IBM Project Report

University Admit Eligibility Predictor

1. **Introduction**

1. **Project Overview**

Concerns about getting into college are common among students. This project's goal is to assist students in narrowing down institutions based on their profiles. They have a good idea based on the expected results regarding the likelihood of getting into a specific university. This analysis ought to provide better insight for students who are or will be preparing for exams and building their profiles. The University Admit Predictor is a web-based program that allows students to register with their personal information and academic record to predict college admission. The main motivation for this project is to help students who cannot afford highly-priced predictor systems. This project will reduce the costs of applications for aspirants significantly because an aspirant will be able to know the college, he has a chance at before actually applying to that college

2. **Purpose**

The overall purpose is to talk about how admittance to a university can be predicted using a variety of variables using linear regression. For postgraduates, many potential students submit applications. The admittance decision is based on requirements set forth by the specific college or degree program. In order to forecast graduate school acceptance, the independent factors in this study will be statistically measured. If successful, exploration and data analysis would enable predictive models to better prioritize the application screening process for Master's degree programs, resulting in the admission of the most qualified applicants.

2. **Literature Survey**

1. **Existing Problem**

The existing problem with the available predictors is that most of them charge way too much money for predicted results. They also as for too many details like where did you go schooling? What was your childhood interest? etc. These processes are time-consuming.

2. **References**

[1] M. Omaer Faruq Goni, A. Matin, T. Hasan, M. Abu Ismail Siddique, O. Jyoti and F. M. Sifnatul Hasnain, "Graduate Admission Chance Prediction Using Deep Neural Network," 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), 2020.

[2] Nandal, P., Deep Learning in diverse Computing and Network Applications Student Admission Predictor using Deep Learning (March 28, 2020). Proceedings of the International Conference on Innovative Computing & Communications (ICICC) 2020.

[3] H. Fathiya and L. Sadath, "University Admissions Predictor Using Logistic Regression," 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), 2021.

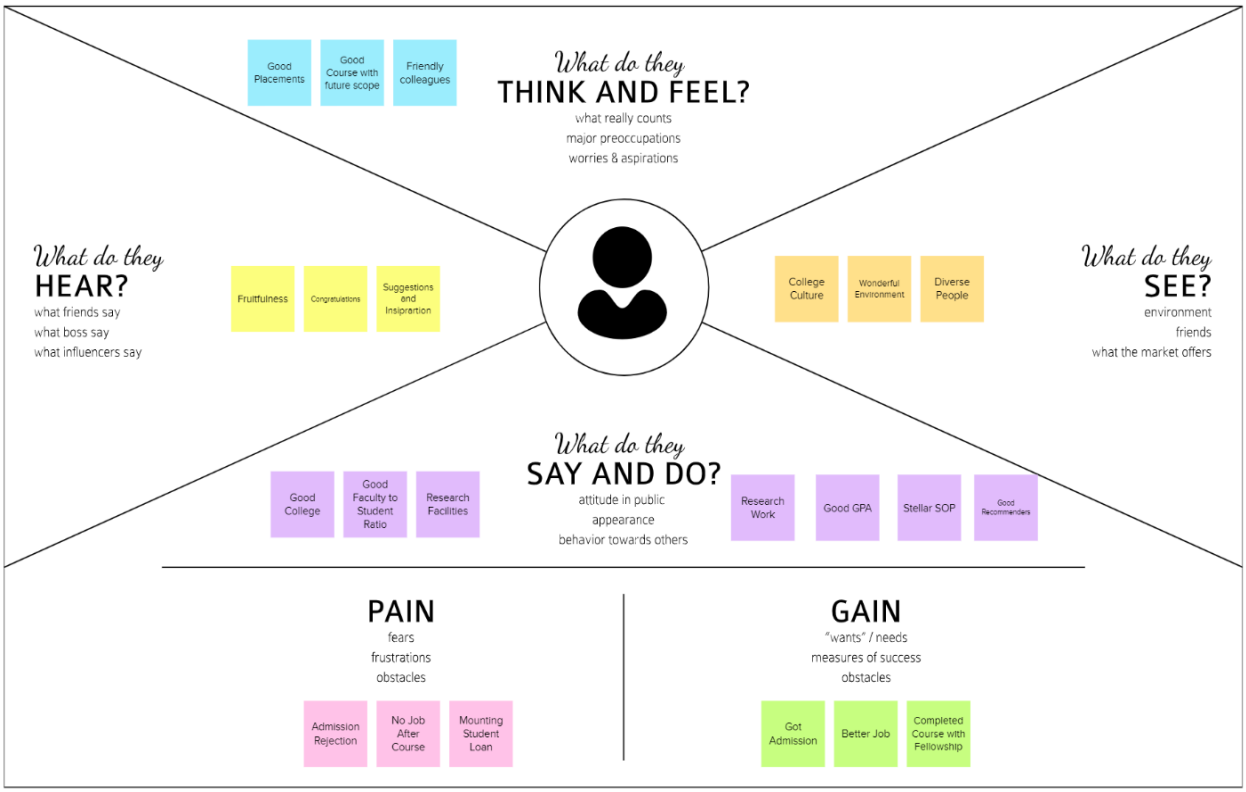
[4] S. Sridhar, S. Mootha and S. Kolagati, "A University Admission Prediction System using Stacked Ensemble Learning," 2020 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA), 2020.
[5] Z. Bitar and A. Al-Mousa, "Prediction of Graduate Admission using Multiple Supervised Machine Learning Models," 2020 SoutheastCon, 2020.

2.3 Problem Statement Definition

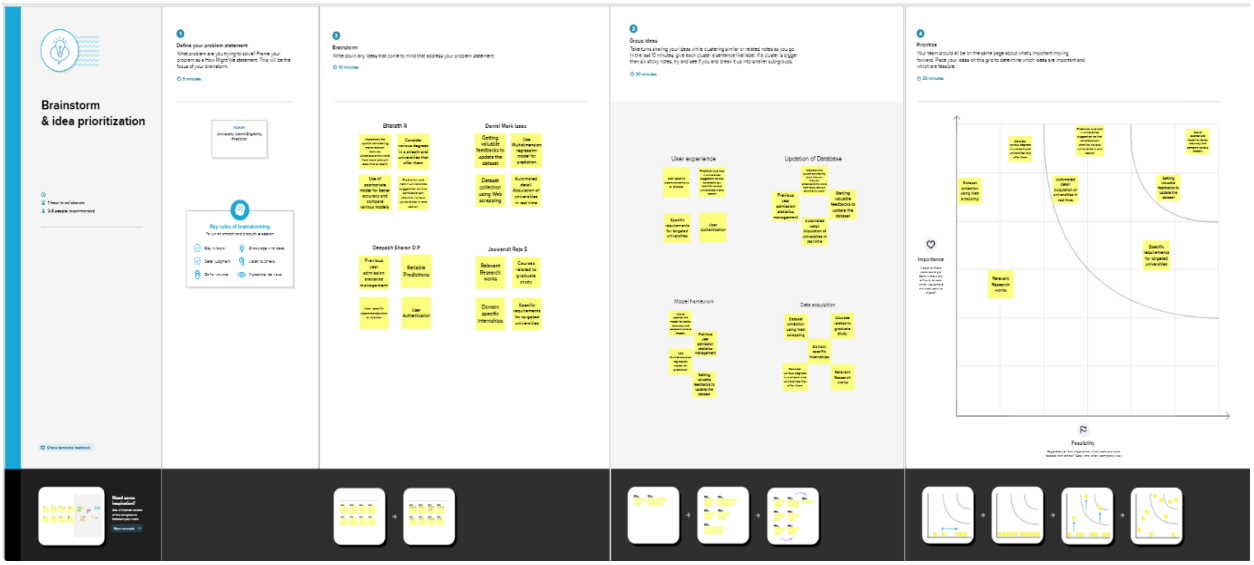
Creating a program that uses machine learning models to forecast a student's prospects of admission to a university. The model will be trained and stored on the IBM cloud. Prediction of student admission to university based on their profile.

3. Ideation and Proposed Solution

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



3.3 Proposed Solution

Proposed Solution Template

Date	27 September 2022
Team ID	PNT2022TMID52898
Project Name	University Admit Eligibility Predictor

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Prediction of student admission to university based on their profile.
2.	Idea / Solution description	In order to guide the students with their admission procedure a Machine Learning based predictor system has to be introduced to provide students with best possible predictions.

3.	Novelty / Uniqueness	So far there are very few solutions that exists which is similar to our proposed solutions. Yocket, College AI, GradCafe are a few. But all those have their own drawbacks in terms of usability and Output. Our solution tries to address these drawbacks in an efficient manner.
4.	Social Impact / Customer Satisfaction	A customer would be able to predict the college he will get beforehand so he will not get disappointed if he didn't get his dream college. It will save money for customers since he will apply for colleges in which he has a better chance of admission.
5.	Business Model (Revenue Model)	This predictor system can be used by consultancy firms to help their customers who seek their help. Apart from this valuable suggestion can be provided for an appropriate subscription fee.
6.	Scalability of the Solution	The solution could be scaled up to include more universities and larger geographical area. But maintenance of large database might pose a few problems.

3.4 Problem Solution Fit

4. Requirement Analysis

4.1 Functional Requirement

Following are the functional requirements of the proposed solution. **FR No.**	**Functional Requirement (Epic)**	**Sub Requirement (Story / Sub-Task)**
FR-1	User Registration	To prevent unauthorised access, users must be able to login using their email Id and password.
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP

FR-3	User Details	Submit the documents such as: 1. GRE, TOEFLT/IELTS Scorecard, 2. Curriculum Vitae, 3. Letter of Recommendation (LOR) 4. Statement of Purpose (SOP)
FR-4	User Profile	User Dashboard containing his/her personal information, wish list, skills, and hobbies.
FR-5	Data Management	A user can create, read, update, and delete data.

4.2 Non-Functional Requirement

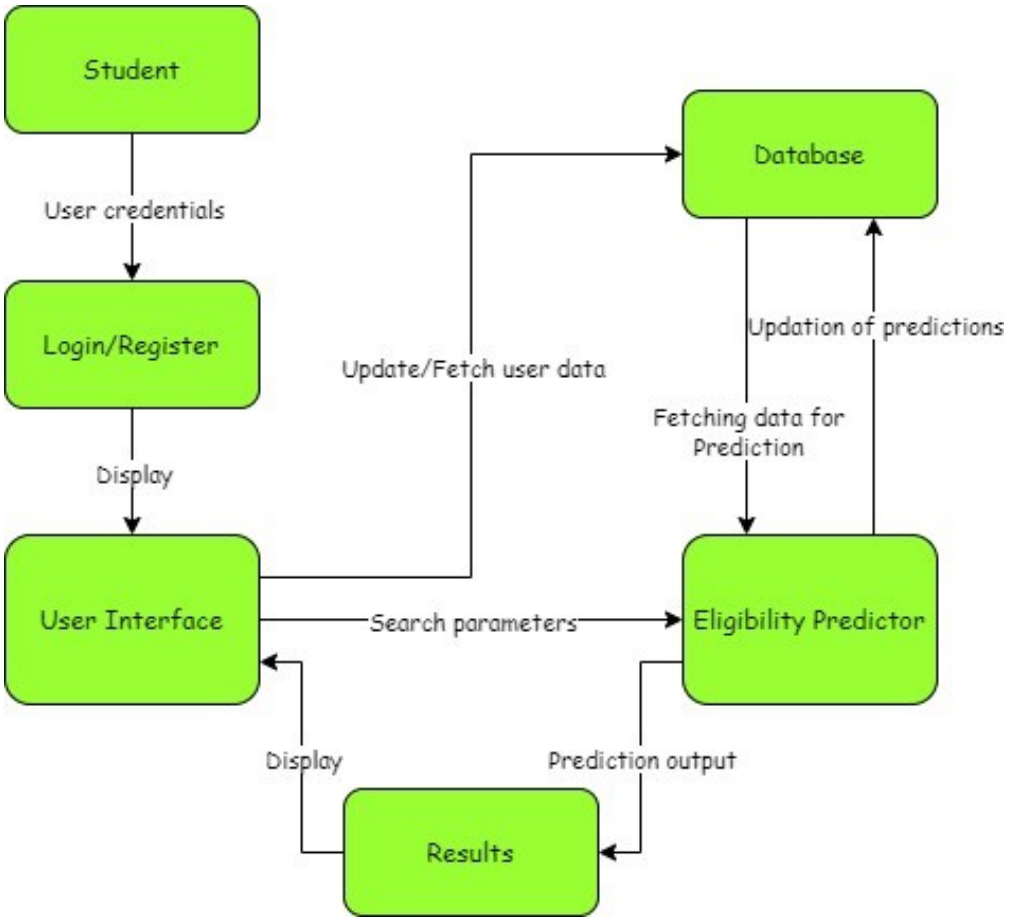
FR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	1. Proposed solution should be user friendly. 2. The system should not require any prior knowledge from the user. 3. The system should be able to load data quickly.
NFR-2	**Security**	1. Only the authorized users can use the site’s services. 2. Some cryptographic techniques need to be used for validation purposes.

NFR-3	**Reliability**	1. Data backups and strategies are to be used to avoid data being lost or data being corrupted. 2. They system should be functional at any time of the day.
NFR-4	**Performance**	1. At any instant the system should be able to support multiple users. 2. The prediction made for user requests must not take more time. Preferably it should be less than 5 seconds.
NFR-5	**Availability**	The system should be functional at any time of the day but in case of error, it should display backup page and retrieve information from backup folder.
NFR-6	**Scalability**	Assesses the highest workloads under which the system will still meet the performance Deals with the measure of the system's response time under different load conditions requirements.

5. **Project Design**

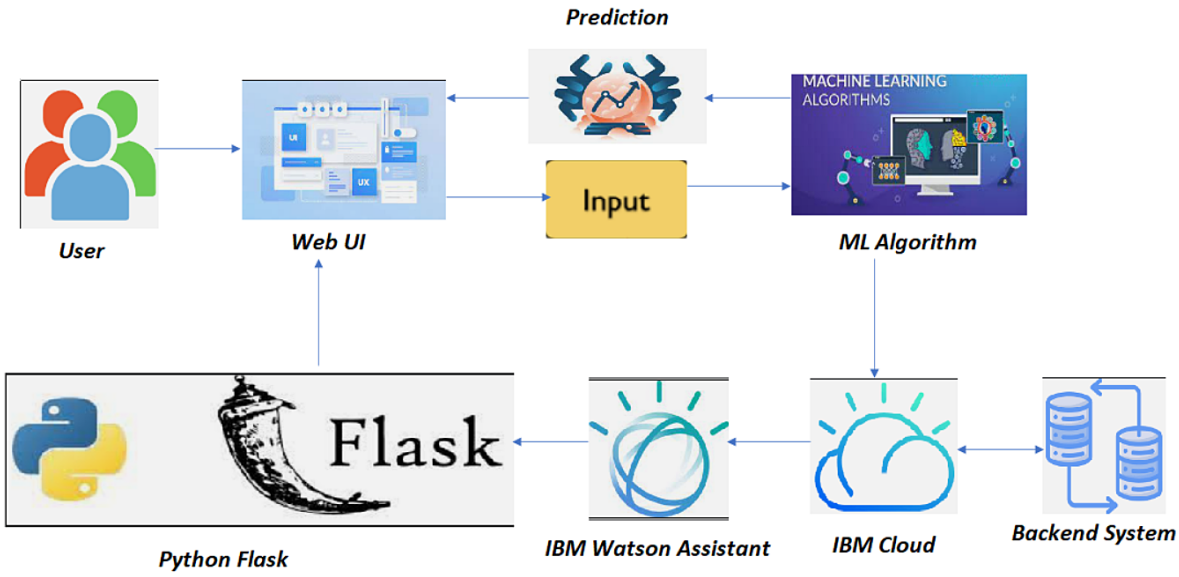
5.1 Data Flow Diagrams

The traditional visual representation of how information moves through a system is a data flow diagram (DFD). A tidy and understandable DFD can graphically represent the appropriate quantity of the system demand. It demonstrates how information enters and exits the system, what modifies the data, and where information is kept.

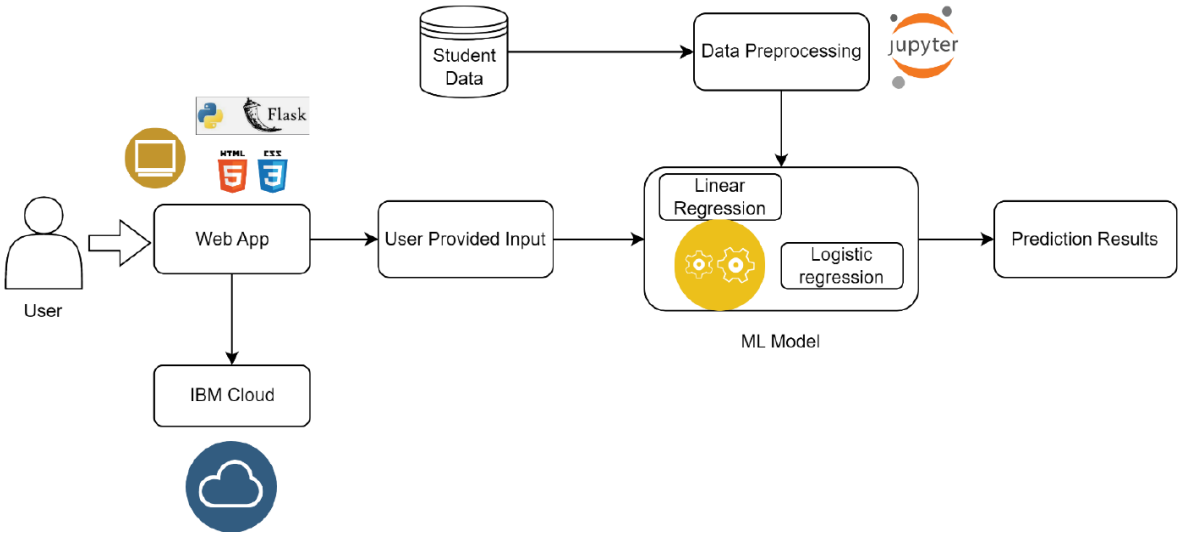


5.2 Solution & Technical Architecture

Solution Architecture



Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Student	Registration	USN-1	User can register by entering the username, email ID and password	I can access my account / dashboard	Medium	Sprint-2
	Login	USN-2	As a user, I can log into the application by entering email & password	I can access my account	Medium	Sprint-2
	Update Profile	USN-3	As a user, after logging in post registration, I will have to update my profile	I can complete the profile updation by filling in details	Medium	Sprint-2
	Prediction Results	USN-4	Exploratory Data Analysis and predict chance of admission using ML models.	I can see the prediction results of chances	High	Sprint-1
	Integrating application with ML model	USN-5	User can get chance of admission after entering the details	I can get chance of admit by filling in the details. The pickle object is integrated with the application	Medium	Sprint-3
	Submit admission data	USN-6	Can submit their admission results with their profile details	I can submit my admission data and receive a confirmation that my data is submitted	Medium	Sprint-4
	IBM Watson Deployment	USN-7	Integrating our application with IBM cloud and deploying it.	I can get reliable prediction in the webpage via the model deployed in cloud.	Medium	Sprint-4

6. Project Planning & Scheduling

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Exploratory Data Analysis	USN-4	Visualizing the dataset and analysing the various trends.	2	High	Bharath R, Daniel Mark Isaac
Sprint-1	Model Building	USN-4	Developing a ML model to predict the chance of admission using the dataset	1	High	Bharath R, Daniel Mark Isaac
Sprint-2	Login	USN-2 USN-3	After login user can update their profile and start predicting their chance of admission	2	Medium	Bharath R, Deepakh Sharan D P
Sprint-2	Register	USN-1	After registering, user can login to view his/her account.	2	Medium	Bharath R, Deepakh Sharan D P
Sprint-3	Application Building	USN-1	Integrate the application with pickle object of the model	1	Low	Deepakh Sharan D P, Jaswandt Raja S
Sprint-3	Application Building	USN-5	User can get the chance of admission after entering the details	1	Low	Deepakh Sharan D P, Jaswandt Raja S
Sprint-4	IBM Watson Deployment	USN-7	Integrating our application with IBM cloud and deploying it.	2	Medium	Daniel Mark Isaac, Jaswandt Raja S

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	15 Nov 22
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	16 Nov 22
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	17 Nov 22
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	18 Nov 22

7. **Coding and Solutioning**

7.1 Feature 1

Training The Linear Regression model:

A Linear regression model is trained on the dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv("Admission_Predict.csv")
dataset.head()


x = dataset[['GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Research']]
y = dataset['Chance of Admit ']


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 100)

mlr = LinearRegression()
mlr.fit(x_train, y_train)

print("Intercept: ", mlr.intercept_)
print("Coefficients:")
list(zip(x, mlr.coef_))
```

Testing the model and evaluating performance metrics:

```
#Prediction of test set

y_pred_mlr= mlr.predict(x_test)

#Predicted values
print("Prediction for test set: {}".format(y_pred_mlr))

mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr})
mlr_diff.head()

from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))
print('R squared: {:.2f}'.format(mlr.score(x,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Intercept:  -1.228285398131014
Coefficients:
Prediction for test set: [0.79793967 0.74792724 0.86871864 0.8856855  0.68403746 0.85370856
 0.67161949 0.7408362  0.90663309 0.58312346 0.69567936 0.70838948
 0.63860073 0.67760163 0.61539916 0.66737881 0.86000658 0.60810758
 0.57801389 0.83481169 0.63781003 0.84700949 0.48541665 0.80094769
 0.46753179 0.74748586 0.73368904 0.77987137 0.53655206 0.95974245
 0.89124757 0.77355509 0.7263695  0.7001736  0.55882143 0.52577021
 0.59819879 0.73301941 0.9236683  0.48339279 0.66735848 0.65187014
 0.78597154 0.59528278 0.77790546 0.41384251 0.61783883 0.88046811
 0.55779058 0.56629417 0.93355935 0.66785998 0.82618029 0.80313414
 0.74052551 0.85051165 0.42295811 0.68660609 0.78066541 0.70283069
 0.81316624 0.52125707 0.63282489 0.8123818  0.61293148 0.73399553
 0.82449109 0.71593313 0.63620347 0.69537323 0.90186005 0.63823487
 0.74906378 0.65595495 0.8590957  0.53615055 0.67875334 0.6856359
 0.67688479 0.85390393 0.84227233 0.90884758 0.69405595 0.48302994
 0.93644606 0.70666517 0.79116822 0.84675279 0.62712644 0.52163612
 0.59865886 0.44918123 0.70366479 0.84887413 0.45395457 0.77122787
 0.83264783 0.68634576 0.64283562 0.7096491  0.68226406 0.80400285
 0.96883608 0.87650171 0.59008477 0.72890014 0.59882941 0.64737974
 0.60326834 0.63152619 0.69819591 0.59452426 0.85136712 0.92622498
 0.83075228 0.64479398 0.62520864 0.59349123 0.6992843  0.68393413]
R squared: 80.17
Mean Absolute Error: 0.04692878285488935
Mean Square Error: 0.003716513784197384
Root Mean Square Error: 0.060963216649036686
```

Deploying the model in cloud:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import model_selection
import warnings
warnings.filterwarnings("ignore")
```

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes y
# You might want to remove those credentials before you share the notebook.
cos_client = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='o7_Ng7bC445bx9xZCqECGhEgfxkZvsBlT42_odbLaFmK',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')

bucket = 'eligibilitypredictionmodelling-donotdelete-pr-dywlho4m5azqwc'
object_key = 'Admission_Predict.csv'

body = cos_client.get_object(Bucket=bucket,Key=object_key)['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

dataset = pd.read_csv(body)
dataset.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

```
dataset.drop('Serial No.',axis = 'columns', inplace = True)
dataset.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

```
x = dataset[['GRE Score','TOEFL Score','University Rating','SOP','LOR ','CGPA','Rese
y = dataset['Chance of Admit ']
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_st

mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

```
LinearRegression()
```

```
y_pred_mlr= mlr.predict(x_test)

from sklearn import metrics
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))
print('R squared: {:.2f}'.format(mlr.score(x,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 80.17
Mean Absolute Error: 0.046928782854889335
Mean Square Error: 0.003716513784197383
Root Mean Square Error: 0.06096321664903668
```

```
y_pred_mlr= mlr.predict([[337,118,4,4.5,4.5,9.65,1],[302,102,1,2,1.5,8,0],[333,117,4
#print(x_test)
#Predicted values
print("Prediction for test set: {}".format(y_pred_mlr))
```

```
Prediction for test set: [0.95174021 0.54237348 0.92612384 0.59008477]
```

```
# create client to access our WML service
from ibm_watson_machine_learning import APIClient

client = APIClient(wml_credentials)
print(client.version)
```

```
1.0.257
```

```
space_id = 'b98a964e-0e59-49d2-bef1-f71a8d8cf1af'
```

```
client.spaces.list(limit=10)
```

```
------------------------------------  --------------------  ------------------------
-
ID                                    NAME                  CREATED
b98a964e-0e59-49d2-bef1-f71a8d8cf1af  UAEP-Deployment-Space  2022-11-18T15:47:48.722
Z
------------------------------------  --------------------  ------------------------
-
```

```
client.set.default_space(space_id)
```

```
'SUCCESS'
```

```
client.software_specifications.list()
```

```
#Upload model
software_spec_uid = client.software_specifications.get_id_by_name("runtime-22.1-py3.
metadata = {
            client.repository.ModelMetaNames.NAME: 'Linear Regression model to predi
            client.repository.ModelMetaNames.TYPE: 'scikit-learn_1.0 ',
            client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: software_spec_uid
}

published_model = client.repository.store_model(model=mlr, meta_props=metadata)
```

```
# Get model details
import json

published_model_uid = client.repository.get_model_uid(published_model)
model_details = client.repository.get_details(published_model_uid)
print(json.dumps(model_details, indent=2))
```

```
This method is deprecated, please use get_model_id()
{
  "entity": {
    "hybrid_pipeline_software_specs": [],
    "software_spec": {
      "id": "12b83a17-24d8-5082-900f-0ab31fbfd3cb",
      "name": "runtime-22.1-py3.9"
    },
    "type": "scikit-learn_1.0"
  },
  "metadata": {
    "created_at": "2022-11-18T16:24:36.447Z",
    "id": "ec0b2afa-09de-47ad-a709-56b524cbf096",
    "modified_at": "2022-11-18T16:24:40.465Z",
    "name": "Linear Regression model to predict the admit chance",
```

```
# Create online deployment
metadata = {
    client.deployments.ConfigurationMetaNames.NAME: "Deployment of Admit Eligibility
    client.deployments.ConfigurationMetaNames.ONLINE: {}
}

created_deployment = client.deployments.create(published_model_uid, meta_props=metad
```

```
#######################################################################################
###

Synchronous deployment creation for uid: 'ec0b2afa-09de-47ad-a709-56b524cbf096' star
ted

#######################################################################################
###


initializing
Note: online_url is deprecated and will be removed in a future release. Use serving_
urls instead.

ready


------------------------------------------------------------------------------------
------------
Successfully finished deployment creation, deployment_uid='10a9a42d-0d6b-42b2-8e89-d
88ce774e56b'
------------------------------------------------------------------------------------
------------
```

```
# list all deployments
client.deployments.list()
```

```
------------------------------------  ------------------------------------------
---  -----  ------------------------
GUID                                  NAME
STATE  CREATED
10a9a42d-0d6b-42b2-8e89-d88ce774e56b  Deployment of Admit Eligibility prediction mod
el   ready  2022-11-18T16:48:59.382Z
5ea14582-7998-4531-805a-9cffc67841e0  Deployment of Admitb Eligibility prediction mo
del  ready  2022-11-18T16:47:15.275Z
------------------------------------  ------------------------------------------
---  -----  ------------------------
```

```
# delete old deployments
client.deployments.delete('5ea14582-7998-4531-805a-9cffc67841e0')
```

```
'SUCCESS'
```

```
client.deployments.list()
```

```
------------------------------------  ------------------------------------------
--  -----  ------------------------
GUID                                  NAME
STATE  CREATED
10a9a42d-0d6b-42b2-8e89-d88ce774e56b  Deployment of Admit Eligibility prediction mod
el  ready  2022-11-18T16:48:59.382Z
------------------------------------  ------------------------------------------
--  -----  ------------------------
```

7.2 Feature 2

Developed a app.py code and integrated it with the deployed model on IBM Cloud.

Respective modules are imported, flask instance is created, local database has been connected. Flask Login has been used for implementing user login.

app.py > ...

```python
from flask import Flask,request,redirect,url_for,flash,render_template
from flask_login.utils import login_required
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import check_password_hash, generate_password_hash
from flask_login import LoginManager, login_user, current_user, logout_user
from forms import *
import numpy as np
#import pickle
import json
import requests

app=Flask(__name__)
#model = pickle.load(open('model.pkl', 'rb'))
app.secret_key='Secret key'
app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///database2.sqlite3'
db=SQLAlchemy()
db.init_app(app)
app.app_context().push()

from model import *

login_manager=LoginManager()
login_manager.login_view='login'
login_manager.init_app(app)

@login_manager.user_loader
def load_user(user_id):
    return user.query.get(int(user_id))

```

Routes for home page and sign up page has been implemented.

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


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

@app.route('/signup',methods=['POST'])
def register():
    from model import user
    email=request.form.get('email')
    username=request.form.get('username')
    password=request.form.get('password')

    User=user.query.filter_by(username=username).first()
    if User:
        flash('Username Already exists')
        return redirect(url_for('signup'))

    new_user=user(username=username, email=email, password=generate_password_hash(password,method='sha256'))
    db.session.add(new_user)
    db.session.commit()
    return redirect(url_for('login'))

```

Routes for login page and dashboard page has been implemented

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

@app.route('/login', methods=['POST'])
def logging_in():
    username=request.form.get('username')
    password=request.form.get('password')
    remember= True if request.form.get('Remember user') else False

    User=user.query.filter_by(username=username).first()

    if User and check_password_hash(User.password, password):
        login_user(User)
        return redirect(url_for('dashboard',current_id=User.id))
    else:
        flash('Please Check your credentials and try again')
        return redirect(url_for('login'))

@app.route('/dashboard/<int:current_id>')
@login_required
def dashboard(current_id):
    students=student.query.filter_by(user_id=current_id)
    return render_template('dashboard.html', collections=students, name=current_user.username)
```

Routes for add profile and edit profile has been implemented with the help of WTforms.

```python
@app.route('/add_profile',methods=['GET','POST'])
def add_profile():
    form=profileform()
    if form.validate_on_submit():
        students=student(name=form.name.data, place=form.place.data, number=form.number.data, occupation=form.occupat
        value=current_user.id
        students.user_id=value
        db.session.add(students)
        db.session.commit()
        return redirect(url_for('dashboard',current_id=value))
    return render_template('add_profile.html', form=form)

@app.route('/edit_profile/<int:student_id>',methods=['GET','POST'])
def edit_profile(student_id):
    form=Editprofileform()
    studente=student.query.filter_by(id=student_id).first()
    if form.validate_on_submit():
        studente.name=form.name.data
        studente.place=form.place.data
        studente.number=form.number.data
        studente.occupation=form.occupation.data
        db.session.add(studente)
        db.session.commit()
        return redirect(url_for('dashboard',current_id=current_user.id))
    return render_template('edit_profile.html', form=form)
```

Routes for delete profile and predict has been implemented.
The deployed model has been used for prediction purposes with the help of the API key.

```python
@app.route('/delete_profile/<int:student_id>')
def delete_deck(student_id):
    studente=student.query.filter_by(id=student_id).first()
    db.session.delete(studente)
    db.session.commit()
    return redirect(url_for('dashboard', current_id=current_user.id))

@app.route('/predict',methods=['GET','POST'])
def predict():
    if request.method=='POST':
        arr = []
        for i in request.form:
            val = request.form[i]
            arr.append(float(val))

        # NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account.
        API_KEY = "gw4WhNuTcyBo4ywiRVKzN3nLK8TarnbgkpRyf_kQUgFe"
        token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
        API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
        mltoken = token_response.json()["access_token"]

        header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

        # NOTE: manually define and pass the array(s) of values to be scored in the next line
        payload_scoring = {"input_data": [{"fields": ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
        'LOR', 'CGPA', 'Research'], "values": [arr]}]}

        response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/10a9a42d-0d6b-42b2-8e89
        headers={'Authorization': 'Bearer ' + mltoken}).json()
```

Route for logout page has been implemented.

```python
        result=response_scoring['predictions'][0]['values']

        value=round(result[0][0]*100,2)
        if value>70:
            return render_template('predict.html',
            pred='High chance of admission.\nChance of admit is {} percent'.format(value))
        else:
            return render_template('predict.html',
            pred='Low chance of admission.\nChance of admit is {} percent'.format(value))
    else:
        return render_template('predict.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))

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

7.3 Database Schema

Two Tables are used. User table and student table.

User table is used for implementing user-registration and user-login.

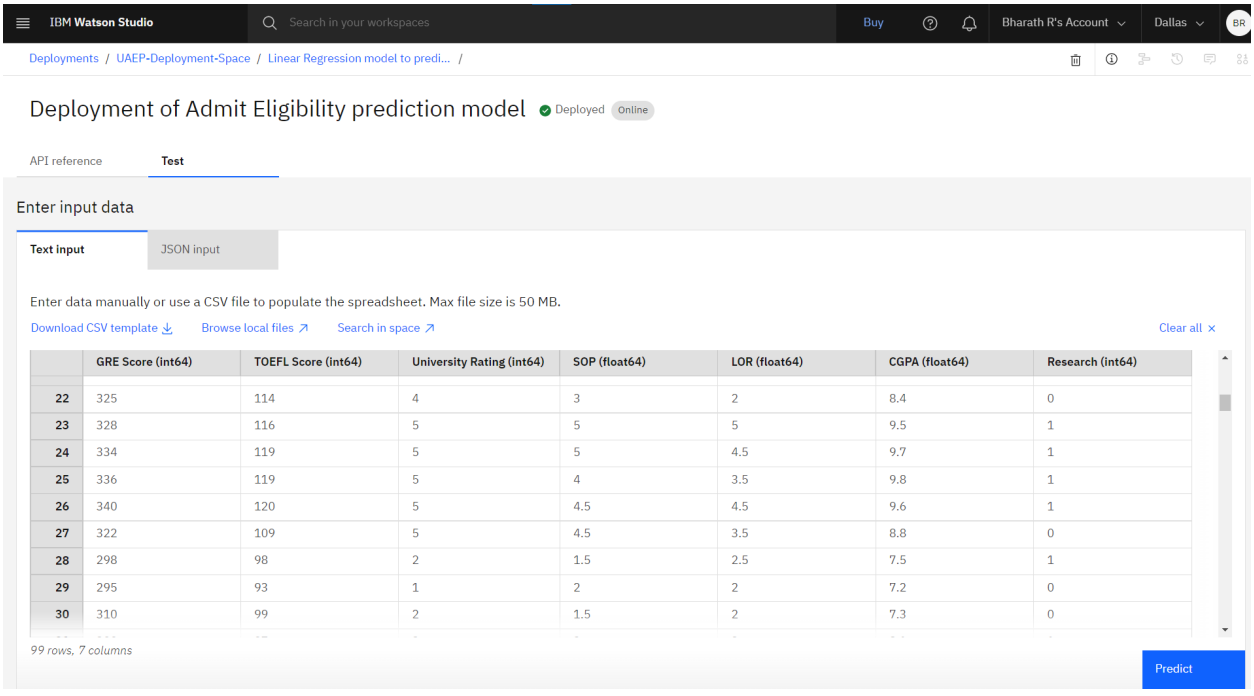
Student table is used for storing profile information.

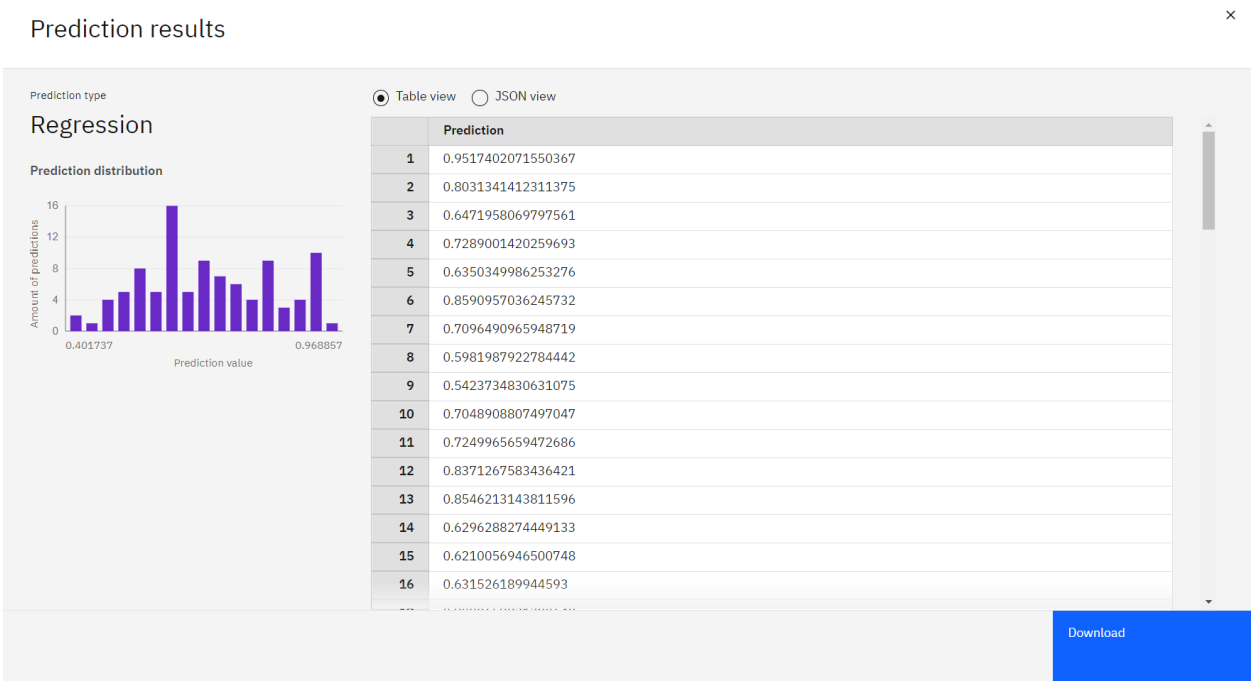
Tables (2)

Name	Type	Schema
student		CREATE TABLE "student" ("id" INTEGER, "name" TEXT NOT NULL, "place" TEXT NOT NULL, "number" TEXT NOT NULL, "occupation" TEXT NOT NULL, "user_id" INTEGER, PRIMARY KEY("id"), FOREIGN KEY("user_id") REFERENCES "user"("id"))
id	INTEGER	"id" INTEGER
name	TEXT	"name" TEXT NOT NULL
place	TEXT	"place" TEXT NOT NULL
number	TEXT	"number" TEXT NOT NULL
occupation	TEXT	"occupation" TEXT NOT NULL
user_id	INTEGER	"user_id" INTEGER
user		CREATE TABLE "user" ("id" INTEGER, "username" TEXT NOT NULL UNIQUE, "email" TEXT NOT NULL UNIQUE, "password" TEXT NOT NULL, PRIMARY KEY("id"))
id	INTEGER	"id" INTEGER
username	TEXT	"username" TEXT NOT NULL UNIQUE
email	TEXT	"email" TEXT NOT NULL UNIQUE
password	TEXT	"password" TEXT NOT NULL

8. Testing

8.1. Test Cases





8.2. User Acceptance Testing

Defect Analysis :

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

Testcase Analysis:

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final ReportOutput	4	0	0	4
Version Control	2	0	0	2

9. **Results**

Performance Metrics

Performance metrics appropriate for linear regression are evaluated.

```
R squared: 80.17
Mean Absolute Error: 0.046928782854889405
Mean Square Error: 0.0037165137841973857
Root Mean Square Error: 0.0609632166490367
```

10. **Advantages and Disadvantages**

Advantages:

- The solution implemented is easy to use and follows a reliable algorithm.
- It helps aspirants an idea about where they stand now and their chances of admission to their list of dream colleges.
- It is fast, efficient, and avoids data redundancy.

Disadvantages:

- Inaccurate results can occur if the input information given is incorrect.
- The initial process can be time-consuming for some users.

11. Conclusion

Therefore, a university admit eligibility predictor is built and implemented with web application and deployed in IBM cloud. The prediction is done using linear regression in python. The web application is built using Flask. User login functionality is also implemented in web site.

12. Future Scope

- The list of universities can be increased with the inclusion of various fields of study.
- The number of countries the predictor system covers can also be increased.

13. Appendix

Source Code

app.py:

```python
from flask import Flask,request,redirect,url_for,flash,render_template
from flask_login.utils import login_required
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import check_password_hash, generate_password_hash
from flask_login import LoginManager, login_user, current_user, logout_user
from forms import *
import numpy as np
#import pickle
import json
import requests

app=Flask(__name__)
#model = pickle.load(open('model.pkl', 'rb'))
app.secret_key='Secret key'
app.config['SQLALCHEMY_DATABASE_URI']='sqlite:///database2.sqlite3'
db=SQLAlchemy()
db.init_app(app)
app.app_context().push()

from model import *

login_manager=LoginManager()
login_manager.login_view='login'
login_manager.init_app(app)

@login_manager.user_loader
def load_user(user_id):
    return user.query.get(int(user_id))

```

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


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

@app.route('/signup',methods=['POST'])
def register():
    from model import user
    email=request.form.get('email')
    username=request.form.get('username')
    password=request.form.get('password')

    User=user.query.filter_by(username=username).first()
    if User:
        flash('Username Already exists')
        return redirect(url_for('signup'))

    new_user=user(username=username, email=email, password=generate_password_hash(password,method='sha256'))
    db.session.add(new_user)
    db.session.commit()
    return redirect(url_for('login'))


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

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

```python
@app.route('/login', methods=['POST'])
def logging_in():
    username=request.form.get('username')
    password=request.form.get('password')
    remember= True if request.form.get('Remember user') else False

    User=user.query.filter_by(username=username).first()

    if User and check_password_hash(User.password, password):
        login_user(User)
        return redirect(url_for('dashboard',current_id=User.id))
    else:
        flash('Please Check your credentials and try again')
        return redirect(url_for('login'))

@app.route('/dashboard/<int:current_id>')
@login_required
def dashboard(current_id):
    students=student.query.filter_by(user_id=current_id)
    return render_template('dashboard.html', collections=students, name=current_user.username)

@app.route('/add_profile',methods=['GET','POST'])
def add_profile():
    form=profileform()
    if form.validate_on_submit():
        students=student(name=form.name.data, place=form.place.data, number=form.number.data, occupation=form.occupation.data)
        value=current_user.id
        students.user_id=value
        db.session.add(students)
        db.session.commit()
        return redirect(url_for('dashboard',current_id=value))
    return render_template('add_profile.html', form=form)
```

```python
@app.route('/edit_profile/<int:student_id>',methods=['GET','POST'])
def edit_profile(student_id):
    form=Editprofileform()
    studente=student.query.filter_by(id=student_id).first()
    if form.validate_on_submit():
        studente.name=form.name.data
        studente.place=form.place.data
        studente.number=form.number.data
        studente.occupation=form.occupation.data
        db.session.add(studente)
        db.session.commit()
        return redirect(url_for('dashboard',current_id=current_user.id))
    return render_template('edit_profile.html', form=form)


@app.route('/delete_profile/<int:student_id>')
def delete_deck(student_id):
    studente=student.query.filter_by(id=student_id).first()
    db.session.delete(studente)
    db.session.commit()
    return redirect(url_for('dashboard', current_id=current_user.id))

@app.route('/predict',methods=['GET','POST'])
def predict():
    if request.method=='POST':
        arr = []
        for i in request.form:
            val = request.form[i]
            arr.append(float(val))

        # NOTE: you must manually set API_KEY below using information retrieved from your IBM Cloud account.
        API_KEY = "gw4WhNuTcyBo4ywiRVKzN3nLK8TarnbgkpRyf_kQUgFe"
        token_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":
        API_KEY, "grant_type": 'urn:ibm:params:oauth:grant-type:apikey'})
        mltoken = token_response.json()["access_token"]
```

```python

        header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

        # NOTE: manually define and pass the array(s) of values to be scored in the next line
        payload_scoring = {"input_data": [{"fields": ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research'], "values": [arr]}]}

        response_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/10a9a42d-0d6b-42b2-8e89-d88ce774e56b/predictions?version=2022-11-18', json=payloa
        headers={'Authorization': 'Bearer ' + mltoken}).json()

        result=response_scoring['predictions'][0]['values']

        value=round(result[0][0]*100,2)
        if value>70:
            return render_template('predict.html', pred='High chance of admission.\nChance of admit is {} percent'.format(value))
        else:
            return render_template('predict.html', pred='Low chance of admission.\nChance of admit is {} percent'.format(value))
    else:
        return render_template('predict.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))

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

Forms.py :

```python
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField, BooleanField
from wtforms.validators import DataRequired

class profileform(FlaskForm):
    name = StringField('Student Name', validators=[DataRequired()])
    place = StringField('Place', validators=[DataRequired()])
    number = StringField('Phone Number', validators=[DataRequired()])
    occupation = StringField('Occupation', validators=[DataRequired()])
    submit = SubmitField('Add')

class Editprofileform(FlaskForm):
    name = StringField('Student Name', validators=[DataRequired()])
    place = StringField('Place', validators=[DataRequired()])
    number = StringField('Phone Number', validators=[DataRequired()])
    occupation = StringField('Occupation', validators=[DataRequired()])
    submit = SubmitField('Edit')
```

Templates :

Add_profile.html :

```html
{% extends "layout.html" %}
{% block body %}
<div class="col-xs-12 col-sm-4 col-lg-3">
    <form method="post">
        {{ form.csrf_token }}
        <div class="form-group">
            {{ form.name.label }}
            {{ form.name(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.place.label }}
            {{ form.place(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.number.label }}
            {{ form.number(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.occupation.label }}
            {{ form.occupation(size=60, class="form-control", placeholder="Name") }}
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </form>
</div>
{% endblock %}
```

Afterlogin.html:

```
<style>
    h3 {text-align: center;}
    h5 {text-align: center;}
    div {text-align: center;}
    .flex-container{display: flex;justify-content: space-around}
</style>
<h3>Hello this is UAEP, where your LIVES are made SIMPLER!!!!</h3>
<h5>If u are a new user, please add your profile using the Add Profile option...</h5>
<h5>If u want to delete your profile please use delete profile option...</h3>
<h5>If u are an exixting user and want to edit your profile, use the Edit Profile option...</h5>
<h5>If u want to start with prediction process, use the Predict option...</h5>
<br>
<br>
<div>
    <a class="btn btn-primary" href="{{url_for('add_profile')}}" role="button">Add Profile</a>
    <a class="btn btn-primary" href="{{url_for('predict')}}" role="button">Predict</a>
  </div>
  <br>
  <div class="flex-container">
    <div class="row">
      <div class="col-lg-6 mb-4">
        {% for collection in collections %}
        <div class="card bg-warning" style="width: 18rem;" >
          <div class="card-body">
            <h5 class="card-title">Profile Information</h5><br>
            <h6 class="card-subtitle mb-2 text-muted">Student Name: {{collection.name}}</h6>
            <h6 class="card-subtitle mb-2 text-muted">Place: {{collection.place}}</h6>
            <h6 class="card-subtitle mb-2 text-muted">Phone Number: {{collection.number}}</h6>
            <h6 class="card-subtitle mb-2 text-muted">Occupation: {{collection.occupation}}</h6>
            <a href="{{url_for('edit_profile',student_id=collection.id)}}" class="card-link"> Edit Profile</a><br>
            <a class="btn btn-danger" href="{{url_for('delete_deck',student_id=collection.id)}}" role="button">Delete</a>
          </div>
        </div>
        <br>
        {% endfor %}

      </div>
    </div>
  </div>
```

Home.html:

```
{% extends "layout.html" %}
{% block body %}
<div class="col-xs-12 col-sm-4 col-lg-3">
    <form method="post">
        {{ form.csrf_token }}
        <div class="form-group">
            {{ form.name.label }}
            {{ form.name(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.place.label }}
            {{ form.place(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.number.label }}
            {{ form.number(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.occupation.label }}
            {{ form.occupation(size=60, class="form-control", placeholder="Name") }}
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </form>
</div>
{% endblock %}
```

Predict.html:

```
{% extends "layout.html" %}
{% block body %}
<style>
    div {text-align: center;}
</style>
<h4>Enter your details to predict chance of admission:::</h4>
<div>
    <form action="{{ url_for('predict')}}" method="post">
        <div>
            <label for="gre_score">GRE:</label>
            <input type="text" id="gre_score" name="gre" placeholder="gre_score">
        </div>
        <br>
        <div>
            <label for="toefl_score">TOEFL:</label>
            <input type="text" id="toefl_score" name="toefl" placeholder="toefl_score">
        </div>
        <br>
        <div>
            <label for="university_rating">RATING:</label>
            <input type="text" id="university_rating" name="rating" placeholder="university_tating">
        </div>
        <br>
        <div>
            <label for="sop_score">SOP:</label>
            <input type="text" id="sop_score" name="sop" placeholder="sop_score">
        </div>
        <br>
        <div>
            <label for="lor_score">LOR:</label>
            <input type="text" id="lor_score" name="lor" placeholder="lor_score">
        </div>
        <br>
        <div>
            <label for="cgpa_score">CGPA:</label>
            <input type="text" id="cgpa_score" name="cgpa" placeholder="cgpa_score">
        </div>
        <br>
        <div>
            <label for="research_score">RESEARCH:</label>
            <input type="text" id="research_score" name="research" placeholder="research_score">
        </div>
        <br>
        <div>
            <button type="submit">Enter</button>
        </div>
        <br>
    </form>
    <br>
    <h4>{{ pred }}</h4>
</div>
{% endblock %}
```

Dashboard.html:

```
{% extends "layout.html" %}
{% block body %}
<h2>Welcome {{name}}</h2>
{% include "afterlogin.html" %}
{% endblock %}
```

Edit_profile.html:

```
{% extends "layout.html" %}
{% block body %}
<div class="col-xs-12 col-sm-4 col-lg-3">
    <form method="post">
        {{ form.csrf_token }}
        <div class="form-group">
            {{ form.name.label }}
            {{ form.name(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.place.label }}
            {{ form.place(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.number.label }}
            {{ form.number(size=60, class="form-control", placeholder="Name") }}
        </div>
        <div class="form-group">
            {{ form.occupation.label }}
            {{ form.occupation(size=60, class="form-control", placeholder="Name") }}
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </form>
</div>
{% endblock %}
```

Layout.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oB
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-blue bg-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">UAEP</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navig
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="/">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/login">Login</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="/signup">Register</a>

         <li class="nav-item">
          <a class="nav-link" href="/logout">Logout</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
    {% block body %}
    {% endblock %}
</body>
</html>
```

Login.html:

```
{% extends "layout.html" %}
{% block body %}
<h2>Login to see dashboard</h2>
    <div class="container">
        {% with messages = get_flashed_messages() %}
        {% if messages %}
            <div class="notification is-danger">
                {{ messages[0] }}
            </div>
        {% endif %}
        {% endwith %}
        <form action="/login" method="POST">
            <div class="form-group">
                <input type="text" name="username" class="form-control" placeholder="username" required><br>
            </div>
            <div class="form-group">
                <input type="password" name="password" class="form-control" placeholder="password" required><br>
            </div>
            <div class="form-group">
                <label class="checkbox">
                    <input type="checkbox">
                    Remember user
                </label>
            </div>
            <div class="form-group">
                <br><button class="btn btn-primary" type="submit">submit</button>
            </div>
        </form>
    </div>
{% endblock %}
```

Signup.html:

```
{% extends "layout.html" %}
{% block body %}
<h1>Sign Up</h1>
    <div class="container">
        {% with messages = get_flashed_messages() %}
        {% if messages %}
            <div class="notification is-danger">
                {{messages[0]}} Go Back to <a href="{{ url_for('login') }}">Login Page</a>
            </div>
        {% endif %}
        {% endwith %}
        <form action="/signup" method="POST">
            <div class="field">
                <input type="text" name="username" class="form-control" placeholder="username" required><br>
            </div>
            <div class="field">
                <input type="email" name="email" class="form-control" placeholder="email" required><br>
            </div>
            <div class="field">
                <input type="password" name="password" class="form-control" placeholder="password" required><br>
            </div>
            <div class="form-group">
                <button class="btn btn-primary" type="submit">submit</button>
            </div>
        </form>
    </div>
{% endblock %}
```

Model.py (schema for local sql database using sqlalchemy):

```
from sqlalchemy.orm import relationship
from app import db
from flask_login import UserMixin

class user(db.Model,UserMixin):
    id=db.Column(db.Integer, primary_key=True)
    username=db.Column(db.String, unique=True, nullable=False)
    email=db.Column(db.String, unique=True, nullable=False)
    password=db.Column(db.String, nullable=False)

class student(db.Model):
    id=db.Column(db.Integer, primary_key=True)
    name=db.Column(db.String, nullable=False)
    place=db.Column(db.String, nullable=False)
    number=db.Column(db.String, nullable=False)
    occupation=db.Column(db.String, nullable=False)
    user_id=db.Column(db.Integer, db.ForeignKey(user.id), nullable=False, )
```

GitHub & Project Demo Link

Github repo link :

https://github.com/IBM-EPBL/IBM-Project-19834-1659707682

Project Demo Link :

https://www.youtube.com/embed/6sa_H0Jmewc