = ES[i] + mydata['DAYS'][i]
            except Exception as e:
                print(f"Error in task {i+1}: {e}")
                errorDaysMsg()

        else:
            for j in mydata['PREDECESSORS'][i]:
                index = getTaskCode(mydata, j)
                # Check the video for the missing line here

                if index == i:
                    print(f"Error in task {i+1}: {errorPredMsg()}")
                else:
                    temp.append(EF[index])

            ES[i] = max(temp)
            try:
                EF[i] = ES[i] + mydata['DAYS'][i]
            except Exception as e:
                print(f"Error in task {i+1}: {e}")
                errorDaysMsg()

            # reset temp
            temp = []

    # Update dataFrame:
    mydata['ES'] = ES
    mydata['EF'] = EF

    return mydata
```

```
def backwardPass(mydata):
    ntask = mydata.shape[0]
    temp = []
    LS = np.zeros(ntask, dtype=np.int32)
    LF = np.zeros(ntask, dtype=np.int32)
    SUCCESSORS = np.empty(ntask, dtype = object)

    #create successor column:

    # create successor column:
    for i in range(ntask-1, -1, -1):
        if(mydata['PREDECESSORS'][i] != None):
            for j in mydata['PREDECESSORS'][i]:
                index = getTaskCode(mydata, j)
                if(SUCCESSORS[index] != None):
                    SUCCESSORS[index].append(mydata['CODE'][i])
                else:
                    SUCCESSORS[index] = [mydata['CODE'][i]]

    # incorporate the column to the data frame:
    mydata["SUCCESSORS"] = SUCCESSORS



    #compute for  EF and LS:

    for i in range(ntask-1, -1, -1):
        if(mydata['SUCCESSORS'][i] == None):
            LF[i] = np.max(mydata['EF'])
            LS[i] = LF[i] - mydata['DAYS'][i]
        else:
            for j in mydata['SUCCESSORS'][i]:
                index = getTaskCode(mydata,j)
                temp.append(LS[index])

            LF[i] = min(temp)
            LS[i] = LF[i] - mydata['DAYS'][i]

            #reset temp list:
            temp = []


    #incorporate LF and LS to data frame :

    mydata['LS'] = LS
    mydata['LF'] = LF

    return mydata
```

```
#compute for SLACK and CRITICAL state
def slack(mydata):
    ntask = mydata.shape[0]

    SLACK = np.zeros(shape = ntask, dtype=np.int32)
    CRITICAL = np.empty(shape = ntask,dtype = object)

    for i in range(ntask):
        SLACK[i] = mydata['LS'][i] - mydata['ES'][i]
        if(SLACK[i] == 0):
            CRITICAL[i] = "YES"
        else:
            CRITICAL[i] = "NO"

    #incorporate SLACK and CRITICAL to data frame

    mydata['SLACK'] = SLACK
    mydata['CRITICAL'] = CRITICAL


    #re arrange columns in dataframe:
    mydata = mydata.reindex(columns = ['DESCR', 'CODE','PREDECESSORS',
            'SUCCESSORS','DAYS','ES','EF','LS','LF','SLACK','CRITICAL'])

    return mydata
#wrapper function:

def computeCPM(mydata):

    mydata = forwardPass(mydata)
    mydata = backwardPass(mydata)
    mydata = slack(mydata)
    return mydata
```

```
def process_category(category, path):
    # ori.csv からデータを読み込む
    username = session.get('username')
    df_ori = pd.read_csv(f'userdata/{username}/{path}_sten.csv', delimiter=',')

    # データの整形
    data = {
        'DESCR': df_ori['単元名'],
        'CODE': df_ori['単元番号'].astype(str),  # 数値から文字列に変換
        'PREDECESSORS': df_ori['前単元'].apply(lambda x: None if pd.isna(x) else [str(int(float(i))) for i in str(x).split(',')]),
        'DAYS': df_ori['指導時数']
    }

    # データフレームを作成
    df = pd.DataFrame(data)

    # CPM計算を実行
    result_data = computeCPM(df)

    # クリティカルなノードのみを取得
    critical_nodes = result_data[result_data['CRITICAL'] == 'YES']

    # 結果を格子枠線のある表で表示
    # print(f"\n{category} - CPM計算結果:")
    # print(result_data.to_markdown(index=False, tablefmt='grid'))

    # クリティカルパス上の経路を表示
    # print(f"\n{category} - Critical Path Route:")
    for index, row in critical_nodes.iterrows():
        print(f"({row['CODE']}) -> ", end="")

    # 結果をCSVファイルに保存
    result_data.to_csv(f'userdata/{username}/{path}_float.csv', index=False)
    # print(f"\n{category} - 結果を {username}_{path}.csv に保存しました。")
```

ここのみ変更

App.py

直近の予定までの日数を計算

```
##########################################################################################################
#######################################納期計算
##########################################################################################################



def event_close_read_csv(file_path):
    data = []
    with open(file_path, newline='', encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            data.append(row)
    return data

def find_closest_event(events_data):
    closest_event = None
    closest_days_until_event = float('inf')

    for event in events_data:
        start_date = event['Start']
        days_until_event = days_until_next_event(start_date)

        if days_until_event < closest_days_until_event:
            closest_days_until_event = days_until_event
            closest_event = event

    return closest_event, closest_days_until_event

def days_until_next_event(start_date):
    today = datetime.today().date()
    start_date = datetime.strptime(start_date, '%Y-%m-%d').date()

    days_until_event = (start_date - today).days

    if days_until_event < 0:
        return float('inf')  # イベントは既に過去の日付です

    return days_until_event
```

App.py スケジュール作成

```
#########################################################################################
#########################################スケジュール作成関数#############################################
#########################################################################################



def generate_study_schedule(categories, start_date, exam_days, study_hours_file):
    study_hours_df = pd.read_csv(study_hours_file, index_col=0)
    username = session.get('username')


    # 圧縮率とpathをunit_press_fileにヘッダーとして書き込む
    unit_press_file = f"userdata/{username}/{username}_press.csv"
    with open(unit_press_file, 'w', encoding='utf-8') as file:
        file.write("科目名,圧縮率\n") # ヘッダーを追加

    for idx, row in categories.iterrows():
        category = row["category"]
        path = row["path"]

        unit_ratios_file = f"userdata/{username}/{path}_float.csv"
        selected_columns = ["CODE", "DAYS", "LS", "DESCR"]
        unit_ratios_df = pd.read_csv(unit_ratios_file, usecols=selected_columns)
        unit_ratios_df = unit_ratios_df.sort_values(by="LS")
        unit_ratios_df = unit_ratios_df.reset_index(drop=True)

        total_study_time = 0
        study_schedule = []

        for day in range(exam_days):
            current_date = start_date + timedelta(days=day)
            day_of_week = current_date.strftime("%a")

            try:
                study_hours = study_hours_df.loc[username, day_of_week]
            except KeyError:
                try:
                    # "username" 列に存在しない場合、整数列で同じ値を探して代用
                    index_for_integer = study_hours_df.index.get_loc(int(username))
                    study_hours = study_hours_df.iloc[index_for_integer][day_of_week]
                except KeyError:
                    # エラーメッセージを表示して処理を中断
                    print(f"エラー: {username} のスタディ時間が設定されていません。")
                    abort(404, description="勉強時間を設定してください")
            total_study_time += study_hours

            study_schedule.append({
                "Day": day + 1,
                "Date": current_date.strftime("%Y-%m-%d"),
                "Day of Week": day_of_week,
                "Study Hours": study_hours,
                "Total Study Time": total_study_time,
            })

        unit_ratios_dict = dict(zip(unit_ratios_df["CODE"], unit_ratios_df["DAYS"]))
        unit_study_time = {unit: total_study_time * int(ratio) / sum(map(int, unit_ratios_dict.values())) for unit, ratio in unit_ratios_dict.items()}
```

```
        study_schedule = []

        for day in range(exam_days):
            current_date = start_date + timedelta(days=day)
            day_of_week = current_date.strftime("%a")
            try:
                study_hours = study_hours_df.loc[username, day_of_week]

            except KeyError:
                # "username" 列に存在しない場合、整数列で同じ値を探して代用
                index_for_integer = study_hours_df.index.get_loc(int(username))
                study_hours = study_hours_df.iloc[index_for_integer][day_of_week]

            unit_to_study = list(unit_ratios_df["CODE"])
            unit_to_study.sort(key=lambda x: unit_ratios_df[unit_ratios_df["CODE"] == x]["LS"].values[0])

            for unit in unit_to_study:
                if unit not in unit_study_time:
                    continue

                if study_hours > unit_study_time[unit]:
                    study_schedule.append({
                        "Day": day + 1,
                        "Date": current_date.strftime("%Y-%m-%d"),
                        "Day of Week": day_of_week,
                        "Unit": unit,
                        "Study Hours": unit_study_time[unit],
                        "Description": unit_ratios_df[unit_ratios_df["CODE"] == unit]["DESCR"].values[0],
                    })

                    study_hours -= unit_study_time[unit]
                    del unit_study_time[unit]
                else:
                    study_schedule.append({
                        "Day": day + 1,
                        "Date": current_date.strftime("%Y-%m-%d"),
                        "Day of Week": day_of_week,
                        "Unit": unit,
                        "Study Hours": study_hours,
                        "Description": unit_ratios_df[unit_ratios_df["CODE"] == unit]["DESCR"].values[0],
                    })

                    unit_study_time[unit] -= study_hours

                    break

        csv_output_file = f"userdata/{username}/{path}_ratio.csv"
        study_schedule_df = pd.DataFrame(study_schedule)
        total_ratio = unit_ratios_df["DAYS"].sum()

        total_ratio = unit_ratios_df["DAYS"].sum()

        total_ratio = unit_ratios_df["DAYS"].sum()
        unit_total_time = 0

        unit_total_time = {unit: 0 for unit in unit_ratios_df["CODE"]}
        for schedule in study_schedule:
            unit_total_time[schedule["Unit"]] += schedule["Study Hours"]

        if total_ratio == 0:
            # If the total ratio is 0, set all Study Hours to 0
            study_schedule_df["Study Hours"] = 0
        else:
            # Calculate compression ratio and perform further calculations
            total_study_time = sum(unit_total_time.values())
            compression_ratio = total_ratio / total_study_time
            # Additional calculations as needed...


        study_schedule_df.to_csv(csv_output_file, index=False)
```

```
        study_schedule_df.to_csv(csv_output_file, index=False)

        # print(f"\nCSVファイル {csv_output_file} に書き込まれた内容:")
        # print(study_schedule_df)


        total_ratio = unit_ratios_df["DAYS"].sum()
        # print(sum(unit_total_time.values()))
        total_study_time = sum(unit_total_time.values())
        compression_ratio = total_ratio / total_study_time

        # print(f"\n{category} - 各単元ごとの合計時間:")
        # for unit, total_time in unit_total_time.items():
        #     print(f"{unit}: {total_time} 時間")

        # print(f"\n{category} - 理想勉強時間: {total_ratio}")
        # print(f"\n{category} - 合計勉強時間: {total_study_time} 時間")
        # print(f"\n{category} - 圧縮率: {compression_ratio}")

        # 圧縮率とpathをunit_press_fileに追記
        with open(unit_press_file, 'a', encoding='utf-8') as file:
            file.write(f"{category},{compression_ratio}\n")
```

App.py スケジュールをhtmlに書き込み

```
##########################################################################################################
#####################################html作成関数
##########################################################################################################

def read_and_concat_csv_files(csv_files):
    dfs = []
    for file in csv_files:
        df = pd.read_csv(file)
        df = df[df['Study Hours'] != 0]
        # print(df)
        df['Source'] = file
        dfs.append(df)

    combined_df = pd.concat(dfs, ignore_index=True)
    return combined_df

def generate_html(schedule_data):
    username = session.get('username')
    html_content = f"""
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
</script>
"""

    press_file = f"userdata/{username}/{username}_press.csv"

    # CSVファイルをDataFrameに読み込む
    if os.path.exists(press_file):
        press_df = pd.read_csv(press_file)

        # path列が0の行を除外
        press_df = press_df[press_df['圧縮率'] != 0]
    else:
        press_df = pd.DataFrame()


    press_html = press_df.to_html(classes='table table-bordered table-striped',index=False)+"※圧縮率が高いほど無理のある勉強スケジュールになっている"




    directory_path = f"templates/{username}"
    if not os.path.exists(directory_path):
        os.makedirs(directory_path)

    with open(f"templates/{username}/{username}.html", "w", encoding="utf-8") as file:
        file.write(html_content+html_content2+press_html)
```

```
def generate_schedule_data(schedule_data):
    # 与えられたデータを日付とソースでグループ化し、各グループにformat_schedule関数を適用
    grouped_data = schedule_data.groupby(['Date', 'Source']).apply(format_schedule)

    # HTML形式の文字列を初期化
    schedule_html = ""

    # グループデータが存在しない場合、"max baka"というエラーメッセージを含むHTML文字列を生成
    if grouped_data.empty:
        return '<div>Error: max baka</div>\n'
    #エラー回避用バカ

    # 各グループごとにHTMLタグを生成
    for (date, source), schedule in grouped_data.items():
        # 各データをHTMLの属性として追加
        schedule_html += f'<div data-date="{date}" data-schedule="{schedule}" data-source="{source}"></div>\n'

    # 生成したHTML文字列を返す
    return schedule_html


def format_schedule(group):
    formatted_schedule = ', '.join([f'{unit}, {description}' for _, (unit, description) in group[['Unit', 'Description']].iterrows()])
    return formatted_schedule
```

App.py 理解度ページ

```python
# subjectフォルダが存在しない場合は作成
subject_folder = "subject"
if not os.path.exists(subject_folder):
    os.makedirs(subject_folder)

@app.route('/submit', methods=['POST'])
def submit():
    if request.method == 'POST':
        username = session['username']
        # フォームデータを取得
        data = request.form
        check_data = {key: data[key] for key in data if '_checkbox' in key}
        # ユーザーごとにsubjectフォルダ内のCSVにデータを保存する例
        save_data_to_subject_csv(username, data)

        # チェックボックスのデータを保存
        checkbox_save(username, check_data)

        # 成功したらどこかにリダイレクトするか、適切なレスポンスを返す
        return render_template('rikai.html')
```

理解度

```python
def save_data_to_subject_csv(user_id, data):
    # カテゴリごとの科目

    # 各科目ごとのCSVファイルパス
    subject_csv_paths = {row["category"]: f"{subject_folder}/{row['path']}.csv" for _, row in categories.iterrows(

    # 各科目ごとにデータを整形
    for _, row in categories.iterrows():
        category = row["category"]
        subjects = row["subjects"]
        units = row["units"]

        formatted_data = {"username": user_id}
        for unit in units:
            # カテゴリに属する科目のみを処理
            key = f"{category.lower()}_{unit.lower()}"

            if key in data:
                formatted_data[unit.lower()] = data[key]
                # print(key)



        csv_path = subject_csv_paths[category]

        # ヘッダーをunitsに変更して新しい行を作成
        new_row = pd.DataFrame([formatted_data], columns=["username"] + units)

        print(csv_path)
        print(new_row)

        # 新しい行に NaN が含まれているか確認
        if new_row.isnull().values.any():
            print("ほかの学年の科目です。")
        else:
            # NaN が含まれていない場合の処理を続行

            # 既存のCSVデータを読み込む
            existing_data = pd.DataFrame(columns=["username"] )
            if os.path.exists(csv_path) and os.path.getsize(csv_path) > 0:
                existing_data = pd.read_csv(csv_path, encoding='utf-8-sig')

            # print("全データ")
            # print(existing_data)
            # 同じ"username"を持つ行を削除
            existing_data = existing_data[existing_data['username'].astype(str) != formatted_data['username']]
            #print("かぶってないデータ")
            #print(existing_data)
            # 既存のデータを更新または新規追加
            existing_data = pd.concat([existing_data,new_row], ignore_index=True)

            # CSVファイルに書き込む
            existing_data.to_csv(csv_path, index=False, encoding='utf-8')
            # print(existing_data)
```

学習範囲

```python
def checkbox_save(username, check_data):
    for index, row in categories.iterrows():
        category = row['category']
        subjects = row['subjects']
        path = row['path']
        units = row['units']

        csv_filename = f"{subject_folder}/{path}_check.csv"

        # print(check_data)

        # print(check_data)


        # 新しい行の作成
        new_row = [username] + [0 if check_data.get(f"{category}_{units}_checkbox") is None else 1 for units in units]
        # print(new_row)

        # 既存のCSVデータを読み込む
        existing_data = pd.DataFrame(columns=["username"] + units)
        if os.path.exists(csv_filename) and os.path.getsize(csv_filename) > 0:
            existing_data = pd.read_csv(csv_filename, encoding='utf-8-sig')

        #print("全データ")
        #print(existing_data)

        # 同じ"username"を持つ行を削除
        existing_data = existing_data[existing_data['username'].astype(str) != username]
        #print("かぶってないデータ")
        #print(existing_data)

        # 既存のデータを更新または新規追加
        existing_data = pd.concat([existing_data, pd.DataFrame([new_row], columns=["username"] + units)], ignore_index=True)

        # CSVファイルに書き込む
        existing_data.to_csv(csv_filename, index=False, encoding='utf-8-sig')
```

App.py

清水さんのプログラムのコピペ&微調整
よくわからない

```
##########################################################################################
#####################################教材ページ
##########################################################################################
```

ログアウト

```
##########################################################################################
#####################################共用
##########################################################################################
@app.route('/logout')
def logout():
    # セッションからユーザー情報を削除し、ログアウト
    session.pop('username', None)
    session.pop('password', None)
    session.pop('name', None)
    return redirect(url_for('home'))

if __name__ == '__main__':
    app.run(debug=True)
```

頻出エラー

Study.csvがおかしくなる

→必要な部分だけ残して消す

（セッションの関係でエラー？）

IDの頭に0がついているとバグる

→新規登録の時点ではじいておけると思う

（システムを研究に使うときはやっておいて）

未評価の教材が3を下回ると教材ページが表示されなくなる

→解決方法がわからない

評価を行わず評価ボタンを押すとエラる

→はじけると思う

レビューを書くとエラーになる

→Mecabが入っていないまたはパスが通ってない可能性がある

→win64のMecabをインストールして