**Identifying Patterns and Trends in Campus Placement Data Using Machine Learning**

**OBJECTIVE:** Campus recruitment is a strategy for sourcing, engaging and hiring young talent for internship and entry-level positions. Our solution revolves around the placement season of a Business School in India. Where it has various factors on candidates getting hired such as work experience, exam percentage etc., Finally it contains the status of recruitment and remuneration details. We will be using algorithms such as KNN, SVM and ANN. We will train and test the data with these algorithms. From this the best model is selected and saved in. pkl format. We will be doing flask integration and IBM deployment.

**Project Flow:**

* **Data collection**:

There are many popular open sources for collecting the data. Eg: kaggle.com Link: <https://www.kaggle.com/code/neesham/prediction-of-placements/data>

* **Data Preparation**:

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps

* **Handling Missing data**:

Handling missing values is a crucial step in data preprocessing and analysis. Missing values can occur in datasets due to various reasons, such as data entry errors, equipment malfunction, or participants not providing certain information. Dealing with missing values appropriately is essential to avoid biased or incorrect results in data analysis. Here are some common methods for handling missing values

Deletion of missing values

Imputation of missing values

Mean, Median, or Mode i

Create indicator variables

Domain-specific knowledge

* **Handling outliers:**

Outliers are the observations in a dataset that deviate significantly from the rest of the data. In any data science project, it is essential to identify and handle outliers, as they can have a significant impact on many statistical methods, such as means, standard deviations, etc., and the performance of ML models.

* **Handling Categorical data:**

Handling categorical data is an important part of data preprocessing and analysis. Categorical data represents qualitative variables with discrete categories or labels

some popular methods in handling categorical data include:

label encoding one hot encoding

**Exploratory Data Analysis**:

Exploratory Data Analysis (EDA) is an approach to analysing and summarizing large datasets to gain insights and understand the underlying patterns, relationships, and trends in the data

**Visual Analysis**:

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data.

* **Univariate analysis:**

In simple words, univariate analysis is understanding the data with a single feature.

* **Bivariate analysis:**

Count plot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

* **Multivariate analysis:**

In simple words, multivariate analysis is to find the relation between multiple features.

### Scaling The Data:

### Scaling data is an essential preprocessing step in many machine learning algorithms, especially those that involve distance-based calculations or gradient-based optimization. Scaling ensures that all features in the dataset are on a similar scale, preventing one feature from dominating others due to its larger magnitude.

Min-Max Scaling (Normalization)

Standardization (Z-score Scaling)

Robust Scaling

Max Abs Scaling

### Splitting The Data into Train and Test:

### Splitting the dataset is crucial for ML model training and evaluation. It involves creating three subsets: training, used to train the model; validation (optional), used for hyperparameter tuning; and test, to assess model generalization. The typical split ratio is 60-80% for training, 10-20% for validation (if used), and 10-20% for testing. Data should be split randomly and represent a fair distribution of samples. Consistency must be maintained to avoid data duplication. Libraries like Scikit-learn aid in easy and randomized dataset splitting, while k-fold cross-validation is used for more reliable performance estimates.

* Data pre-processing

### Model Building:

### Building a model in machine learning is creating a mathematical representation by generalizing and learning from training data. Then, the built machine learning model is applied to new data to make predictions and obtain results.

### Training The Model in Multiple Algorithms:

We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

**SVM model:**

A function named Support vector machine is created and train and test data are passed as the parameters. Inside the function, SVMClassifier algorithm is initialized and training data is passed to the model with. Fit () function. Test data is predicted with. predict () function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

**KNN model:**

A function named KNN is created and train and test data are passed as the parameters. Inside the function, KNeighborsClassifier algorithm is initialized and training data is passed to the model with. fit () function. Test data is predicted with. predict () function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

**Artificial neural network model:**

We will also be using a neural network to train the model.

### Performance Testing & Hyperparameter Tuning:

### Performance testing and hyperparameter tuning are crucial steps in the development of machine learning models.

### Performance testing involves evaluating the model's performance on a separate test dataset to assess its ability to generalize to new, unseen data. This helps identify issues like overfitting or underfitting and ensures the model's reliability in real-world scenarios.

### Hyperparameter tuning is the process of selecting the best hyperparameters for the model to achieve optimal performance. Hyperparameters are configuration settings that are not learned during training, such as learning rate, number of layers, or number of trees.

### Grid search and random search are common techniques for hyperparameter tuning, where different combinations of hyperparameters are tested to find the best combination.

### Both performance testing and hyperparameter tuning are iterative processes that require experimentation and careful analysis to build an accurate and robust machine learning model.

### Model Deployment:

Model deployment is the process of deploying a machine learning model into a production environment, where it can be used to make predictions on new data. It involves taking the trained model and integrating it into an application or system, and making it available to end-users.

**Save The Best Model:**

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

**Integrate With Web Framework:**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

**PROJECT CODE:**

**Importing the libraries:**

import numpy as np

import pandas as pd

import os

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn import preprocessing

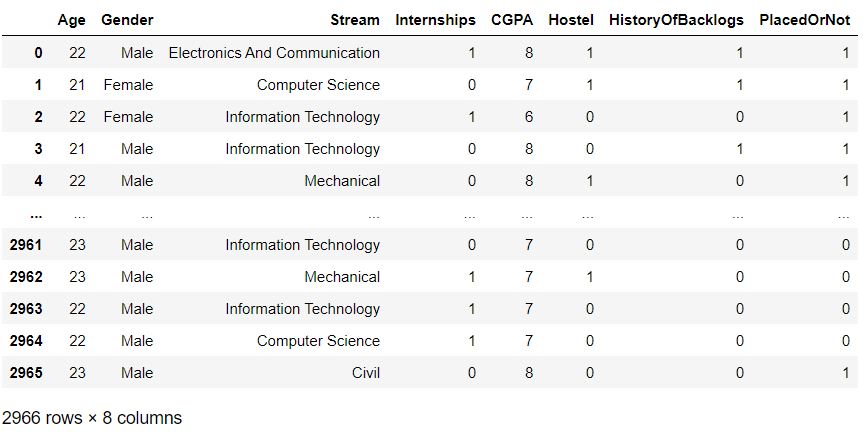
from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import cross\_val\_score

**Read or Load data:**

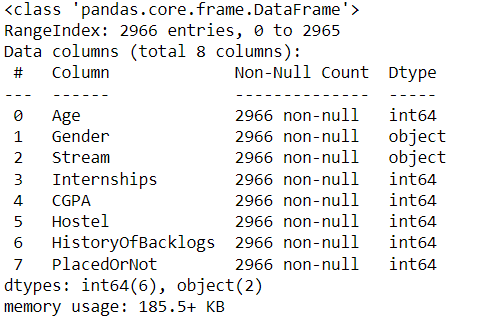
data=pd.read\_csv("collegePlace.csv")

data



**The info() method prints information about the DataFrame.**

data.info()



**The purpose of the given code is to create a transformation plot for the 'Age' feature using the logarithmic transformation.**

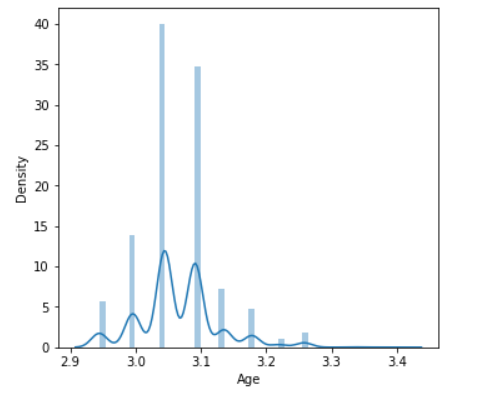
def transformationplot(feature):

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)

sns.distplot(feature)

transformationplot(np.log(data['Age']))



performing ordinal encoding on the 'Gender' and 'Stream' columns in the 'data' DataFrame.

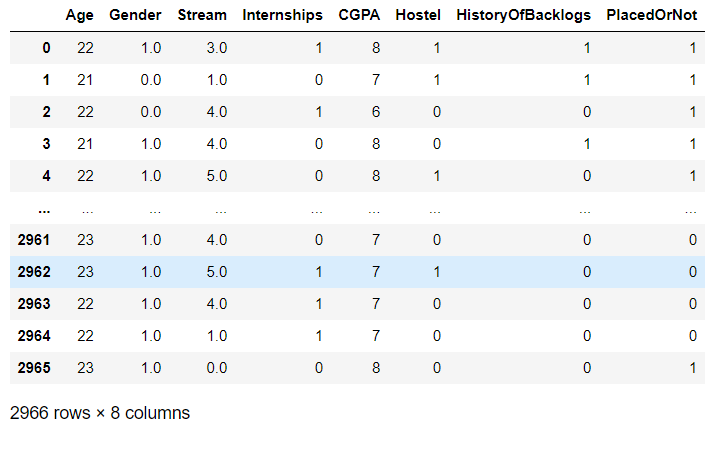
from sklearn.preprocessing import OrdinalEncoder

oe=OrdinalEncoder()

data[['Gender']]=oe.fit\_transform(data[['Gender']])

data[['Stream']]=oe.fit\_transform(data[['Stream']])

data



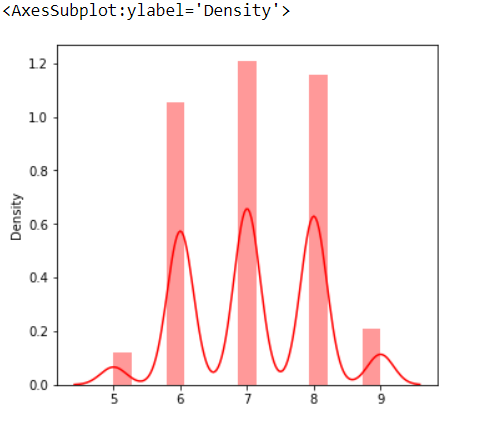
**Univariate Analysis:**

create a visualization of the distribution of the 'CGPA' column and ‘PlacedOrNot’ column from the 'data' DataFrame using a histogram.

plt.figure(figsize=(12,5))

plt.subplot(121)

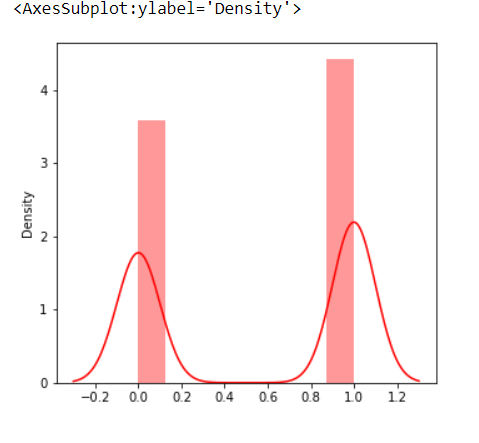
sns.distplot(data[['CGPA']],color='r')



plt.figure(figsize=(12,5))

plt.subplot(121)

sns.distplot(data[['PlacedOrNot']],color='r')



**Bivariate Analysis:**

creates a visualization with two subplots

plt.figure(figsize=(18,4))

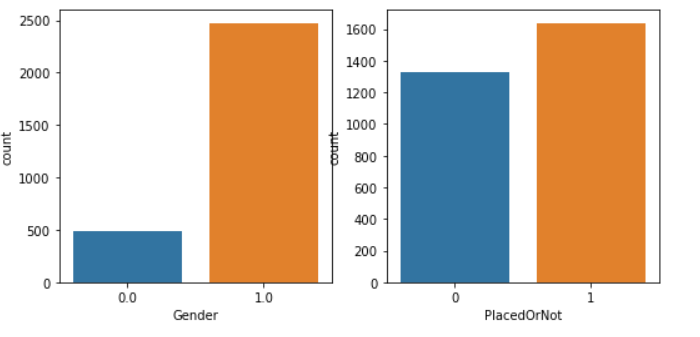
plt.subplot(1,4,1)

sns.countplot(data['Gender'])

plt.subplot(1,4,2)

sns.countplot(data['PlacedOrNot'])

plt.show()



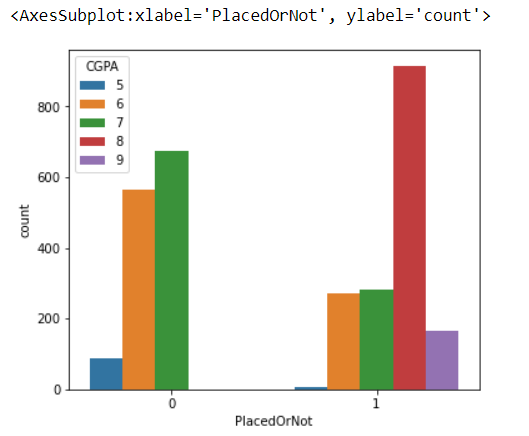
**Multivariate Analysis:**

creates a visualization with three subplots

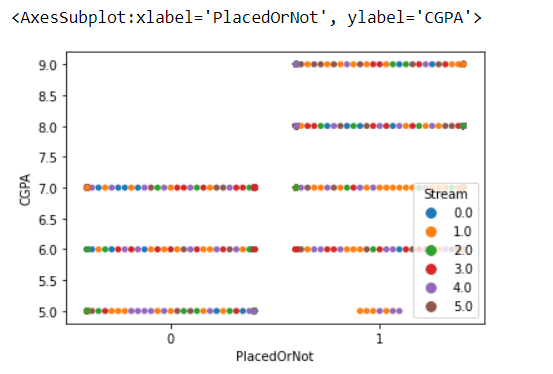
plt.figure(figsize=(20,5))

plt.subplot(131)

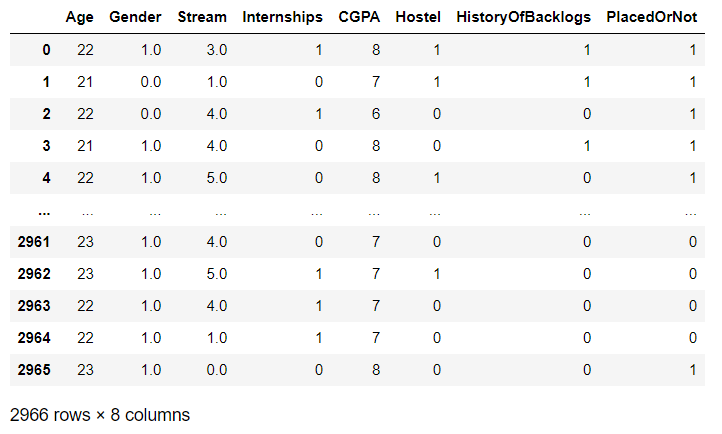
sns.countplot(data['PlacedOrNot'],hue=data['CGPA'])



sns.swarmplot(data['PlacedOrNot'],data['CGPA'],hue=data['Stream'])



data

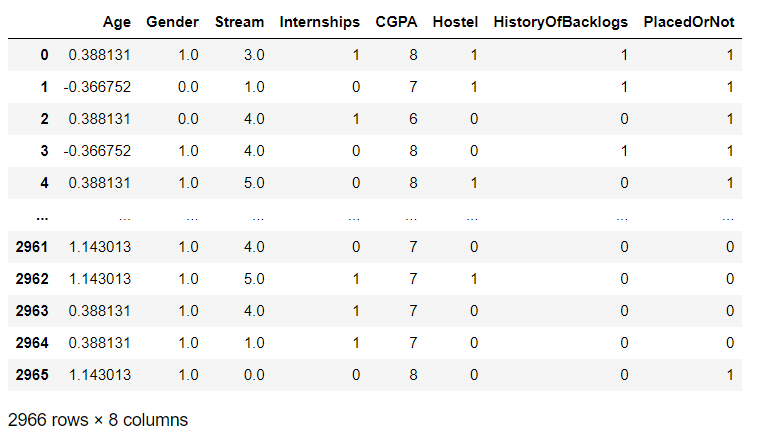


**Scaling:**

sc=StandardScaler()

data[['Age']]=sc.fit\_transform(data[['Age']])

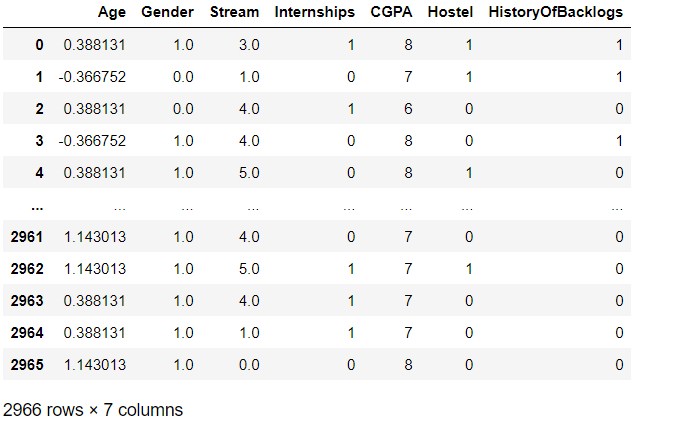
data



**Splitting the data into train and test:**

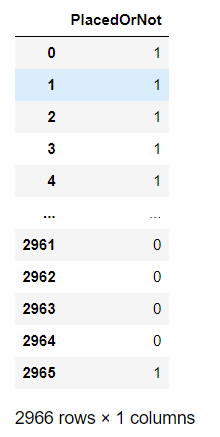
X=data.iloc[:,0:-1]

X



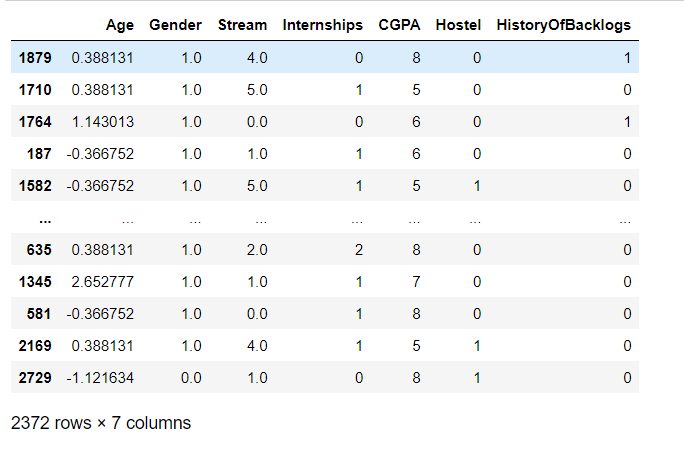
y=data.iloc[:,-1:]

y

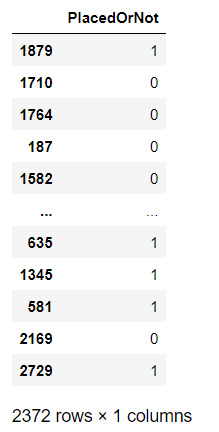


X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,train\_size=0.8,random\_state=16)

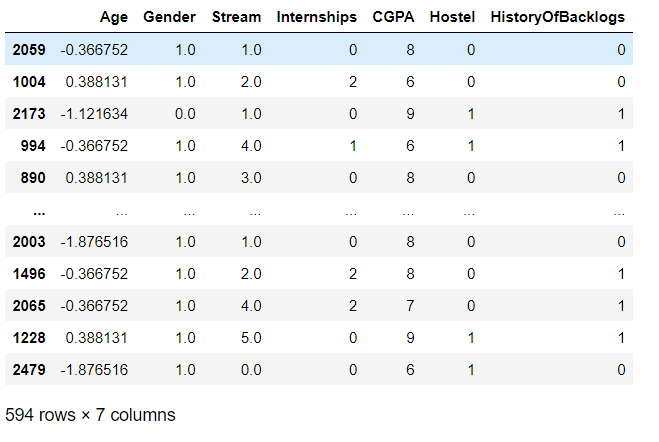
X\_train



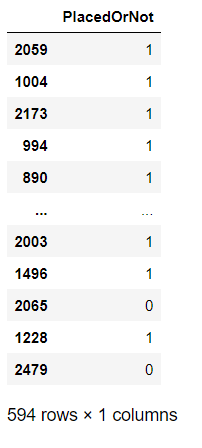
y\_train



X\_test



y\_test



**Model Building:**

**SVM**

from sklearn.svm import SVC

svm=SVC(kernel='linear')

svm.fit(X\_train,y\_train)

**OUTPUT: SVC(kernel='linear')**

pred\_test=svm.predict(X\_test)

pred\_train=svm.predict(X\_train)

train\_accuracy=accuracy\_score(pred\_train,y\_train)

test\_accuracy=accuracy\_score(pred\_test,y\_test)

print("accuracy on training data is",train\_accuracy)

print("accuracy on testing data is",test\_accuracy)

**OUTPUT: accuracy on training data is 0.7765598650927488**

**accuracy on testing data is 0.7474747474747475**

**KNN**

best\_k=0

best\_score=0

for k in range(3,50,2):

knn\_temp=KNeighborsClassifier(n\_neighbors=k)

knn\_temp.fit(X\_train,y\_train)

knn\_temp\_predict=knn\_temp.predict(X\_test)

score=accuracy\_score(y\_test,knn\_temp\_predict)\*100

if score>best\_score and score<100:

best\_score=score

best\_k=k

print("k=",best\_k)

print("accuracy=",best\_score)

**OUTPUT: k= 11**

**accuracy= 87.37373737373737**

**ANN**

import tensorflow as tf

from tensorflow import keras

from keras.models import Sequential

from keras import layers

from keras.layers import Dense

from keras.layers import Dropout

from keras.losses import BinaryCrossentropy

classifier=Sequential()

classifier.add(Dense(7,activation='relu',input\_dim=7))

classifier.add(Dropout(0.50))

classifier.add(Dense(7,activation='relu'))

classifier.add(Dropout(0.50))

classifier.add(Dense(1,activation='sigmoid'))

loss1=BinaryCrossentropy()

classifier.compile(optimizer='Adam',loss=loss1,metrics=['accuracy'])

classifier.fit(X\_train,y\_train,batch\_size=20,epochs=100)

**Conclusion:**

By training the model with multiple algorithms we got the accuracy as follows:

for SVM algorithm we got the accuracy as 74%

for KNN algorithm we got the accuracy as 84% and

for ANN the accuracy is 66%

**By this, we can say that KNN is best suit for predicting Patterns and Trends in Campus Placement Data Using Machine Learning.**

**Creating Simple User Interface Using Gradio:**

import gradio as gr

import pandas as pd

from sklearn.preprocessing import LabelEncoder

def placement\_model(input\_data):

knn\_temp=KNeighborsClassifier(n\_neighbors=best\_k)

knn\_temp.fit(X\_train,y\_train)

input\_data\_2d = np.array(input\_data).reshape(1, -1)

knn\_temp\_predict=knn\_temp.predict(input\_data\_2d)

if knn\_temp\_predict ==0:

return "not placed"

else:

return "placed"

def preprocess\_data(Age, Gender, Stream, Internships, CGPA, Hostel, HistoryOfBacklogs):

le = LabelEncoder()

gender = le.fit\_transform([Gender])[0] if Gender else 0

stream = le.fit\_transform([Stream])[0] if Stream else 0

sc = StandardScaler()

age = sc.fit\_transform([[Age]])[0][0]

input\_data = [age, gender, stream, Internships, CGPA, Hostel, HistoryOfBacklogs]

return input\_data

inputs = ["number", "text", "text", "number", "number", "number", "number"]

outputs = "text"

gr\_interface = gr.Interface(

fn=lambda Age, Gender, Stream, Internships, CGPA, Hostel, HistoryOfBacklogs:

placement\_model(preprocess\_data(Age, Gender, Stream, Internships, CGPA, Hostel, HistoryOfBacklogs)),

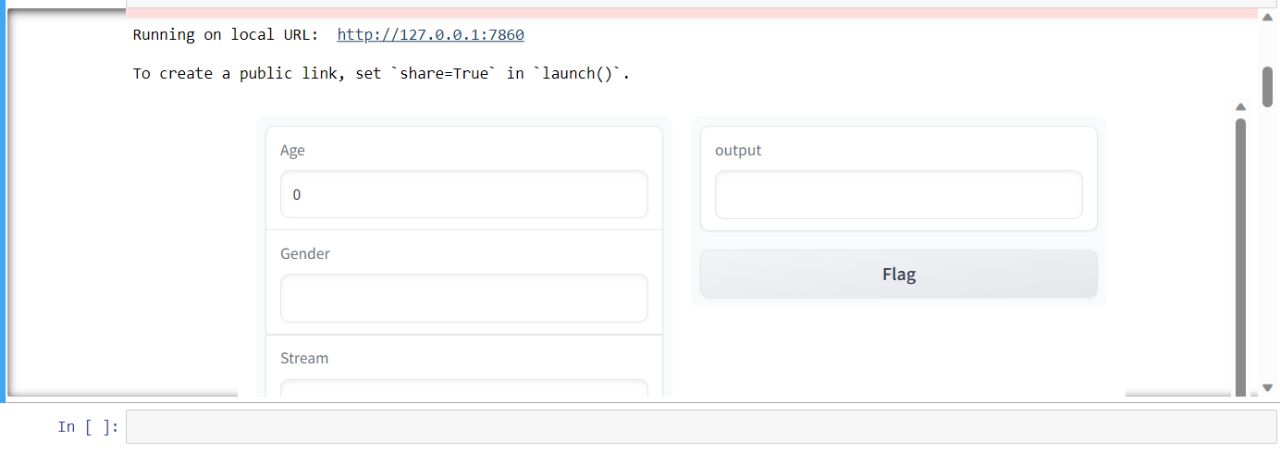
inputs=inputs,

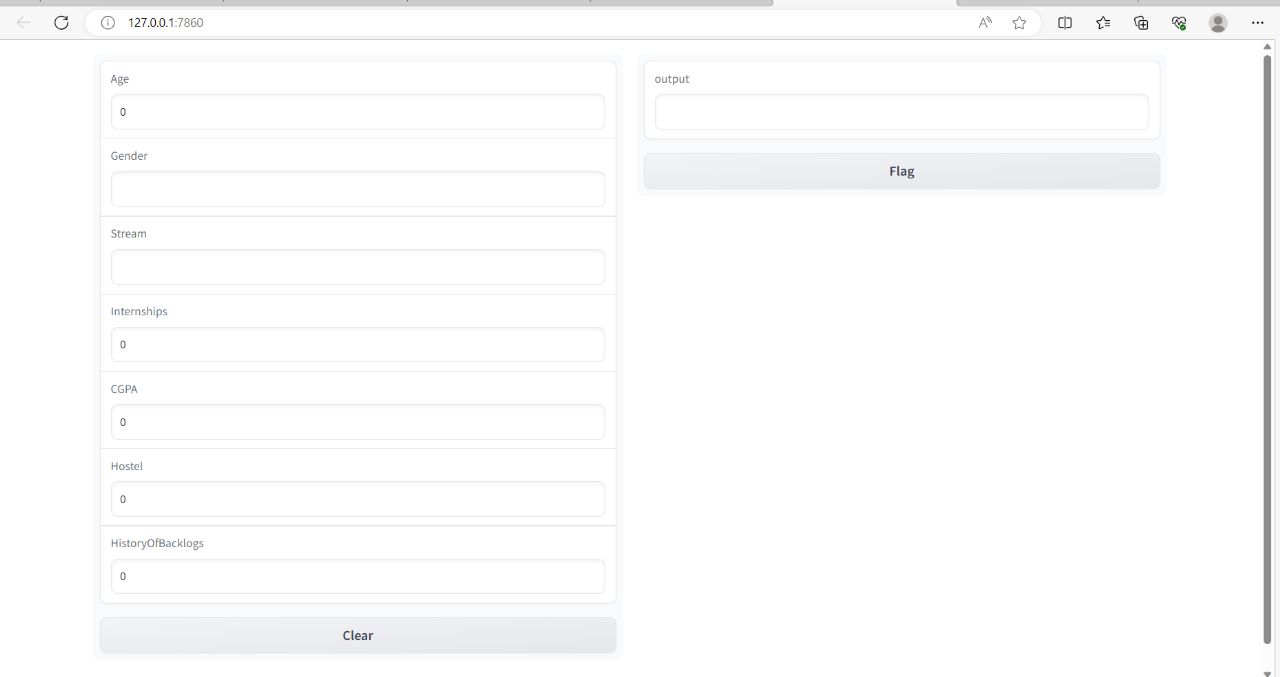
outputs=outputs,

live=True

**)**

gr\_interface.launch()

**OUTPUT:** **simple ui**



**Model prediction in UI**

