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 finishes.

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"

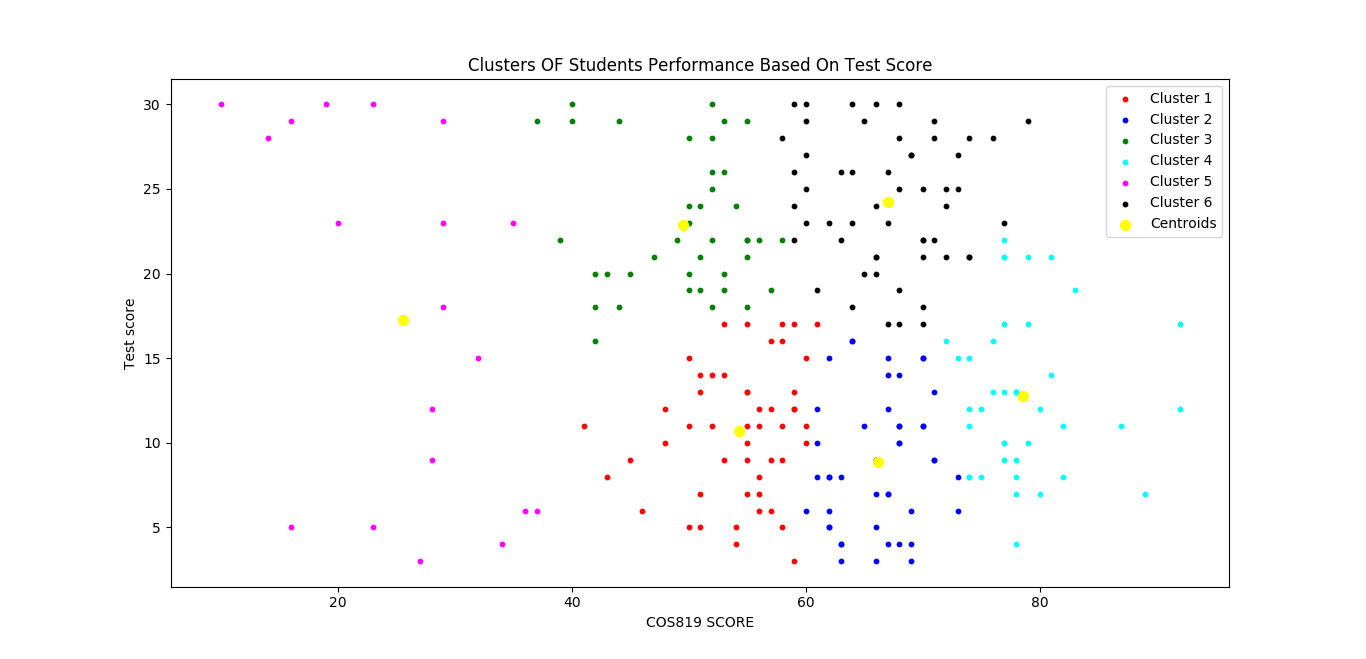
WORKING OF OUR CODE:

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
*  Update Profile
*  Online Quiz
*  Admin Dashboard
*  Create Users
*  Delete Users
*  Regex matching for Registration Number
*  Online Prediction
*  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:-



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.