A Recommendation System for selection of Minors in LPU

End Term Report

By

Group 16

Yantrapati Nikhil

Kummari Jaya Ram

Akshat Sharma

Manish Kumar Singh

Section: K18VQ

Roll No's: 61,62,63,64

Department of Intelligent Systems

School of Computer Science Engineering

Lovely Professional University, Jalandhar

Student Declaration

This is to declare that this report has been written by us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. We aver that if any part of the report is found to be copied, we are shall take full responsibility for it.

Yantrapati Nikhil

61

Kummari Jaya Ram

62

Akshat Sharma

63

Manish Kumar Singh

64

Place: Lovely Professional University Jalandhar

Date: 08/04/2020

Introduction

ARTIFICIAL INTELLIGENCE

It is the science & engineering of making intelligent machines, especially intelligent computer programs.It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to method which are biologically observable.

In computer science artificial intelligence(AI) sometimes called machine intelligence is intelligence demonstrated by machines in contrast to the natural intelligence displayed by humans and animals. Leading AI textbooks define the field as the study of intelligence agents any device that perceives its environment and takes actions that maximizes its chances of successfully achieving its goals. Colloquially the term artificial intelligence is often used to describe machines or computers that mimic cognitive functions that humans associate with the human mind such as learning and problem solving.

ABOUT MY PROJECT

Our project namely "A Recommendation system for selection of minors in LPU" A Minor is a group of 4 courses that shall be offered to the students in an unrelated area to develop additional skills enhancing employment opportunities. This is a web based recommender System ,this project involves two parts the first part deals with the selection of area of specialization for a student, this is done by asking series of questions and then a ranking system is used to generate an area of specializtion from the persons score. Our code is basically divided into nine sections below is our code & the explaination about our code will be there after the code.

Code:

```
# STUDENT'S INFORMATION FROM DATABASE



import sqlite3
connection = sqlite3.connect('Credentials.db')
cursor = connection.cursor()

create_table = "CREATE TABLE IF NOT EXISTS users_credential (id INTEGER PRIMARY KEY," \
            "reg_number text," \
            "surname text," \
            "middle_name text," \
            "first_name text," \
            "email text,"\
            "date_registered text," \
            "profile_picture text," \

cursor.execute(create_table)

connection.commit()
connection.close()



#GETTING STUDENTS MARKS AND PREDICTING THEIR CHANCES



import numpy as np
```

```
from models.Prediction.Training import DecisionTreeClassifier
returned_grades = list()
    student_mark = []
    model_year = DecisionTreeClassifier()
    new_input_value = [scores]
    new_input = np.array(new_input_value)
    new_input = new_input.reshape(-1, 1)
    model_year.load_dataset()
    model_year.encode_variables_for_y(model_year.Y)
    model_year.spliting_to_training_and_test_set_no_return(model_year.X, model_year.Y)
    scaled_input = model_year.feature_scaling(new_input)
    new_prediction = saved_model.predict(scaled_input)
    if new_prediction == 0:
        new_prediction = '(0, 5]'
    elif new_prediction == 1:
        new_prediction = '(10, 15]'
    elif new_prediction == 2:
        new_prediction = '(15, 20]'
    elif new_prediction == 3:
        new_prediction = '(20, 25]'
    elif new_prediction == 4:
        new_prediction = '(25, 30]'
    elif new_prediction == 5:
        new_prediction = '(30, 35]'
    elif new_prediction == 6:
        new_prediction = '(35, 40]'
    elif new_prediction == 7:
        new_prediction = '(40, 45]'
    elif new_prediction == 8:
        new_prediction = '(45, 50]'
```

```
    elif new_prediction == 9:
        new_prediction = '(5, 10]'
    elif new_prediction == 10:
        new_prediction = '(50, 55]'
    elif new_prediction == 11:
        new_prediction = '(55, 60]'
    elif new_prediction == 12:
        new_prediction = '(60, 65]'
    elif new_prediction == 13:
        new_prediction = '(65, 70]'
    elif new_prediction == 14:
        new_prediction = '(70, 75]'
    elif new_prediction == 15:
        new_prediction = '(75, 80]'
    elif new_prediction == 16:
        new_prediction = '(80, 85]'
    elif new_prediction == 17:
        new_prediction = '(85, 90]'
    elif new_prediction == 18:
        new_prediction = '(90, 95]'
    elif new_prediction == 19:
        new_prediction = '(95, 100]'

    student_mark.append(new_input_value[0])
    student_mark.append(new_prediction)
    returned_grades.append(student_mark)
    return returned_grades


#CLUSTERING
```

```
import pandas as pd
import matplotlib
matplotlib.use('Qt4Agg')
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv('dataset/clustering/system_eng_cluster.csv')
print(dataset.describe())
print(dataset.get_values())


#K MEAN CLUSTERING


import matplotlib
import pandas as pd

matplotlib.use('Qt4Agg')
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans


class Clustering(object):
    def __init__(self, csv, ymeans_1=None, ymeans_2=None):
        self.csv = csv
        self.ymeans_1 = ymeans_1
```

```
    self.ymeans_2 = ymeans_2

    # importing the dataset with pandas
    # 'dataset/clustering/system_eng_cluster.csv'
    self.dataset_loader = pd.read_csv(self.csv)
    self.X1 = self.dataset_loader.iloc[:, [2, 4]].values
    self.X2 = self.dataset_loader.iloc[:, [3, 4]].values

@staticmethod
def process_wcss(x_column_for_wcss):
    wcss_to_process = []
    for i in range(1, 11):
        kmeans_1 = KMeans(n_clusters=i, init='k-means++', max_iter=300,
                          n_init=10, random_state=0)
        kmeans_1.fit(x_column_for_wcss)
        wcss_to_process.append(kmeans_1.inertia_)

    return wcss_to_process

@staticmethod
def plot_wcss(wcss_list, course_title):
    plt.plot(range(1, 11), wcss_list)
    plt.title("The Elbow Method For Test")
    plt.xlabel("Number of clusters")
    plt.ylabel("wcss for {}".format(course_title))
    plt.show()
    plt.imsave()

def predict_data(self):
    # applying k-means to the mall dataset
```

```
    kmeans_predict = KMeans(n_clusters=6, init='k-means++', max_iter=300,
                            n_init=10, random_state=0)
    self.ymeans_1 = kmeans_predict.fit_predict(self.X1)
    self.ymeans_2 = kmeans_predict.fit_predict(self.X2)
    return self.ymeans_1, self.ymeans_2

@staticmethod
def visualise_clusters(x_column_to_visualize, y_column_to_visualise, test_title):
    kmeans_clusters = KMeans(n_clusters=6, init='k-means++', max_iter=300,
                             n_init=10, random_state=0)
    kmeans_clusters.fit(x_column_to_visualize)
    # Visualizing the clusters
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 0, 0],
                x_column_to_visualize[y_column_to_visualise == 0, 1],
                s=10, c='red', label='Cluster 1')
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 1, 0],
                x_column_to_visualize[y_column_to_visualise == 1, 1],
                s=10, c='blue', label='Cluster 2')
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 2, 0],
                x_column_to_visualize[y_column_to_visualise == 2, 1],
                s=10, c='green', label='Cluster 3')
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 3, 0],
                x_column_to_visualize[y_column_to_visualise == 3, 1],
                s=10, c='cyan', label='Cluster 4')
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 4, 0],
                x_column_to_visualize[y_column_to_visualise == 4, 1],
                s=10, c='magenta', label='Cluster 5')
    plt.scatter(x_column_to_visualize[y_column_to_visualise == 5, 0],
                x_column_to_visualize[y_column_to_visualise == 5, 1],
                s=10, c='black', label='Cluster 6')
```

```
    plt.scatter(kmeans_clusters.cluster_centers_[:, 0], kmeans_clusters.cluster_centers_[:, 1],
                s=50, c='yellow', label='Centroids')
    plt.title("Clusters OF Students Performance Based On Test Score")
    plt.xlabel("{} SCORE".format(test_title))
    plt.ylabel("Test score")
    plt.legend()
    plt.show()



#QUESTION AND ANSWER SESSIONS TO SEE THE INTEREST OF THE STUDENT



import random

import pandas as pd
from models.aos_questions_and_answer.processedlistofdictionaries import Util

# Initializing variables
ai_correct = 0
ai_failed = 0
se_correct = 0
se_failed = 0
cn_correct = 0
cn_failed = 0
sye_correct = 0
sye_failed = 0
tc_correct = 0
```

```
tc_failed = 0

AI = []
SE = []
CN = []
SYE = []
TC = []
final_scores = []

current_question_number = 0
total_questions = 0

# Reading the CSV file that contains all compiled questions with respective answers
dataset = pd.read_csv('models/aos_questions_and_answer/dataset/core_courses.csv')

# AI Data processing
ai_questions = dataset.iloc[:, :1].values
ai_answers = dataset.iloc[:, 1].values
ai_list_of_dictionaries_of_questions_and_answers = Util.processed_list_dict(ai_questions, ai_answers)
ai_selected_six_random = Util.select_six_random(ai_list_of_dictionaries_of_questions_and_answers)

# Software Engineering Data processing
software_engineering_questions = dataset.iloc[:, 2:3].values
software_engineering_answers = dataset.iloc[:, 3].values
software_engineering_list_of_dictionaries_of_questions_and_answers = \
    Util.processed_list_dict(software_engineering_questions, software_engineering_answers)
se_selected_six_random                                                                    =
Util.select_six_random(software_engineering_list_of_dictionaries_of_questions_and_answers)

# Computer Networks Data processing
```

```
computer_networks_questions = dataset.iloc[:, 4:5].values
computer_networks_answers = dataset.iloc[:, 5].values
computer_networks_list_of_dictionaries_of_questions_and_answers =\
    Util.processed_list_dict(computer_networks_questions, computer_networks_answers)
cn_selected_six_random = Util.select_six_random(computer_networks_list_of_dictionaries_of_questions_and_answers)

# Systems Engineering Data processing
systems_engineering_questions = dataset.iloc[:, 6:7].values
systems_engineering_answers = dataset.iloc[:, 7].values
systems_engineering_list_of_dictionaries_of_questions_and_answers = \
    Util.processed_list_dict(systems_engineering_questions, systems_engineering_answers)
sye_selected_six_random = Util.select_six_random(systems_engineering_list_of_dictionaries_of_questions_and_answers)

# Theoretical Computing Data processing
theoretical_computing_questions = dataset.iloc[:, 8:9].values
theoretical_computing_answers = dataset.iloc[:, 9].values
theoretical_computing_list_of_dictionaries_of_questions_and_answers = \
    Util.processed_list_dict(theoretical_computing_questions, theoretical_computing_answers)
tc_selected_six_random = Util.select_six_random(theoretical_computing_list_of_dictionaries_of_questions_and_answers)

# Getting total questions and answers to be asked for ever user
total_questions_and_answer = Util.all_selected_questions_with_answers(ai_selected_six_random,
                                                                      se_selected_six_random,
                                                                      cn_selected_six_random,
                                                                      sye_selected_six_random,
                                                                      tc_selected_six_random)

# print(total_questions_and_answer)
```

```
for i in total_questions_and_answer.values():
    for j in i:
        total_questions += 1



#APPLICATION FORMS



from flask_wtf import FlaskForm
from flask_wtf.file import FileField, FileAllowed
from flask_login import current_user
from models.users.users import User
from wtforms import StringField, PasswordField, SubmitField, BooleanField, RadioField, SelectField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError


class AdminAddUserForm(FlaskForm):
    registration_number = StringField('Registration Number',
                                      validators=[DataRequired(),
                                                  Length(min=1, max=20)])
    surname = StringField('Surname',
                          validators=[DataRequired(),
                                      Length(min=1, max=20)])
    middle_name = StringField('Middle Name',
                              validators=[DataRequired(),
                                          Length(min=1, max=20)])
    first_name = StringField('First Name',
                             validators=[DataRequired(),
```

```
                                Length(min=1, max=20)])
    email = StringField('Email',
                        validators=[DataRequired(),
                                    Email()])
    password = PasswordField('Password',
                             validators=[DataRequired(),
                                         Length(min=2)])
    submit = SubmitField('Register')

    def validate_email(self, email):
        _, all_emails_from_database = User.find_all_emails_and_registration_number()
        if email.data:
            if email.data in all_emails_from_database:
                raise ValidationError("That email is taken. Please choose another one!")
        else:
            raise ValidationError("This field cannot be blank!")

    def validate_registration_number(self, registration_number):
        all_registration_number_from_database, _ = User.find_all_emails_and_registration_number()
        if registration_number.data:
            if registration_number.data in all_registration_number_from_database:
                raise ValidationError("That Registration Number is taken. Please choose another one!")


class UserLoginForm(FlaskForm):
    registration_number = StringField('Registration Number/Username',
                                      validators=[DataRequired(),
                                                  Length(min=1)])
    password = PasswordField('Password',
                             validators=[DataRequired(),
```

```
                                Length(min=2)])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Log In')


class UpdateAccountForm(FlaskForm):
    registration_number = StringField('Registration Number',
                                validators=[DataRequired(),
                                        Length(min=1)])
    surname = StringField('Surname',
                        validators=[DataRequired(),
                                Length(min=1, max=20)])
    middle_name = StringField('Middle Name',
                          validators=[DataRequired(),
                                   Length(min=1, max=20)])
    first_name = StringField('First Name',
                         validators=[DataRequired(),
                                Length(min=1, max=20)])
    password = PasswordField('Password',
                         validators=[DataRequired(),
                                Length(min=2)])
    email = StringField('Email',
                        validators=[DataRequired(),
                                Email()])
    picture = FileField('Update Profile Picture', validators=[FileAllowed(['jpg', 'png', 'jpeg'])])
    submit = SubmitField('Update')

    def validate_email(self, email):
        _, all_emails_from_database = User.find_all_emails_and_registration_number()
        if email.data != current_user.email:
```

```
            if email.data in all_emails_from_database:
                raise ValidationError("That email is taken. Please choose another one!")


class UpdateAdminAccountForm(FlaskForm):
    registration_number = StringField('Username',
                                      validators=[DataRequired(),
                                                  Length(min=1)])
    surname = StringField('Surname',
                          validators=[DataRequired(),
                                      Length(min=1, max=20)])
    middle_name = StringField('Middle Name',
                              validators=[DataRequired(),
                                          Length(min=1, max=20)])
    first_name = StringField('First Name',
                             validators=[DataRequired(),
                                         Length(min=1, max=20)])
    password = PasswordField('Password',
                             validators=[DataRequired(),
                                         Length(min=2)])
    email = StringField('Email',
                        validators=[DataRequired(),
                                    Email()])
    picture = FileField('Update Profile Picture', validators=[FileAllowed(['jpg', 'png', 'jpeg'])])
    submit = SubmitField('Update')

    def validate_email(self, email):
        _, all_emails_from_database = User.find_all_emails_and_registration_number()
        if email.data != current_user.email:
            if email.data in all_emails_from_database:
```

```
            raise ValidationError("That email is taken. Please choose another one!")


class AdminUpdateStudentAccountForm(FlaskForm):
    registration_number = StringField('Registration Number',
                                      validators=[DataRequired(),
                                                  Length(min=1)])
    surname = StringField('Surname',
                          validators=[DataRequired(),
                                      Length(min=1, max=20)])
    middle_name = StringField('Middle Name',
                              validators=[DataRequired(),
                                          Length(min=1, max=20)])
    first_name = StringField('First Name',
                             validators=[DataRequired(),
                                         Length(min=1, max=20)])
    email = StringField('Email',
                        validators=[DataRequired(),
                                    Email()])
    submit = SubmitField('Update')

    def validate_email(self, email):
        _, all_emails_from_database = User.find_all_emails_and_registration_number()
        if email.data:
            if email.data in all_emails_from_database:
                raise ValidationError("That email is taken. Please choose another one!")


class SelectElectiveCourses(FlaskForm):
    user_type = SelectField('Select Suited Area Of Specialization', validators=[DataRequired()],
```

```
                    choices=(("ai", "Artificial Intelligence"), ("cn", "Computer Networks"),
                             ("se", "Software Engineering"), ("sye", "Systems Engineering")))
    submit = SubmitField('START TEST')


class StartQuiz(FlaskForm):
    submit = SubmitField('START TEST')


class QuestionForm(FlaskForm):
    question_option = RadioField("Answers", coerce=str)
    submit_next = SubmitField('NEXT')
    # submit_previous = SubmitField('PREVIOUS')


#USER LOGIN SYSTEM



import datetime
import sqlite3
import uuid
from flask_login import UserMixin
from extensions import login_manager
from utils import Utils


@login_manager.user_loader
def load_user(user_id):
    return User.find_by_id(user_id)
```

```
class User(UserMixin):

    date_time = str(datetime.datetime.utcnow()).split()
    date, time = date_time
    date = str(date)
    time = time.split(".")
    time = time[0].__str__()

    def __init__(self, inc_id=None, reg_number=None, surname=None, middle_name=None,
                 first_name=None, email=None, password=None, _id=None, timestamp=time,
                 date=date, default_image=None, account_type=None):
        self.inc_id = inc_id
        self.reg_number = reg_number
        self.surname = surname
        self.middle_name = middle_name
        self.first_name = first_name
        self.email = email
        self.password = password
        self.id = uuid.uuid4().__str__() if _id is None else _id
        self.timestamp = timestamp
        self.date_registered = date
        self.default_image = "default.png" if default_image is None else default_image
        self.account_type = account_type

    def save_to_db(self):
        """
        This saves the question to the database
        Returns: A notification string
```

```
        """
        connection = sqlite3.connect("./database/Credentials.db")
        cursor = connection.cursor()

        query = "INSERT INTO users_credential VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
        cursor.execute(query, (self.reg_number, self.surname, self.middle_name, self.first_name, self.email,
                               self.password, self.id, self.timestamp, self.date_registered, self.default_image,
                               self.account_type,))
        connection.commit()
        connection.close()

    def create_admin(self, surname, middle_name, first_name, email, password, username):
        self.account_type = "admin"
        connection = sqlite3.connect("./database/credentials.db")
        cursor = connection.cursor()

        query = "INSERT INTO users_credential VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
        cursor.execute(query, (username, surname, middle_name, first_name, email,
                               password, self.id, self.timestamp, self.date_registered, self.default_image,
                               self.account_type,))
        connection.commit()
        connection.close()

    def insert_student_into_db(self, surname, middle_name, first_name, reg_number, email, password):
        encrypted_password = Utils.encrypt_password(password=password)
        self.account_type = "student"
        connection = sqlite3.connect("./database/credentials.db")
        cursor = connection.cursor()

        query = "INSERT INTO users_credential VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
```

```
        cursor.execute(query, (reg_number, surname, middle_name, first_name, email,
                               encrypted_password, self.id, self.timestamp, self.date_registered,
                               self.default_image, self.account_type,))
        connection.commit()
        connection.close()

    @staticmethod
    def update_profile(username, surname, middle_name, first_name, password, email, picture_to_update, user_corresponding_id):
        encrypted_password = Utils.encrypt_password(password=password)
        connection = sqlite3.connect('./database/Credentials.db')
        cursor = connection.cursor()

        query = "UPDATE users_credential SET reg_number=?, surname=?, middle_name=?, first_name=?, password=?," \
                "email=?, profile_picture=? WHERE _id=?"
        cursor.execute(query, (username,surname, middle_name, first_name, encrypted_password, email, picture_to_update,
                               user_corresponding_id,))

        connection.commit()
        connection.close()

    @staticmethod
    def update_student_profile_by_admin(reg_number, surname, middle_name, first_name, email, user_corresponding_id):
        connection = sqlite3.connect('./database/Credentials.db')
        cursor = connection.cursor()

        query = "UPDATE users_credential SET reg_number=?, surname=?, middle_name=?, first_name=?," \
                "email=? WHERE _id=?"
```

```
        cursor.execute(query, (reg_number, surname, middle_name, first_name, email, user_corresponding_id,))

        connection.commit()
        connection.close()

    @staticmethod
    def update_password(new_password, user_corresponding_id):
        connection = sqlite3.connect('./database/Credentials.db')
        cursor = connection.cursor()

        query = "UPDATE users_credential SET password=? WHERE _id=?"
        cursor.execute(query, (new_password, user_corresponding_id,))

        connection.commit()
        connection.close()

    @staticmethod
    def update_email(email_update, user_corresponding_id):
        connection = sqlite3.connect('./database/Credentials.db')
        cursor = connection.cursor()

        query = "UPDATE users_credential SET email=? WHERE _id=?"
        cursor.execute(query, (email_update, user_corresponding_id,))

        connection.commit()
        connection.close()

    @staticmethod
    def update_profile_picture(picture_file, user_corresponding_id):
```

```
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()

    query = "UPDATE users_credential SET profile_picture=? WHERE _id=?"
    cursor.execute(query, (picture_file, user_corresponding_id,))

    connection.commit()
    connection.close()

@classmethod
def find_by_registration_number(cls, reg_number):
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()

    query = "SELECT * FROM users_credential WHERE reg_number=?"
    result = cursor.execute(query, (reg_number,))
    row = result.fetchone()
    if row:
        user = cls(*row)  # same as row[0], row[1], row[2]...passing args by position
    else:
        user = None

    connection.close()
    return user

@classmethod
def find_by_email(cls, email):
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()
```

```
    query = "SELECT * FROM users_credential WHERE email=?"
    result = cursor.execute(query, (email,))
    row = result.fetchone()
    if row:
        user = cls(*row)  # same as row[0], row[1], row[2]...passing args by position
    else:
        user = None

    connection.close()
    return user

@staticmethod
def find_all_emails_and_registration_number():
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()
    query = "SELECT * FROM users_credential ORDER BY email ASC "
    result = cursor.execute(query, )
    rows = result.fetchall()
    new_registration_number = []
    new_email = []
    for row in rows:
        new_registration_number.append(row[1])
        new_email.append(row[5])
    return new_registration_number, new_email

@classmethod
def find_by_id(cls, _id):
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()
```

```
    query = "SELECT * FROM users_credential WHERE _id=?"
    result = cursor.execute(query, (_id,))
    row = result.fetchone()
    if row:
        user = cls(*row)  # same as row[0], row[1], row[2]...passing args by position
    else:
        user = None

    connection.close()
    return user

@classmethod
def fetch_all_students_by_account_type(cls):
    student = []
    connection = sqlite3.connect('./database/Credentials.db')
    cursor = connection.cursor()

    query = "SELECT * FROM users_credential WHERE account_type='student'"
    result = cursor.execute(query,)
    rows = result.fetchall()
    if rows:
        for row in rows:
            student.append(row)
    else:
        student = []

    connection.close()
    return student
```

#APP CREATION AND RATING BY STUDENTS

```
from flask import Flask
# Blueprints Imports
from blueprints.page import page
from blueprints.users import user
from blueprints.questions import aos_test, elective_course

# extensions Import
from extensions import mail, csrf, login_manager

CELERY_TASK_LIST = ['blueprints.contact.tasks', ]

# app = Flask(__name__, instance_relative_config=True)
# app.config.from_object('config.settings')
# app.config.from_pyfile('settings.py', silent=True)
#
# app.register_blueprint(page)


def create_app(settings_override=None):
    """
    Create a Flask application using the app factory pattern.
    :param settings_override: Override settings
    :return: Flask app
    """
    application = Flask(__name__, instance_relative_config=True)
```

```
    application.config.from_object('config.settings')
    application.config.from_pyfile('settings.py', silent=True)

    if settings_override:
        application.config.update(settings_override)

    application.register_blueprint(page)
    application.register_blueprint(user)
    application.register_blueprint(aos_test)
    application.register_blueprint(elective_course)
    extensions(application)

    return application


def extensions(our_app):

    mail.init_app(our_app)
    csrf.init_app(our_app)
    login_manager.init_app(our_app)
    login_manager.login_view = 'user.login'
    login_manager.login_message_category = 'info'
    return None


#CONTACT US


from flask_mail import Mail
from flask_wtf import CSRFProtect
```

```
from flask_login import LoginManager

mail = Mail()
csrf = CSRFProtect()
login_manager = LoginManager()


#HOW TO USE


from app import create_app
from models.users.users import User
from utils import Utils


if __name__ == '__main__':
    app = create_app()
    with open("first_time_server_run.txt", "r") as new_file:
        content = new_file.read()
        if content == "":
            var = True
            while var:
                print("Welcome Admin Please put in the following Credentials")
                surname = input("Surname: ")
                middle_name = input("Middle Name: ")
                first_name = input("First Name: ")
                user_name = input("Username: ")
                email = input("E-mail: ")
                password = input("Password: ")
```

```
            if surname != "" and middle_name != "" and first_name != "" and email != "" and user_name !=
"" \
                    and password != "":
                encrypted_password = Utils.encrypt_password(password)
                grand_admin = User()
                grand_admin.create_admin(surname=surname, middle_name=middle_name, email=email,
                                         first_name=first_name,                password=encrypted_password,
username=user_name)
                with open("first_time_server_run.txt", "a") as new_file_write:
                    new_file_write.write("true")
                    var = False
                break
            else:
                continue
    app.run()



#UTILS



from passlib.hash import pbkdf2_sha512
import constants
import re



class Utils(object):

    @staticmethod
    def encrypt_password(password):
```

```
    return pbkdf2_sha512.encrypt(password)

@staticmethod
def check_encrypted_password(password, hashed_password):

    return pbkdf2_sha512.verify(password, hashed_password)

@staticmethod
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in constants.ALLOWED_EXTENSIONS

@staticmethod
def strong_password(password_to_check):
    a = b = c = d = e = f = ''
    try:
        matcher_digits = re.compile(r'[0-9]+')
        matcher_lowercase = re.compile(r'[a-z]+')
        matcher_uppercase = re.compile(r'[A-Z]+')
        matcher_special = re.compile(r'[\W.\\?\[\]|+*$()_^{\}]+')

        mo_digits = matcher_digits.search(password_to_check)
        mo_lowercase = matcher_lowercase.search(password_to_check)
        mo_uppercase = matcher_uppercase.search(password_to_check)
        mo_special = matcher_special.search(password_to_check)

        if mo_digits and mo_lowercase and mo_uppercase and mo_special:
            return None
        if not mo_digits or not mo_lowercase or not mo_uppercase or not mo_special:
            if not mo_special:
```

```
            a += "one special character is required"
        if not mo_digits:
            b += "a number is required"
        if not mo_lowercase:
            c += "a lowercase letter is required"
        if not mo_uppercase:
            d += "an uppercase letter is required"
    if not mo_digits and not mo_lowercase and not mo_uppercase and not mo_special:
        e += "Password should include a Lowercase, a Uppercase, Numbers and special characters"
    return a, b, c, d, e
  except Exception as _:
    f += "Password should include a Lowercase, a Uppercase, Numbers and special characters"
    return f

@staticmethod
def check_reg_number(reg_num):
  try:
    matcher = re.compile(r'\d{4}/\d{6}')
    matching_reg_number = matcher.search(reg_num)

    reg_num_format_length = reg_num.split("/")
    reg_num_format_length_first = reg_num_format_length[0]
    reg_num_format_length_last = reg_num_format_length[1]

    if matching_reg_number and \
        len(reg_num_format_length_first) == 4 and \
        len(reg_num_format_length_last) == 6 and \
        len(reg_num_format_length) == 2:
      return None
    else:
```

```
            return "Incorrect formatted Registration Number"
        except Exception as _:
            return "Incorrect formatted Registration Number"
```

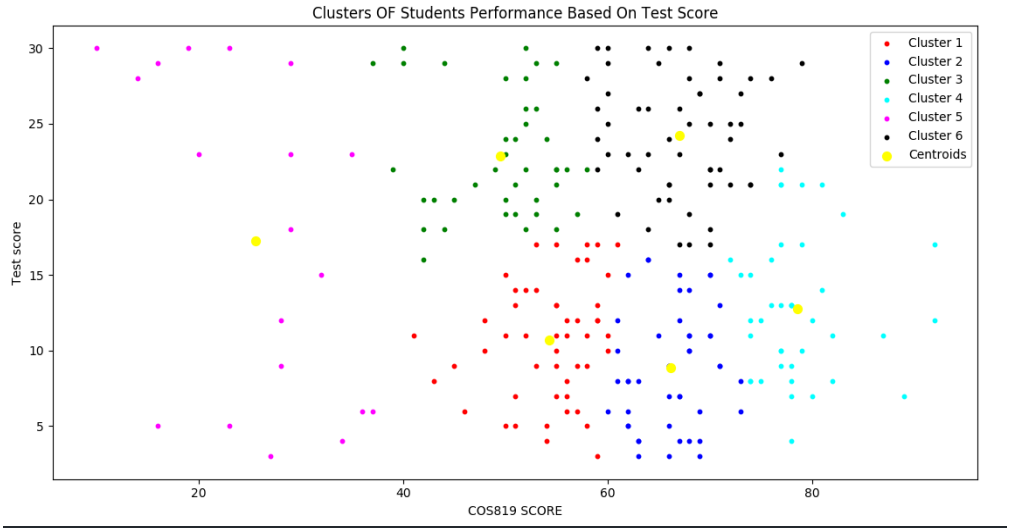
WORKING OF OUR CODE:

This project involves two part, the first part deals with the selection of Area of Specialization for a student, this is done by asking series of questions and then a ranking system is used to generate an area of specializtion from the persons score.

There are total nine sections in our code , In the very first section with the help of sqlite3 we are taking all the necessary information of the student from LPU database, then we are filtering them on the basis of marks so that we can provide them with the list of minors which will be best suited for them.

Clustering

This is done to analyse the previous records of students in the selected area of specialization and a pie chart is shown to the user to give a detailed anaysis of the rate of failure to success in the area of specializtion selected.



Model Selection

The second part of the project has to do with the Selection of elective courses in the preferred area of specialization. Here, the data is binned and the courses are ranked based on the model decision trees which was evaluated and then the top four courses are selected.

The models used for training are:

- Decision Trees
- Support Vector Machine (Linear and Kernel)
- K-Nearest Neighbour
- Naïve Bayes
- Logistic Regression

Now comes the next i.e. application details here they will be providing us with their personal details like Name, Place, Regno. , etc. to authenticate that wheather they are from our university or not if not then also they are allowed to participate with the option of external courses.

Now comes the next section here the Admin creates the students as uts assumed that each student is registered for the session No Registration was performed.

The following below are the task list:

- ☑ Login

- [x] Update Profile
- [x] Online Quiz
- [x] Admin Dashboard
- [x] Create Users
- [x] Delete Users
- [x] Regex matching for Registration Number
- [x] Online Prediction
- [x] Online viewing of pie charts for clustered analysis
- [ ] Validation Emails
- [ ] saving individual scores after each predictions for personalised performance anyalysis
- [ ] Admin tracking students preferences

How to run

- First make sure you run database.py first to create a database file
- Secondly make sure the "run_fist_time.txt" is empty,do not delete
- then run python run.py in he root folder

FULL EXPLAINATION OF CODE IS HERE IN THIS AUDIO FILE:-

EXPLAINATION.mp4

Work distribution:

1. **AKSHAT SHARMA:-** COMPLETE PROJECT REPORT HAS BEEN DONE BY HIM & ALSO HELPED IN THE CODING.
2. **MANISH KUMAR SINGH** :- CODING IS DONE BY HIM WITH AKSHAT & ALSO HELPED IN PROJECT REPORT.
3. **Kummari Jaya Ram**:- Helped in the coding section.
4. **Yantrapati Nikhil:-** Helped in the coding section.