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 somuch 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
import tensorflow as tf
import pickle
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.model_selection import cross_val_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Read the Data

```
df=pd.read_csv("collegePlace.csv")
df
```

	Age	Gender	Stream	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot
0	22	Male	Electronics And Communication	1	8	1	1	1
1	21	Female	Computer Science	0	7	1	1	1
2	22	Female	Information Technology	1	6	0	0	1
3	21	Male	Information Technology	0	8	0	1	1
4	22	Male	Mechanical	0	8	1	0	1
...	...	...	...	...	...	...	...	...
2961	23	Male	Information Technology	0	7	0	0	0
2962	23	Male	Mechanical	1	7	1	0	0
2963	22	Male	Information Technology	1	7	0	0	0
2964	22	Male	Computer Science	1	7	0	0	0
2965	23	Male	Civil	0	8	0	0	1

2966 rows × 8 columns

Data Preparation

```
df.shape
```

(2966, 8)

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2966 entries, 0 to 2965
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Age                2966 non-null   int64
 1   Gender             2966 non-null   object
 2   Stream             2966 non-null   object
 3   Internships        2966 non-null   int64
 4   CGPA               2966 non-null   int64
 5   Hostel             2966 non-null   int64
 6   HistoryOfBacklogs  2966 non-null   int64
 7   PlacedOrNot        2966 non-null   int64
dtypes: int64(6), object(2)
memory usage: 185.5+ KB
```

 # Statistical Descriptions of the numerical values in the dataset
df.describe()

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot
count	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000	2966.000000
mean	21.485840	0.703641	7.073837	0.269049	0.192178	0.552596
std	1.324933	0.740197	0.967748	0.443540	0.394079	0.497310
min	19.000000	0.000000	5.000000	0.000000	0.000000	0.000000
25%	21.000000	0.000000	6.000000	0.000000	0.000000	0.000000
50%	21.000000	1.000000	7.000000	0.000000	0.000000	1.000000
75%	22.000000	1.000000	8.000000	1.000000	0.000000	1.000000
max	30.000000	3.000000	9.000000	1.000000	1.000000	1.000000

 # Getting to know the correlation between the target column and other features.
df.corr()['PlacedOrNot']

```
Age                 0.046943
Internships         0.179334
CGPA                0.588648
Hostel             -0.038182
HistoryOfBacklogs  -0.022337
PlacedOrNot         1.000000
Name: PlacedOrNot, dtype: float64
```

Preprocessing

df.isnull().sum()

```
Age                0
Gender             0
Stream             0
Internships        0
CGPA               0
Hostel             0
HistoryOfBacklogs  0
PlacedOrNot        0
dtype: int64
```

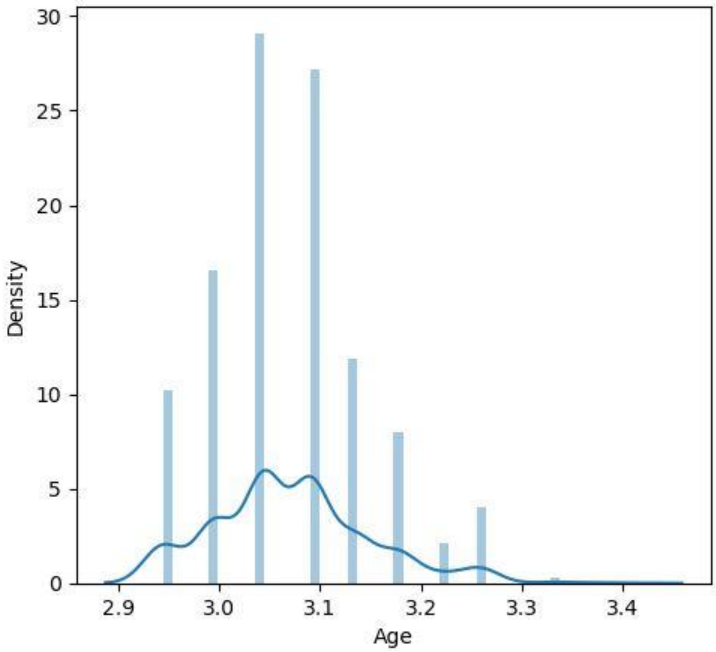
```
# duplicate rows
print(df.duplicated().sum())
#drop duplicates
df.drop_duplicates(inplace=True)
```

1829

```
# Check if the duplicate rows are removed
print(df.duplicated().sum())
```

0

```
def transformationplot(feature):
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    sns.distplot(feature)
transformationplot(np.log(df['Age']))
```



df.replace({"Male":1,"Female":0})

	Age	Gender	Stream	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot
0	22	1	Electronics And Communication	1	8	1	1	1
1	21	0	Computer Science	0	7	1	1	1
2	22	0	Information Technology	1	6	0	0	1
3	21	1	Information Technology	0	8	0	1	1
4	22	1	Mechanical	0	8	1	0	1
...	...	...	...	...	...	...	...	...
2946	23	1	Information Technology	1	7	1	1	0
2952	23	1	Mechanical	0	8	1	0	1
2954	23	0	Computer Science	1	8	0	1	1
2958	23	1	Computer Science	0	6	0	1	0
2960	23	1	Mechanical	1	7	1	0	0

1137 rows × 8 columns

Stream = df[["Stream"]]
Stream = pd.get_dummies(Stream, drop_first = **True**)
Stream.head()

	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Information Technology	Stream_Mechanical
0	0	0	1	0	0
1	1	0	0	0	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	0	1

pre_train=pd.concat([df, Stream], axis = 1)

pre_train.head()

	Age	Gender	Stream	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot	Stream_Computer Science	Stream_Electrical
0	22	Male	Electronics And Communication	1	8	1	1	1	0	0
1	21	Female	Computer Science	0	7	1	1	1	1	0
2	22	Female	Information Technology	1	6	0	0	1	0	0
3	21	Male	Information Technology	0	8	0	1	1	0	0
4	22	Male	Mechanical	0	8	1	0	1	0	0

```
pre_train=pd.concat([df, Stream], axis = 1)
pre_train.head()
```

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication
0	22	1	8	1	1	1	0	0	1
1	21	0	7	1	1	1	1	0	0
2	22	1	6	0	0	1	0	0	0
3	21	0	8	0	1	1	0	0	0
4	22	0	8	1	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
2946	23	1	7	1	1	0	0	0	0
2952	23	0	8	1	0	1	0	0	0
2954	23	1	8	0	1	1	1	0	0
2958	23	0	6	0	1	0	1	0	0
2960	23	1	7	1	0	0	0	0	0

1137 rows × 11 columns

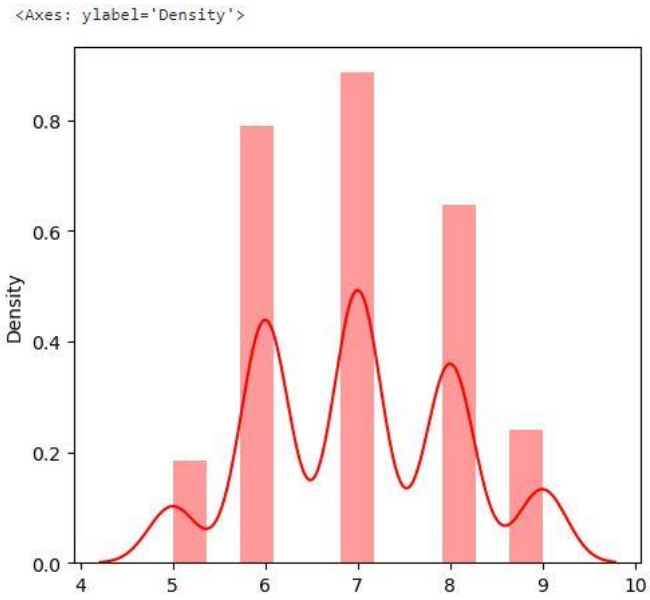

```
tol_df=pre_train.drop(["Gender",'Stream'],axis=1)
tol_df
```

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	PlacedOrNot	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication
0	22	1	8	1	1	1	0	0	1
1	21	0	7	1	1	1	1	0	0
2	22	1	6	0	0	1	0	0	0
3	21	0	8	0	1	1	0	0	0
4	22	0	8	1	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
2946	23	1	7	1	1	0	0	0	0
2952	23	0	8	1	0	1	0	0	0
2954	23	1	8	0	1	1	1	0	0
2958	23	0	6	0	1	0	1	0	0
2960	23	1	7	1	0	0	0	0	0

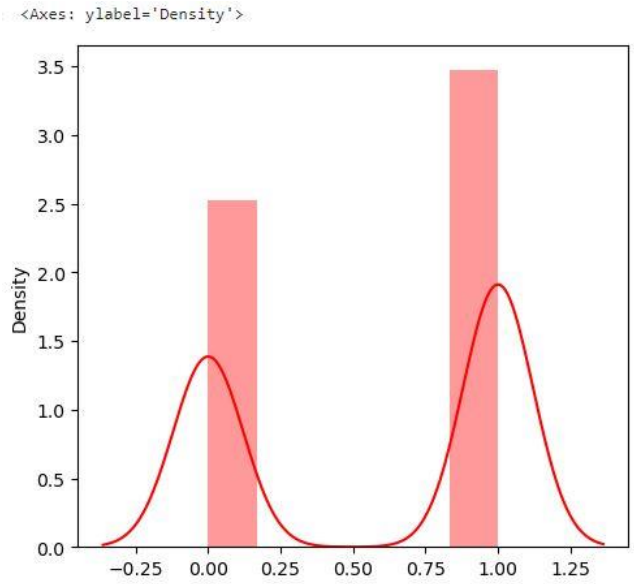
1137 rows × 11 columns

Univariate Analysis

```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(tol_df[['CGPA']],color='r')
```

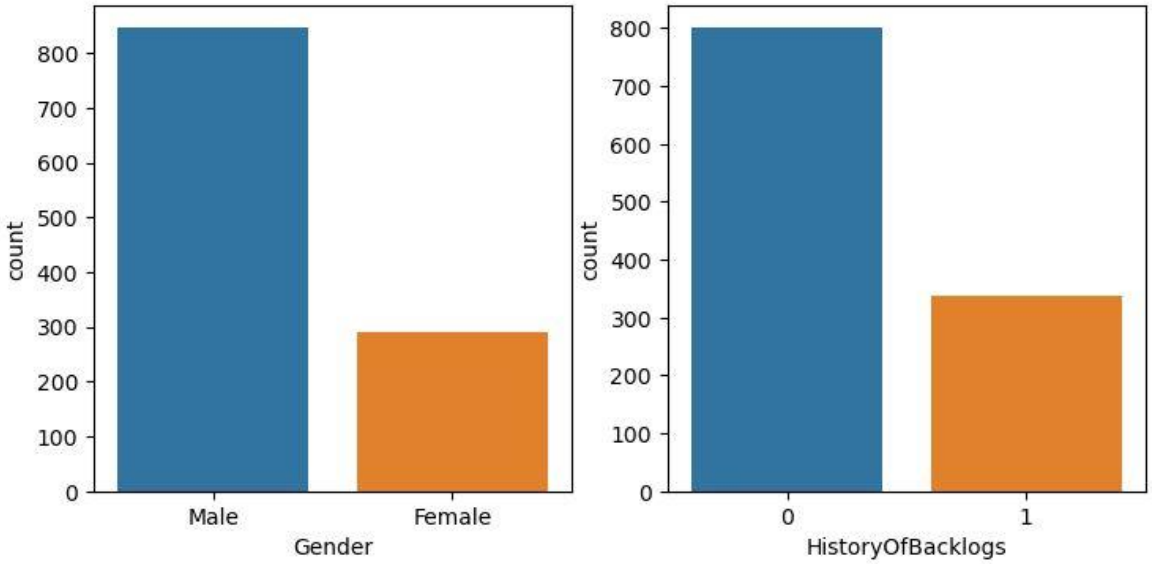



```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(tol_df[['PlacedOrNot']],color='r')
```



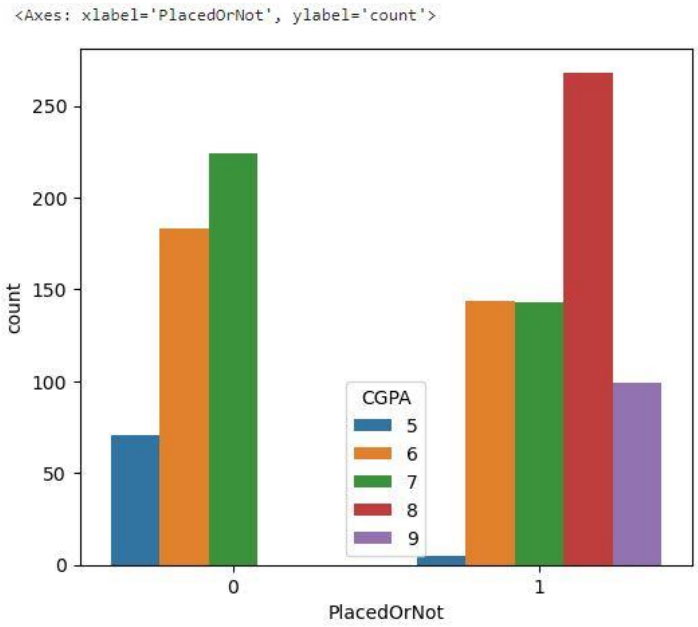
Bivariate Analysis

```
plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(x="Gender",data=df)
plt.subplot(1,4,2)
sns.countplot(x="HistoryOfBacklogs",data=df)
plt.show()
```

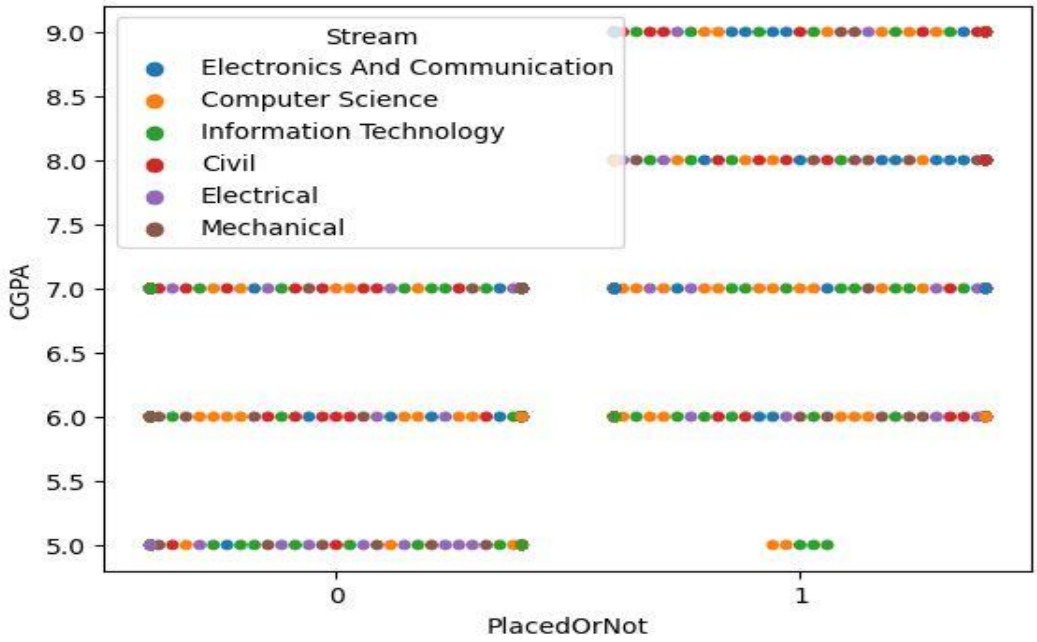


Multivariate Analysis

```
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(x=tol_df['PlacedOrNot'],hue=tol_df['CGPA'])
```



sns.swarmplot(x=tol_df['PlacedOrNot'],y=tol_df['CGPA'],hue=df['Stream'])



tol_df

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Informa Technol
0	22	1	8	1	1	0	0	1	
1	21	0	7	1	1	1	0	0	
2	22	1	6	0	0	0	0	0	
3	21	0	8	0	1	0	0	0	
4	22	0	8	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	
2946	23	1	7	1	1	0	0	0	
2952	23	0	8	1	0	0	0	0	
2954	23	1	8	0	1	1	0	0	
2958	23	0	6	0	1	1	0	0	
2960	23	1	7	1	0	0	0	0	

1137 rows × 10 columns

Splitting the data into train and test

x=tol_df.drop(['PlacedOrNot'],axis=1)

x

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Informa Technol
0	22	1	8	1	1	0	0	1	
1	21	0	7	1	1	1	0	0	
2	22	1	6	0	0	0	0	0	
3	21	0	8	0	1	0	0	0	
4	22	0	8	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	
2946	23	1	7	1	1	0	0	0	
2952	23	0	8	1	0	0	0	0	
2954	23	1	8	0	1	1	0	0	
2958	23	0	6	0	1	1	0	0	
2960	23	1	7	1	0	0	0	0	

1137 rows × 10 columns

y=tol_df['PlacedOrNot']
y

```
0       1
1       1
2       1
3       1
4       1
       ..
2946    0
2952    1
2954    1
2958    0
2960    0
Name: PlacedOrNot, Length: 1137, dtype: int64
```

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_stat
x_train

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Informa Technol
363	21	0	8	0	1	1	0	0	
1763	23	1	6	0	0	0	0	0	
1948	20	0	7	0	1	0	0	0	
1618	22	2	7	1	0	0	0	0	
1780	22	1	7	1	0	0	0	1	
...	...	...	...	...	...	...	...	...	
827	21	0	7	0	0	0	0	0	
1501	24	1	6	1	1	1	0	0	
1309	26	1	6	0	1	0	0	0	
142	24	3	7	0	0	0	0	0	
1590	22	2	6	0	0	0	0	0	


```
y=tol_df['PlacedOrNot']
y
```

```
0       1
1       1
2       1
3       1
4       1
       ..
2946    0
2952    1
2954    1
2958    0
2960    0
Name: PlacedOrNot, Length: 1137, dtype: int64
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_stat
x_train
```

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Informa Technol
363	21	0	8	0	1	1	0	0	
1763	23	1	6	0	0	0	0	0	
1948	20	0	7	0	1	0	0	0	
1618	22	2	7	1	0	0	0	0	
1780	22	1	7	1	0	0	0	1	
...	...	...	...	...	...	...	...	...	
827	21	0	7	0	0	0	0	0	
1501	24	1	6	1	1	1	0	0	
1309	26	1	6	0	1	0	0	0	
142	24	3	7	0	0	0	0	0	
1590	22	2	6	0	0	0	0	0	

909 rows × 10 columns

y_train

```
363     1
1763    0
1948    1
1618    0
1780    0
       ..
827     1
1501    1
1309    1
142     1
1590    0
Name: PlacedOrNot, Length: 909, dtype: int64
```

x_test

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Informa Technol
1966	19	0	8	0	1	0	0	0	
1591	24	0	6	0	0	0	0	0	
108	24	1	7	0	1	0	0	0	
2651	20	0	9	1	1	0	0	0	
751	23	1	6	1	1	0	0	1	
...	...	...	...	...	...	...	...	...	
86	21	0	8	0	0	1	0	0	
2173	20	0	9	1	1	1	0	0	
2082	19	0	8	1	0	0	0	0	
684	22	2	6	1	0	0	0	0	
2937	23	0	6	0	0	0	0	0	

228 rows × 10 columns

y_test

```
1966    1
1591    1
108     1
2651    1
751     0
       ..
86      1
2173    1
2082    1
684     1
2937    0
Name: PlacedOrNot, Length: 228, dtype: int64
```

Model Building

SVM

```
from sklearn.svm import SVC
svm=SVC(kernel='linear')
svm.fit(x_train,y_train)

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",train_accuracy)
print("accuracy on testing data",test_accuracy)
```

```
accuracy on training data 0.7370737073707371
accuracy on testing data 0.7105263157894737
```

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)
```

```
k= 21
accuracy= 79.82456140350878
```

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(10,activation='relu',input_dim=10))
classifier.add(Dropout(0.50))
```

```
classifier.add(Dense(16,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)
```

```
Epoch 1/100
46/46 [==============================] - 5s 5ms/step - loss: 0.8619 - accuracy: 0.5149
Epoch 2/100
46/46 [==============================] - 0s 5ms/step - loss: 0.7768 - accuracy: 0.5138
Epoch 3/100
46/46 [==============================] - 0s 6ms/step - loss: 0.7187 - accuracy: 0.5094
Epoch 4/100
46/46 [==============================] - 0s 5ms/step - loss: 0.7025 - accuracy: 0.5171
Epoch 5/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6862 - accuracy: 0.5391
Epoch 6/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6693 - accuracy: 0.5567
Epoch 7/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6622 - accuracy: 0.5567
Epoch 8/100
46/46 [==============================] - 0s 5ms/step - loss: 0.6723 - accuracy: 0.5567
Epoch 9/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6650 - accuracy: 0.5886
Epoch 10/100
46/46 [==============================] - 0s 4ms/step - loss: 0.6568 - accuracy: 0.5677
Epoch 11/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6628 - accuracy: 0.5666
Epoch 12/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6508 - accuracy: 0.5710
Epoch 13/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6593 - accuracy: 0.5490
Epoch 14/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6527 - accuracy: 0.5732
Epoch 15/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6545 - accuracy: 0.5666
Epoch 16/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6536 - accuracy: 0.5688
Epoch 17/100
46/46 [==============================] - 0s 9ms/step - loss: 0.6392 - accuracy: 0.5732
Epoch 18/100
46/46 [==============================] - 0s 9ms/step - loss: 0.6471 - accuracy: 0.5699
Epoch 19/100
46/46 [==============================] - 0s 11ms/step - loss: 0.6457 - accuracy: 0.5743
Epoch 20/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6427 - accuracy: 0.5787
Epoch 21/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6368 - accuracy: 0.5743
Epoch 22/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6454 - accuracy: 0.5721
Epoch 23/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6284 - accuracy: 0.5688
Epoch 24/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6281 - accuracy: 0.5688
Epoch 25/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6300 - accuracy: 0.5732
```

```
Epoch 26/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6293 - accuracy: 0.5743
Epoch 27/100
46/46 [==============================] - 0s 5ms/step - loss: 0.6285 - accuracy: 0.5765
Epoch 28/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6142 - accuracy: 0.5787
Epoch 29/100
46/46 [==============================] - 0s 7ms/step - loss: 0.6213 - accuracy: 0.5721
Epoch 30/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6179 - accuracy: 0.5743
Epoch 31/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6053 - accuracy: 0.5787
Epoch 32/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6216 - accuracy: 0.6238
Epoch 33/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6171 - accuracy: 0.6117
Epoch 34/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6116 - accuracy: 0.6271
Epoch 35/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5994 - accuracy: 0.6568
Epoch 36/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6090 - accuracy: 0.6216
Epoch 37/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6155 - accuracy: 0.6271
Epoch 38/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6099 - accuracy: 0.6458
Epoch 39/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6002 - accuracy: 0.6425
Epoch 40/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6082 - accuracy: 0.6260
Epoch 41/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5906 - accuracy: 0.6623
Epoch 42/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5978 - accuracy: 0.6469
Epoch 43/100
46/46 [==============================] - 0s 4ms/step - loss: 0.6093 - accuracy: 0.6315
Epoch 44/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5997 - accuracy: 0.6392
Epoch 45/100
46/46 [==============================] - 0s 8ms/step - loss: 0.6021 - accuracy: 0.6348
Epoch 46/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5950 - accuracy: 0.6634
Epoch 47/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6077 - accuracy: 0.6392
Epoch 48/100
46/46 [==============================] - 0s 6ms/step - loss: 0.6000 - accuracy: 0.6348
Epoch 49/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5878 - accuracy: 0.6414
Epoch 50/100
46/46 [==============================] - 0s 4ms/step - loss: 0.5996 - accuracy: 0.6513
Epoch 51/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5991 - accuracy: 0.6447
Epoch 52/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5851 - accuracy: 0.6403
Epoch 53/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5866 - accuracy: 0.6381
Epoch 54/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5830 - accuracy: 0.6700
Epoch 55/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5903 - accuracy: 0.6436
```

```
Epoch 56/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5814 - accuracy: 0.6425
Epoch 57/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5787 - accuracy: 0.6568
Epoch 58/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5690 - accuracy: 0.6700
Epoch 59/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5860 - accuracy: 0.6557
Epoch 60/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5800 - accuracy: 0.6513
Epoch 61/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5827 - accuracy: 0.6535
Epoch 62/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5693 - accuracy: 0.6524
Epoch 63/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5842 - accuracy: 0.6601
Epoch 64/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5787 - accuracy: 0.6513
Epoch 65/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5750 - accuracy: 0.6634
Epoch 66/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5751 - accuracy: 0.6645
Epoch 67/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5665 - accuracy: 0.6744
Epoch 68/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5675 - accuracy: 0.6667
Epoch 69/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5636 - accuracy: 0.6722
Epoch 70/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5775 - accuracy: 0.6535
Epoch 71/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5753 - accuracy: 0.6458
Epoch 72/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5708 - accuracy: 0.6524
Epoch 73/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5679 - accuracy: 0.6744
Epoch 74/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5833 - accuracy: 0.6535
Epoch 75/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5700 - accuracy: 0.6645
Epoch 76/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5637 - accuracy: 0.6876
Epoch 77/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5697 - accuracy: 0.6601
Epoch 78/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5704 - accuracy: 0.6711
Epoch 79/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5623 - accuracy: 0.6744
Epoch 80/100
46/46 [==============================] - 0s 4ms/step - loss: 0.5559 - accuracy: 0.6821
Epoch 81/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5776 - accuracy: 0.6601
Epoch 82/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5612 - accuracy: 0.6568
Epoch 83/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5647 - accuracy: 0.6799
Epoch 84/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5569 - accuracy: 0.6623
Epoch 85/100
46/46 [==============================] - 0s 10ms/step - loss: 0.5590 - accuracy: 0.6722
```

```
Epoch 86/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5678 - accuracy: 0.6612
Epoch 87/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5661 - accuracy: 0.6689
Epoch 88/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5710 - accuracy: 0.6601
Epoch 89/100
46/46 [==============================] - 0s 5ms/step - loss: 0.5693 - accuracy: 0.6425
Epoch 90/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5549 - accuracy: 0.6777
Epoch 91/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5580 - accuracy: 0.6667
Epoch 92/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5624 - accuracy: 0.6623
Epoch 93/100
46/46 [==============================] - 0s 8ms/step - loss: 0.5698 - accuracy: 0.6601
Epoch 94/100
46/46 [==============================] - 0s 8ms/step - loss: 0.5578 - accuracy: 0.6766
Epoch 95/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5562 - accuracy: 0.6766
Epoch 96/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5686 - accuracy: 0.6656
Epoch 97/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5687 - accuracy: 0.6623
Epoch 98/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5733 - accuracy: 0.6590
Epoch 99/100
46/46 [==============================] - 0s 6ms/step - loss: 0.5659 - accuracy: 0.6491
Epoch 100/100
46/46 [==============================] - 0s 7ms/step - loss: 0.5648 - accuracy: 0.6678

<keras.src.callbacks.History at 0x21589e044d0>
```

Model Selecting using Pickle

```
y_pred=classifier.predict(x_test)

8/8 [==============================] - 1s 3ms/step


y_preds=[]
for i in y_pred:
    if i >0.5:
        y_preds.append(1)
    else:
        y_preds.append(0)


y_preds
[1,
 0,
 0,
 1,
 0,
 0,
 0,
 1,
 1,
```

1,
1,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
1,
0,
1,
0,
1,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
0,
1,
1,
0,
1,
0,

1,
0,
0,
0,
0,
0,
1,
0,
0,
1,
1,
1,
1,
1,
0,
0,
1,
0,
1,
0,
1,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,
0,
1,
1,
0,
1,
0,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
1,
0,
0,
0,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,

1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
1,
1,
0,
0,
1,
0,
1,
0,
0,
1,
1,
0,
0,
1,
1,
0,
1,
1,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
1,
0,
1,

1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
0,
0,
1,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
1,
1,
0,
0,
1,
1,
1,
0,
0]

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_preds))
```

```
              precision    recall  f1-score   support

           0       0.59      0.84      0.69        92
           1       0.85      0.61      0.71       136

    accuracy                           0.70       228
   macro avg       0.72      0.72      0.70       228
weighted avg       0.74      0.70      0.70       228
```

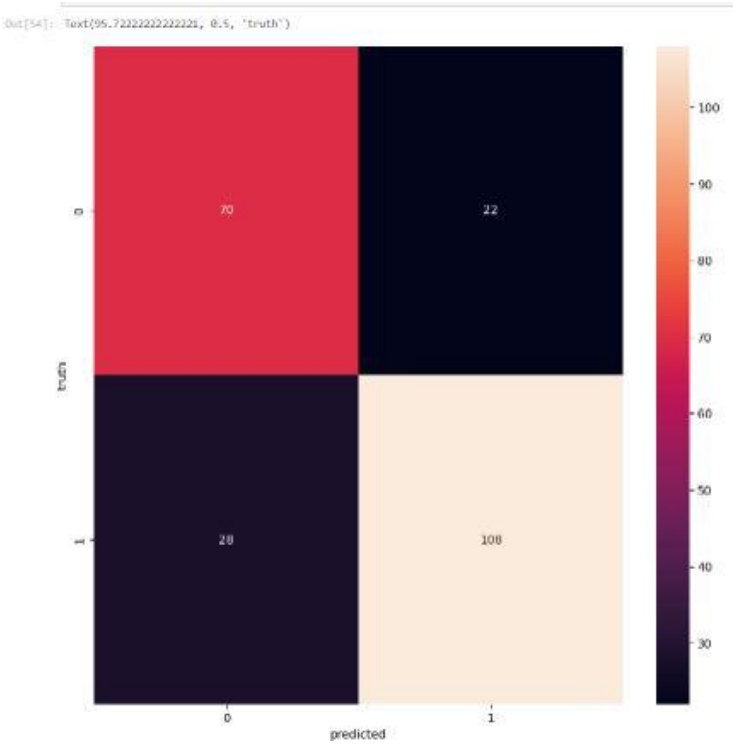
```
model =KNeighborsClassifier(n_neighbors=31)
model.fit(x_train,y_train)
y_predict=model.predict(x_test)
model.score(x_test,y_test)
```

0.7807017543859649

```
import pickle
with open('model_pickle_modified','wb') as f:
    pickle.dump(model,f)

with open('model_pickle_modified','rb') as f:
    mp=pickle.load(f)

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

```
array([[ 70,  22],
       [ 28, 108]], dtype=int64)
```

```
import seaborn as sb

plt.figure(figsize=(10,10))
sb.heatmap(cm,fmt='d',annot=True)
plt.xlabel('predicted')
plt.ylabel('truth')
```




```
x_test.columns
```

```
Index(['Age', 'Internships', 'CGPA', 'Hostel', 'HistoryOfBacklogs',
       'Stream_Computer Science', 'Stream_Electrical',
       'Stream_Electronics And Communication', 'Stream_Information Technology',
       'Stream_Mechanical'],
      dtype='object')
```

```
df["Stream"].unique()
```

```
array(['Electronics And Communication', 'Computer Science',
       'Information Technology', 'Mechanical', 'Electrical', 'Civil'],
      dtype=object)
```

```
mp.predict(x_test)
```

```
array([1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 0], dtype=int64)
```

x_test

	Age	Internships	CGPA	Hostel	HistoryOfBacklogs	Stream_Computer Science	Stream_Electrical	Stream_Electronics And Communication	Stream_Information Technology
1966	19	0	8	0	1	0	0	0	0
1591	24	0	6	0	0	0	0	0	1
108	24	1	7	0	1	0	0	0	0
2651	20	0	9	1	1	0	0	0	0
751	23	1	6	1	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...
86	21	0	8	0	0	1	0	0	0
2173	20	0	9	1	1	1	0	0	0
2082	19	0	8	1	0	0	0	0	0
684	22	2	6	1	0	0	0	0	0
2937	23	0	6	0	0	0	0	0	0

228 rows × 10 columns

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 81% 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.

Flask Implementation:

util.py

```
import pickle
import numpy as np
__model=None
def prediction(Internships,Age, CGPA, hostel, hob, branch):
    if (branch == 'Computer_Science'):
        Computer_Science = 1
        Electronics_And_Communication = 0
        Information_Technology = 0
        Mechanical = 0
        Electrical = 0
        Civil = 0

    elif (branch == 'Electronics_And_Communication'):
        Computer_Science = 0
        Electronics_And_Communication = 1
        Computer_Science = 1
        Information_Technology = 0
        Mechanical = 0
        Electrical = 0
        Civil = 0
```

```
elif (branch == 'Information_Technology'):
    Computer_Science = 0
    Electronics_And_Communication = 0
    Information_Technology = 1
    Mechanical = 0
    Electrical = 0
    Civil = 0

elif (branch == 'Mechanical'):
    Computer_Science = 0
    Electronics_And_Communication = 0
    Information_Technology = 0
    Mechanical = 1
    Electrical = 0
    Civil = 0

elif (branch == 'Electrical'):
    Computer_Science = 0
    Electronics_And_Communication =0
    Information_Technology = 0
    Mechanical = 0
    Electrical = 1
    Civil = 0

else:
    Computer_Science = 0
    Electronics_And_Communication = 0
    Information_Technology = 0
    Mechanical = 0
```

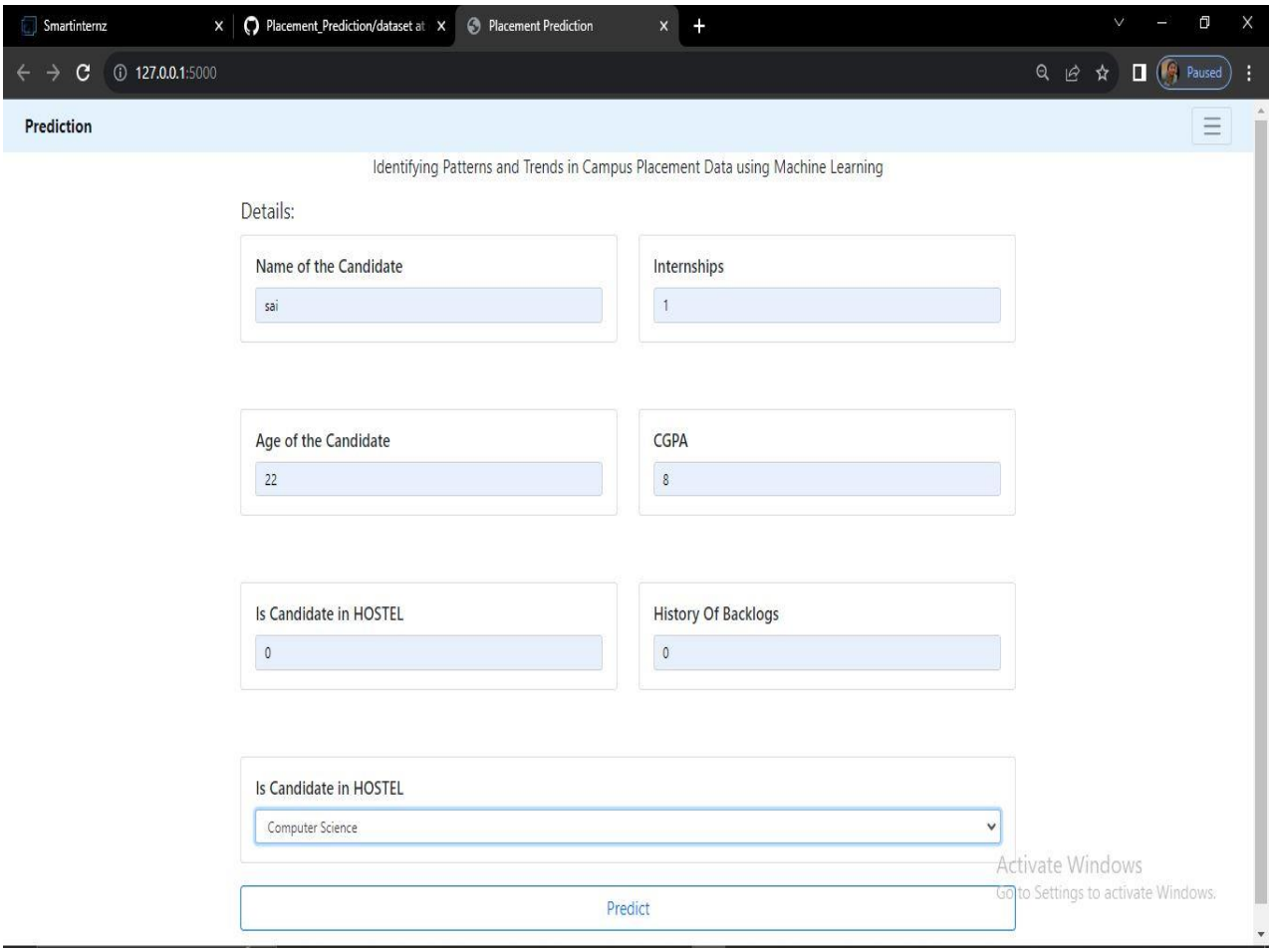
```
        Electrical = 0
        Civil = 1

branchs=[Electronics_And_Communication,Computer_Science,Information_Technology,Mechanical,Electrical]
    x=[]
    x.append(Age)
    x.append(Internships)
    x.append(CGPA)
    x.append(hostel)
    x.append(hob)
    for i in branchs:
        x.append(i)
    print(x)
    return (__model.predict([x])[0])


def load_artifacts():
    global __model
    with open("model_pickle_modified",'rb') as f:
        __model=pickle.load(f)
    print("arti facts are loaded successfully ...")
if __name__ =='__main__':
    load_artifacts()
    print(__model.predict([[22, 1, 7, 0, 1, 0, 0, 0, 0, 0]]))
```

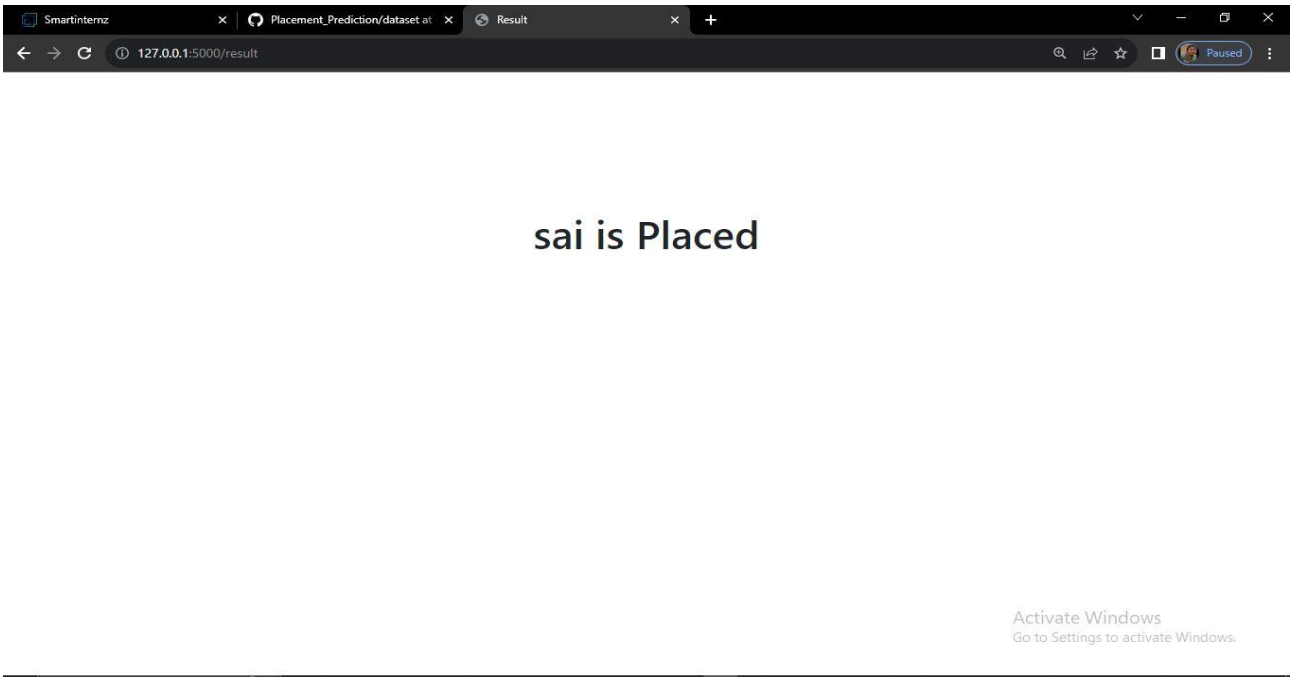
app.py

```
import util
#import numpy as np
from flask import Flask, render_template, request
app = Flask(__name__)
global result
@app.route('/')
def home():
  return render_template('index.html')
@app.route('/result',methods = ['POST', 'GET'])
def result():
  if request.method == 'POST':
      name=str(request.form['namec'])
      Internships=int(request.form['Internships'])
      Age=int(request.form['Age'])
      CGPA=int(request.form['CGPA'])
      hostel=int(request.form['hostel'])
      hob=int(request.form['hob'])
      branch=str(request.form['branch'])
      print(name,Internships,Age,CGPA,hostel,hob,branch)
      ans=util.prediction(Internships,Age, CGPA, hostel, hob, branch)

      print(ans)
      if int(ans)==0:
         s='Not Placed'
      else:
         s='Placed"'
      return render_template('result.html',name=name, ans=s)

if __name__ == '__main__':
```

```
util.load_artifacts()
app.run()
```

OUTPUT:


Smartinternz
Placement_Prediction/dataset at
Placement Prediction
127.0.0.1:5000
Paused
Prediction
Identifying Patterns and Trends in Campus Placement Data using Machine Learning
Details:
Name of the Candidate
sai
Internships
1
Age of the Candidate
22
CGPA
8
Is Candidate in HOSTEL
0
History Of Backlogs
0
Is Candidate in HOSTEL
Computer Science
Activate Windows
Go to Settings to activate Windows.
Predict


Smartinternz
Placement_Prediction/dataset at
Result
127.0.0.1:5000/result
Paused
sai is Placed
Activate Windows
Go to Settings to activate Windows.