IBM HACK CHALLENGE 2023

CAMPUS PLACEMENT PREDICTION USING MACHINE LEARNING

APPLICATION_ID: SPS_CH_APL_20230020499

Team Name	**:**	**Rummorboy**
Team Size & Members Name	**:**	**1** **Shahib**
Bussiness Challenge	**:**	**Identifying Patterns and Trends in Campus Placement Data using Machine Learning**

IGC- Indo Global College of Engineering, Abhipur Village, Mohali, New Chandigarh, Punjab-140109.

INDEX

CAMPUS PLACEMENT PREDICTION

1. INDRODUCTION

OVERVIEW

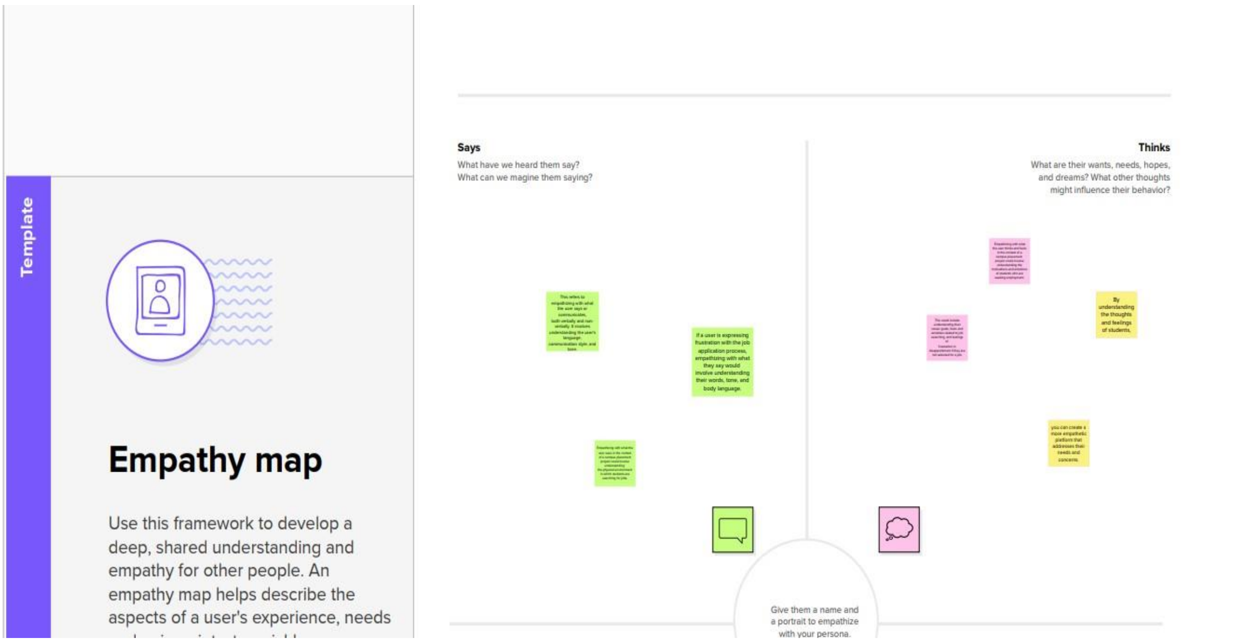
Campus placement is a crucial aspect of the education system, as it connects students with job opportunities and helps them transition into their careers. By participating in campus placement programs, students can interact with potential employers, gain exposure to different industries, and secure job offers before they even graduate. For universities, campus placement is also important for attracting and retaining students. If a university has a strong track record of placing students in top companies and industries, it can be a major selling point for prospective students and their families. To facilitate campus placement, universities and companies often work together to organize job fairs, interviews, and other events. However, the process of matching students with job opportunities can be complex and time-consuming, which is where machine learning can come in. By using machine learning algorithms like the random forest classifier, we can analyze data on student performance, interests, and other factors to predict which students are most likely to be placed in jobs after graduation. This can help universities and companies optimize their campus placement strategies, leading to better outcomes for students and employers alike.

PURPOSE

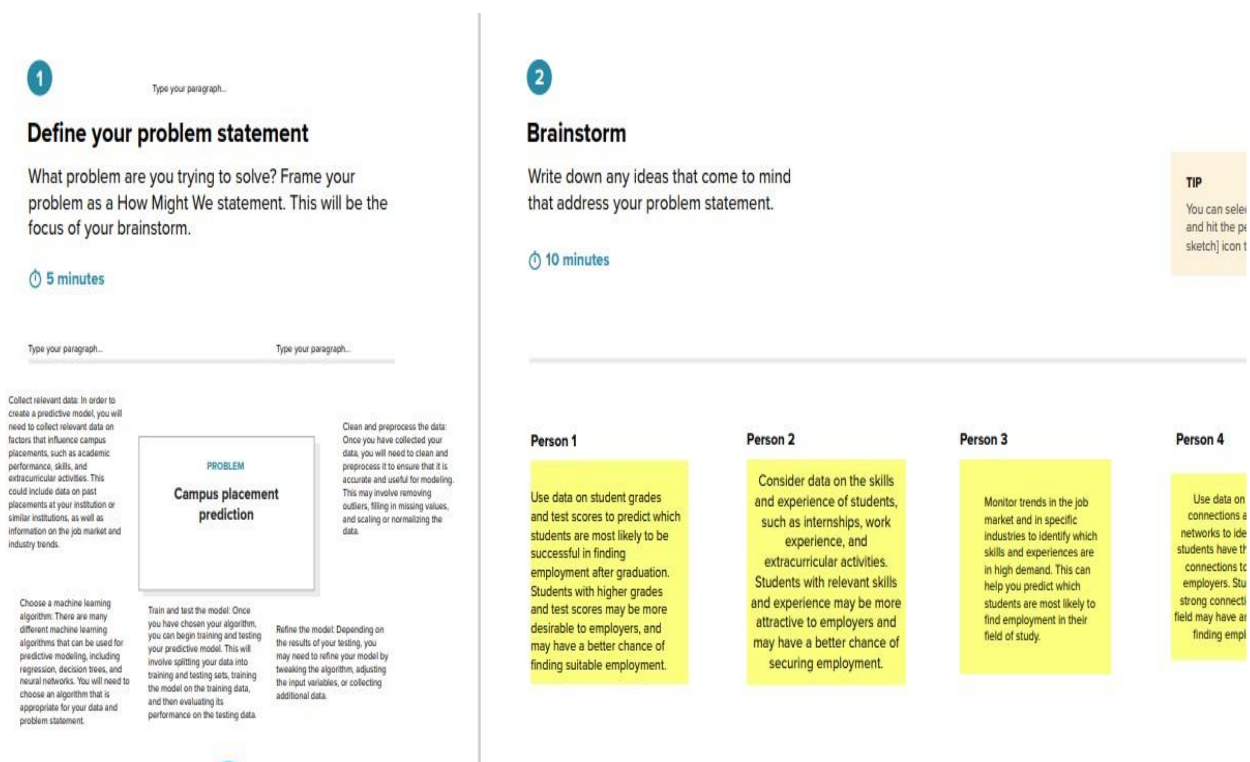
The purpose of your campus placement prediction project is to develop a machine learning model that can accurately predict whether a student will be placed in a job after completing their studies. This model can be used to optimize campus placement strategies for universities and employers, leading to better outcomes for both parties. There are several benefits to developing an accurate campus placement prediction model. For universities, a better understanding of which students are most likely to be placed in jobs can help them tailor their programs and career services to better meet student needs. It can also help universities attract and retain students by demonstrating their commitment to providing practical career preparation. For employers, a more accurate campus placement model can help them identify top talent early in the recruitment process, reducing the time and resources required to fill open positions. It can also help employers build stronger relationships with universities by participating in campus placement programs and offering internships and other opportunities to students. Overall, the purpose of your project is to use machine learning to solve an important real-world problem and improve the campus placement process for both students and employers. By developing a high-quality prediction model, you can help optimize campus placement strategies and improve outcomes for everyone involved.

2. PROBLEM DEFINITION & DESIGN THINKING

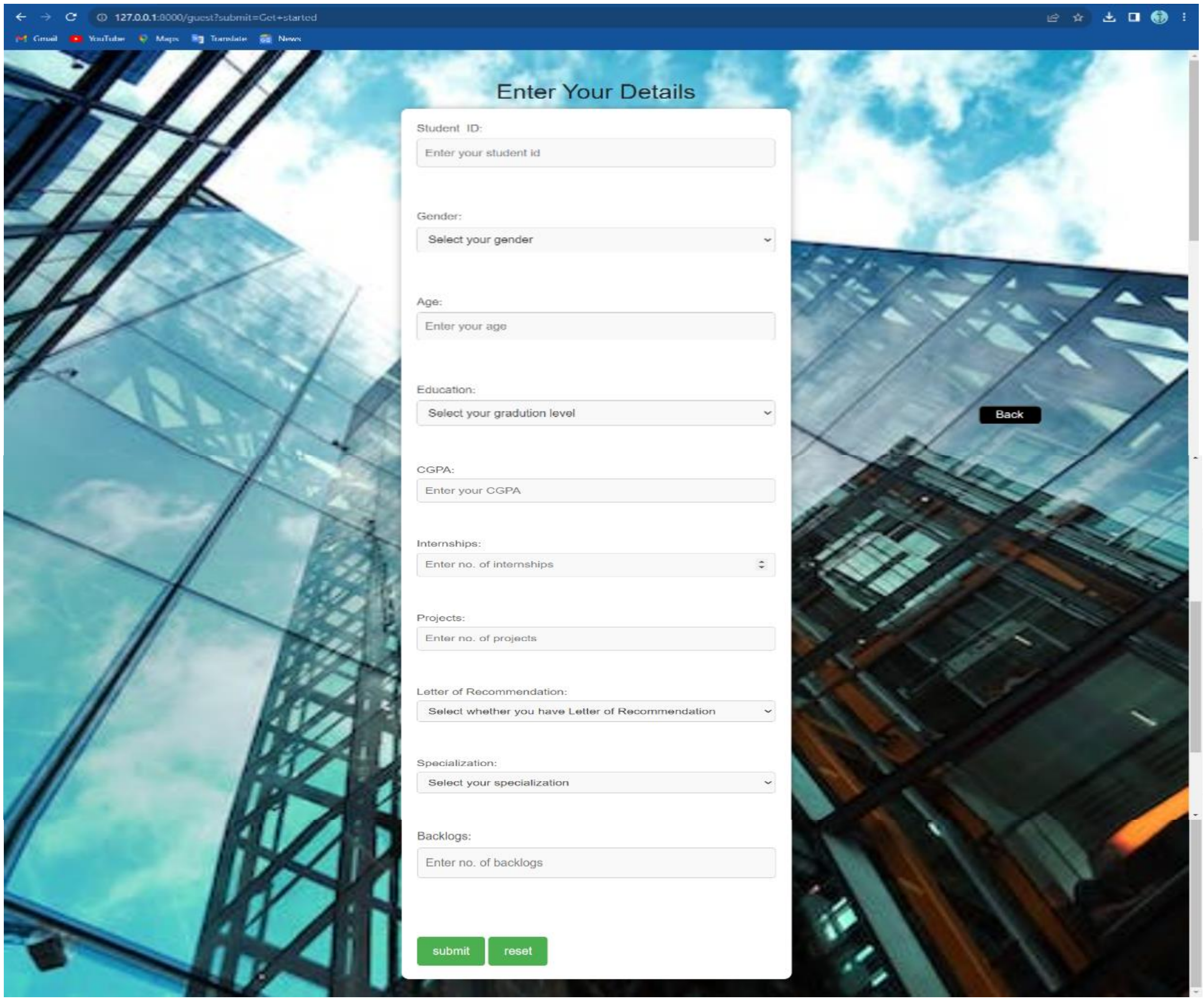
EMPATHY MAP



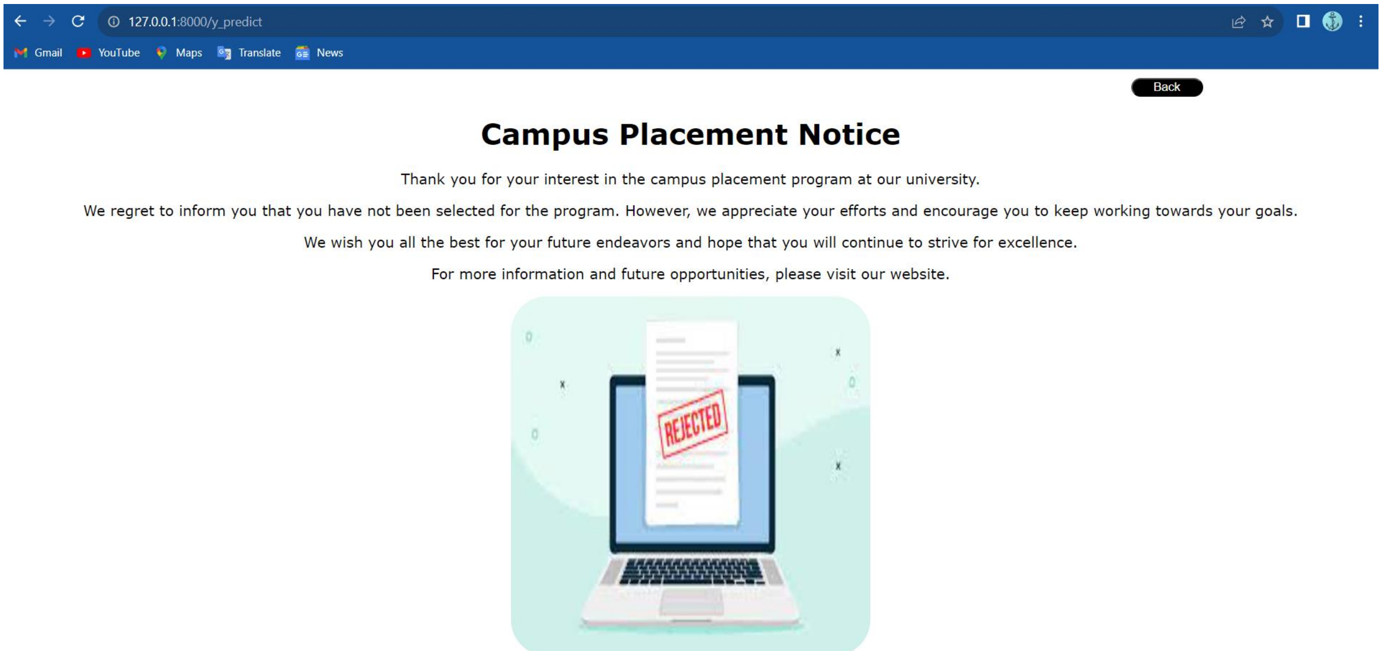
IDEATION & BRAINSTORMING MAP



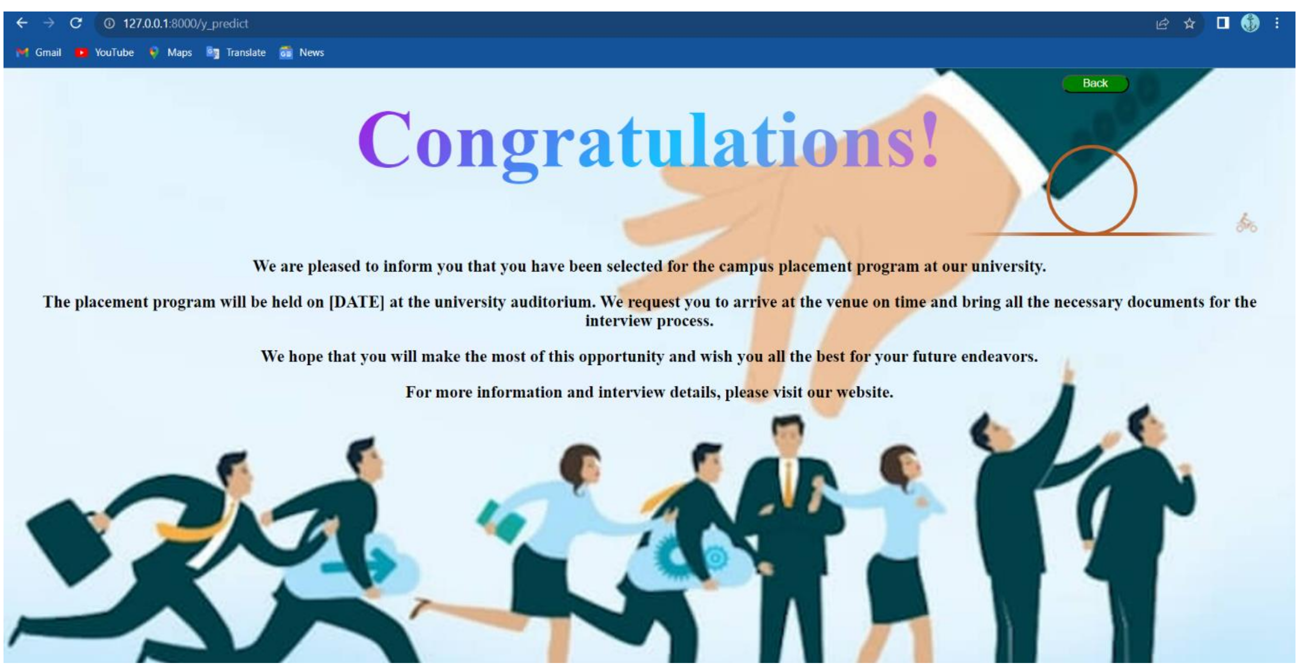
3. RESULT


Enter Your Details
Student ID:
Enter your student id
Gender:
Select your gender
Age:
Enter your age
Education:
Select your gradution level
CGPA:
Enter your CGPA
Internships:
Enter no. of internships
Projects:
Enter no. of projects
Letter of Recommendation:
Select whether you have Letter of Recommendation
Specialization:
Select your specialization
Backlogs:
Enter no. of backlogs
submit
reset
Back

STUDENT NOT PLACED OUTPUT



STUDENT PLACED OUTPUT



4. ADAVANTAGES AND DISADVANTAGES

ADVANTAGES

Increased accuracy: Machine learning algorithms like the random forest classifier can analyze large amounts of data and identify complex patterns that may not be apparent through traditional statistical methods. This can lead to more accurate predictions of which students are most likely to be placed in jobs after graduation. Optimization of resources: By accurately predicting which students are most likely to be placed in jobs, universities and employers can focus their resources on the students who are most likely to benefit from them. This can lead to more efficient use of time, money, and other resources. Improved decision-making: The insights provided by a machine learning model can help universities and employers make better decisions about which programs and strategies to pursue. For example, if the model shows that students with certain academic or extracurricular backgrounds are more likely to be placed in jobs, universities can focus on developing those programs and activities. Enhanced student outcomes: By improving the campus placement process, machine learning models can help more students secure jobs after graduation, leading to better outcomes for those students and their families. This can also help universities attract and retain students by demonstrating their commitment to providing practical career preparation.

DISADVANTAGES

Dependence on data quality and availability: The accuracy of the prediction model depends heavily on the quality and quantity of the data used to train it. If the data is incomplete or inaccurate, the model's predictions may not be reliable. Potential for bias in the data or model: If the data used to train the model contains bias or discriminatory factors, the model may perpetuate those biases and lead to unfair outcomes for certain groups of students. It's important to carefully evaluate and address any potential biases in the data and model. Complexity of the machine learning algorithms and potential difficulty in interpretation: Machine learning models can be complex and difficult to interpret, especially for stakeholders who are not familiar with the underlying algorithms and techniques. It's important to communicate the model's findings and limitations clearly and transparently.

Need for ongoing updates and maintenance of the model: Machine learning models require ongoing updates and maintenance to remain accurate and relevant. This can be timeconsuming and resource-intensive. Possible resistance or skepticism from stakeholders who are unfamiliar with machine learning and predictive modeling techniques: Some stakeholders may be skeptical of the accuracy and validity of machine learning models, especially if they are not familiar with the underlying techniques. It's important to communicate the benefits and limitations of the model clearly and transparently to build trust and support among stakeholders.

5. APPLICATIONS

The model can be used to provide personalized career counseling to students, based on their individual strengths and weaknesses. This can help students make more informed decisions about their future career paths. Universities and job placement agencies can use the model to optimize their resources by focusing on the students who are most likely to be placed in jobs after graduation. This can lead to more efficient use of time, money, and other resources. The model can help universities plan their curricula to better align with the needs of the job market. By identifying the skills and experiences that are most valued by employers, universities can develop programs that prepare students for the most in-demand jobs. The model can be used by employers to identify the most promising candidates for their job openings. This can save time and resources by allowing employers to focus on the candidates who are most likely to succeed in their organizations. The insights provided by the model can be used by policymakers to make decisions about funding and support for education and job training programs. By identifying the factors that lead to successful job placement, policymakers can develop policies that better support students and job seekers.

6. CONCLUSION

In conclusion, the development of a campus placement prediction model using machine learning algorithms like random forest classifier has many potential benefits for students, universities, and employers. By analyzing data on student characteristics and job placement outcomes, the model can accurately predict which students are most likely to be placed in jobs after graduation. This can lead to more efficient use of resources, better decisionmaking, and improved career outcomes for students. While there are potential challenges and drawbacks to developing such a model, including data quality and bias concerns and the complexity of the underlying algorithms, these challenges can be mitigated through careful data selection and evaluation, transparency in the model's findings and limitations, and ongoing updates and maintenance of the model. Overall, the campus placement prediction model has a wide range of practical applications in the education and job sectors, from career counseling and curriculum planning to employer recruitment and policymaking. By leveraging the power of machine learning algorithms, we can improve the campus placement process and help students achieve their career goals. One additional point to consider is the potential for scalability of the campus placement prediction model. Once developed and validated, the model can be applied to a larger dataset of students and job placement outcomes, which can help to refine and improve its accuracy. This scalability can also allow for the model to be applied across different universities and geographic regions, which can help to identify broader trends and insights into the job market. Furthermore, the scalability of the model can enable universities and employers to make better-informed decisions about where to allocate their resources, leading to more efficient and effective campus placement processes.

7. FUTURE SCOPE

The development of a campus placement prediction model using machine learning algorithms like random forest classifier has a significant future scope in the education and job sectors. The campus placement prediction model can be integrated with other predictive models, such as models that predict student retention and graduation rates. This can provide a more comprehensive view of student outcomes and help universities make more informed decisions about resource allocation and program development. The model can be expanded to include new variables and data sources, such as social media data and job market trends. This can improve the accuracy and relevance of the model's predictions and provide new insights into the factors that influence job placement outcomes. Explainable AI techniques can be used to make the model more transparent and interpretable. This can help to build trust and support among stakeholders and enable better decision-making based on the model's findings. The machine learning techniques used in the campus placement prediction model can be applied to other domains, such as healthcare and finance, to predict outcomes and inform decision-making. Overall, the future scope of the campus placement prediction model is vast, and there are many potential areas for research and development. By leveraging the power of machine learning algorithms and integrating new data sources and techniques, we can improve the accuracy and relevance of the model's predictions and help students achieve their career goals.

8. APPENDIX

Ipynb file :

```
## Importing Required Libraries Libraries

import numpy as np
import pandas as pd
import matplotlib,pyplot aas plt
import  seaborn as sns
```

```
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score,
recall_score, f1_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore', category=UserWarning)
import pickle
import joblib

## Creating Dataset of Campus Placement
np.random.seed(42)

num_samples = 1650

data = {
    'Student_ID': np.arange(20001, 20001 + num_samples),
    'Gender': np.random.choice(['Male', 'Female'], size=num_samples),
    'Age': np.random.randint(21, 31, size=num_samples),
    'Education': np.random.choice(['Bachelor', 'Master'], size=num_samples),
    'CGPA': np.round(np.random.uniform(5, 10, size=num_samples), 2),
    'Internships': np.random.randint(0, 5, size=num_samples),
    'Year': np.random.randint(2018, 2024, size=num_samples),
    'Hostel': np.random.choice(['Opted', 'Not Opted'], size=num_samples),
    'Projects' : np.random.randint(0, 6, size=num_samples),
    'Letter_of_Recommendation': np.random.choice(['Yes', 'No'], size=num_samples),
    'Specialization': np.random.choice(['Computer Science', 'Information Technology', 'Electrical', 'Electronics',
'Mechanical', 'Civil'], size=num_samples, p=[0.30, 0.20, 0.18, 0.17, 0.10, 0.05]),
    'On/Off Campus': np.random.choice(['On Campus', 'Off Campus'], size=num_samples),
    'Package(LPA)': np.random.randint(200000, 1000000, size=num_samples),
}

# Introduce missing values only for some features
missing_indices = np.random.choice(np.arange(num_samples), size=int(num_samples * 0.05), replace=False)
data['CGPA'][missing_indices] = np.nan

# Create DataFrame
df = pd.DataFrame(data)

# Set max age to 27 for bachelor's degree students
df.loc[(df['Education'] == "Bachelor") & (df['Age'] > 27), 'Age'] = 27

# Set min age to 24 for master's degree students
df.loc[(df['Education'] == "Master") & (df['Age'] < 24), 'Age'] = 24

# Add the 'HistoryOfBacklogs' column with random data
num_rows = df.shape[0]
df['HistoryOfBacklogs'] = np.random.randint(0, 5, size=num_rows)
```

```
# Add the 'Placement_Status' column with default 'Yes' values
df['Placement_Status'] = 'Placed'

# Update 'Placement_Status' based on conditions
df.loc[df['HistoryOfBacklogs'] > 2, 'Placement_Status'] = 'Not Placed'

# Modify the 'Package(LPA)' column to set salary to 0 for "Not Placed" category
df['Package(LPA)'] = np.where(df['Placement_Status'] == 'Not Placed', 0, df['Package(LPA)'])

# Save the dataset to CSV file
df.to_csv('campus_placement_dataset.csv', index=False)

print("Dataset created and saved as 'campus_placement_dataset.csv'")

## Data Collection

#Load the dataset
df = pd.read_csv('campus_placement_dataset.csv')

# Getting to know the shape of data
df.shape

# Showing the first 5 rows of the dataset
df.head()

# Showing the last 10 rows of the dataset
df.tail(10)

# Showing 4 rows of the dataset at random
df.sample(4)

# Getting to know the data type of columns that are in the dataset
df.dtypes

# Getting to know the data type of columns that are in the dataset
df.dtypes

# Getting to know the detailed information of the columns
df.info()

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

## Data Preprocessing

# missing values
df.isna().sum()
```

```
# Group by education level and specialization and fill missing CGPA with the median CGPA of the group
df['CGPA'] = df.groupby(['Education', 'Specialization'])['CGPA'].transform(lambda x: x.fillna(x.median()))
df.isna().sum()
# duplicate rows
df.duplicated().sum()
#drop duplicates
df.drop_duplicates(inplace=True)
# Check if the duplicate rows are removed
df.duplicated().sum()


##EDA
# Getting to know the correlation between the target column and other features.
df.corr(numeric_only=True)
# Correlation matrix to visualize the correlation between variables
plt.figure(figsize=(8, 6))
corr_matrix = df.corr(numeric_only=True)
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
# Plotting  the graph so that we can visualize the output with respect to major features
figure = px.scatter(df, x="CGPA", y="Internships", color="Placement_Status", color_discrete_map={"Placed": "green", "Not Placed": "red"}, hover_data=['CGPA'])
figure.show()
fig = px.box(df, x="Placement_Status", y="HistoryOfBacklogs",color="Placement_Status",color_discrete_map={"Placed": "green", "Not Placed": "red"}, title="Box Plot of Placement Status by History of Backlogs")
fig.show()
fig = px.box(df, x="Education", y="CGPA", title="Box and Violin Plot of CGPA by Education")
fig.update_traces(boxpoints="all", jitter=0.3, pointpos=-1.8)
fig.show()
```

```
# Plotting Histogram for the count of place and not placed

px.histogram(df, x='Placement_Status', color='Placement_Status',color_discrete_map={"Placed": "green", "Not Placed": "red"}, barmode='group')

# Pie Chart: Percentage pie chart of Placed or Not Placed

figure = px.pie(df, values=df['Placement_Status'].value_counts().values, names=df['Placement_Status'].value_counts().index, title='Placed Vs Not Placed')

figure.show()

# Pie chart for Gender distribution

plt.figure(figsize=(8, 6))

gender_counts = df['Gender'].value_counts()

plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', colors=['skyblue', 'pink'])

plt.title("Gender Distribution")

plt.show()


# Histogram for Numeric varibles

plt.figure(figsize=(8, 6))

sns.histplot(data=df, x='Age', kde=True)

plt.title("Age Distribution")

plt.show()


plt.figure(figsize=(8, 6))

sns.histplot(data=df, x="CGPA", kde=True)

plt.title("CGPA Distribution")

plt.show()


plt.figure(figsize=(8, 6))

sns.histplot(data=df, x="Internships", kde=True)

plt.title("Internships Distribution")

plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x="Year", kde=True)
plt.title("Year Distribution")
plt.show()

plt.figure(figsize=(8, 6))
sns.histplot(data=df, x="Package(LPA)", kde=True)
plt.title("Package(LPA) Distribution")
plt.show()

plt.figure(figsize=(8, 6))
sns.histplot(data=df, x="Projects", kde=True)
plt.title("Projects Distribution")
plt.show()

plt.figure(figsize=(8, 6))
sns.histplot(data=df, x="HistoryOfBacklogs", kde=True)
plt.title("HistoryOfBacklogs Distribution")
plt.show()

# Box plot for Numeric varibles by Placement_Status
plt.figure(figsize=(8, 6))
sns.boxplot(x="Placement_Status", y="Age", data=df, palette='Set1')
plt.title("Age Distribution by Placement Status")
plt.show()
plt.figure(figsize=(8, 6))
sns.boxplot(x="Placement_Status", y="CGPA", data=df, palette='Set2')
plt.title("CGPA Distribution by Placement Status")
plt.show()
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(x="Placement_Status", y="Internships", data=df, palette='Set3')
plt.title("Internships Distribution by Placement Status")
plt.show()
plt.figure(figsize=(8, 6))
sns.boxplot(x="Placement_Status", y="Year", data=df, palette='Set1')
plt.title("Year Distribution by Placement Status")
plt.show()
plt.figure(figsize=(8, 6))
sns.boxplot(x="Placement_Status", y="Projects", data=df, palette='Set2')
plt.title("Projects Distribution by Placement Status")
plt.show()
# Count plot for all varibles by Placement_Status
colors={"Placed": "blue", "Not Placed": "red"}
sns.countplot(x="Age", hue="Placement_Status", data=df, palette=colors)
plt.title("Age Distribution by Placement Status")
plt.show()
sns.countplot(x="Gender", hue="Placement_Status", data=df, palette=colors)
plt.title("Gender Distribution by Placement Status")
plt.show()
sns.countplot(x="Education", hue="Placement_Status", data=df, palette=colors)
plt.title("Education Distribution by Placement Status")
plt.show()
sns.countplot(x="Internships", hue="Placement_Status", data=df, palette=colors)
plt.title("Internships Distribution by Placement Status")
plt.show()
sns.countplot(x="Hostel", hue="Placement_Status", data=df, palette=colors)
plt.title("Hostel distribution by Placement Status")
plt.show()
```

```
sns.countplot(x="Letter_of_Recommendation", hue="Placement_Status", data=df, palette=colors)
plt.title("Letter_of_Recommendation distribution by Placement Status")
plt.show()
plt.figure(figsize=(12, 8))
sns.countplot(x="Specialization", hue="Placement_Status", data=df, palette=colors)
plt.title("Specialization distribution by Placement Status")
plt.show()
sns.countplot(x="HistoryOfBacklogs", hue="Placement_Status", data=df, palette=colors)
plt.title("HistoryOfBacklogs distribution by Placement Status")
plt.show()
sns.countplot(x="On/Off Campus", hue="Placement_Status", data=df, palette=colors)
plt.title("On/Off Campus distribution by Placement Status")
plt.show()
sns.countplot(x="Projects", hue="Placement_Status", data=df, palette=colors)
plt.title("Projects Distribution by Placement Status")
plt.show()

# Stacked bar chart for Education vs. Placement_Status
education_placement = df.groupby(['Education', 'Placement_Status']).size().unstack()
plt.figure(figsize=(8, 6))
education_placement.plot(kind='bar', stacked=True, color={"Placed": 'skyblue', "Not Placed": 'lightcoral'})
plt.title("Education vs. Placement Status")
plt.show()

# Calculate maximum and minimum placement counts per year
placement_counts = df.groupby('Year')['Placement_Status'].value_counts().unstack()
max_placement_year = placement_counts['Placed'].idxmax()
min_placement_year = placement_counts['Placed'].idxmin()
# Plot maximum and minimum placement counts
```

```
plt.figure(figsize=(10, 6))
placement_counts['Placed'].plot(marker='o', label='Placed')
plt.scatter(max_placement_year, placement_counts['Placed'].max(), color='green', label='Max Placement')
plt.scatter(min_placement_year, placement_counts['Placed'].min(), color='red', label='Min Placement')
plt.title("Placement Counts Over the Years")
plt.xlabel("Year")
plt.ylabel("Number of Placements")
plt.legend()
plt.show()
print(f"Year with the maximum placements: {max_placement_year}")
print(f"Year with the minimum placements: {min_placement_year}")

# Calculate total placements per year
total_placements_per_year = df[df['Placement_Status'] == 'Placed'].groupby('Year').size()
print("\nTotal placements per yearwise:")
print(total_placements_per_year)

# Line chart for Placement trends over years
plt.figure(figsize=(10, 6))
placement_trends = df.groupby('Year')['Placement_Status'].value_counts().unstack().fillna(0)
placement_trends.plot(marker='o')
plt.title("Placement Trends Over Years")
plt.xlabel("Year")
plt.ylabel("Number of Students")
plt.legend(title="Placement Status")
plt.show()

# Filter placed students
placed_students = df[df['Placement_Status'] == 'Placed']
```

```
# Find the youngest and eldest placed students using the 'Age' column
youngest_student = placed_students.loc[placed_students['Age'].idxmin()]
eldest_student = placed_students.loc[placed_students['Age'].idxmax()]
# Print the results
print("Youngest Student:")
print(youngest_student)
print("\nEldest Student:")
print(eldest_student)

# Calculate maximum and minimum internships for placed students
max_internships = placed_students['Internships'].max()
min_internships = placed_students['Internships'].min()
# Set up Seaborn style
plt.figure(figsize=(8, 6))
plt.title('Box Plot of Maximum and Minimum Internships for Placed Students')
# Create a box plot
plt.boxplot([placed_students['Internships']], labels=['Placed Students'])
plt.ylabel('Number of Internships')
plt.ylim(0, 5)  # Set y-axis limit to better visualize the data
plt.annotate(f"Max: {max_internships}", xy=(1.1, max_internships), xytext=(1.3, max_internships + 0.2),
             arrowprops=dict(facecolor='black', arrowstyle='->'))
plt.annotate(f"Min: {min_internships}", xy=(1.1, min_internships), xytext=(1.3, min_internships - 0.2),
             arrowprops=dict(facecolor='black', arrowstyle='->'))
plt.show()
# Calculate the count of students with the maximum and minimum number of internships
count_max_internships = len(df[(df['Placement_Status'] == 'Yes') & (df['Internships'] == max_internships)])
count_min_internships = len(df[(df['Placement_Status'] == 'Yes') & (df['Internships'] == min_internships)])
# Print the results
print("Maximum number of internships done by placed students: ", max_internships)
print("Number of students who did the maximum internships: ", count_max_internships)
```

```
print("Minimum number of internships done by placed students: ", min_internships)
print("Number of students who did the minimum internships: ", count_min_internships)


# Calculate the counts for each CGPA bin
cgpa_counts = df['CGPA'].value_counts().sort_index()
# Find the maximum and minimum CGPA values
max_cgpa = df['CGPA'].max()
min_cgpa = df['CGPA'].min()
# Set up Seaborn style
plt.figure(figsize=(10, 6))
# Create bar plot for CGPA counts
plt.bar(cgpa_counts.index, cgpa_counts.values, color='blue', alpha=0.5, label='CGPA Counts')
# Mark the maximum and minimum CGPA values with red color
plt.bar(max_cgpa, cgpa_counts[max_cgpa], color='red', label='Max CGPA')
plt.bar(min_cgpa, cgpa_counts[min_cgpa], color='green', label='Min CGPA')
plt.title('Bar Plot: Max and Min CGPA Counts')
plt.xlabel('CGPA')
plt.ylabel('Counts')
plt.legend()
plt.grid(True)
plt.show()
# Count the number of students with maximum and minimum CGPA
count_max_cgpa = df[df['CGPA'] == max_cgpa].shape[0]
count_min_cgpa = df[df['CGPA'] == min_cgpa].shape[0]
# Print the results
print("Maximum CGPA: ", max_cgpa)
print("Number of students with maximum CGPA: ", count_max_cgpa)
print("Minimum CGPA: ", min_cgpa)
print("Number of students with minimum CGPA: ", count_min_cgpa)
# Define the desired specialization order
```

```
specialization_order = ['Computer Science', 'Information Technology', 'Electrical', 'Electronics', 'Mechanical',
'Civil']
# Create a count plot for placement trends by specialization
plt.figure(figsize=(10, 6))
sns.countplot(x='Specialization', hue='Placement_Status', data=df, order=specialization_order,
palette=['skyblue', 'red'])
plt.title('Placement Trends by Specialization')
plt.xlabel('Specialization')
plt.ylabel('Number of Students')
plt.legend(title='Placement Status')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
# Filter only placed students
placed_df = df[df['Placement_Status'] == 'Placed']
# Calculate average placement package by year
average_placement_by_year = placed_df.groupby('Year')['Package(LPA)'].mean()
# Find highest and lowest placement packages
highest_placement = placed_df.loc[placed_df['Package(LPA)'].idxmax()]
lowest_placement = placed_df.loc[placed_df['Package(LPA)'].idxmin()]
# Create a bar plot for average placement by year
plt.figure(figsize=(10, 6))
sns.barplot(x=average_placement_by_year.index, y=average_placement_by_year.values, palette='viridis')
plt.xlabel('Year')
plt.ylabel('Average Placement Package (in LPA)')
plt.title('Average Placement Package by Year')
plt.tight_layout()
# Display the plot
plt.show()
# Visualize the highest and lowest placement packages
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=['Highest', 'Lowest'], y=[highest_placement['Package(LPA)'], lowest_placement['Package(LPA)']], palette='magma')

plt.ylabel('Placement Package (in LPA)')

plt.title('Highest and Lowest Placement Packages')

plt.tight_layout()

# Display the plot

plt.show()

print("Average placement by year:")

print(average_placement_by_year)

print("\nHighest placement:")

print(highest_placement)

print("\nLowest placement:")

print(lowest_placement)


# Set up Seaborn style

sns.set(style="whitegrid")

# Create a swarm plot for CGPA by Placement_Status

plt.figure(figsize=(10, 6))

sns.swarmplot(x='Placement_Status', y='CGPA', data=df, hue='Placement_Status', palette={'Placed': 'green', 'Not Placed': 'red'})

plt.title("CGPA Distribution by Placement Status")

plt.xlabel("Placement Status")

plt.ylabel("CGPA")

plt.show()


from sklearn.cluster import KMeans

# Select relevant attributes for clustering

attributes_for_clustering = ['Age', 'CGPA', 'Internships', 'Package(LPA)']

# Remove rows with missing values

df_cleaned = df.dropna(subset=attributes_for_clustering)
```

```
# Normalize the data
normalized_data = (df_cleaned[attributes_for_clustering] - df_cleaned[attributes_for_clustering].mean()) /
df_cleaned[attributes_for_clustering].std()
# Perform K-Means clustering
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters, n_init=10, random_state=42)
df_cleaned['Cluster'] = kmeans.fit_predict(normalized_data)
# Visualize the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_cleaned, x='CGPA', y='Package(LPA)', hue='Cluster', palette='tab10')
plt.title('K-Means Clustering of Campus Placement Data')
plt.xlabel('CGPA')
plt.ylabel('Package(LPA)')
plt.show()

# sns plot for all varibles by Placement_Status
sns.pairplot(data=df, hue="Placement_Status")

## Feature Engineering
df
# Convert categorical variables to numerical using LabelEncoder
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df['Education'] = le.fit_transform(df['Education'])
df['Hostel'] = le.fit_transform(df['Hostel'])
df['Letter_of_Recommendation'] = le.fit_transform(df['Letter_of_Recommendation'])
df['Specialization'] = le.fit_transform(df['Specialization'])
df['On/Off Campus'] = le.fit_transform(df['On/Off Campus'])
df['Placement_Status'] = le.fit_transform(df['Placement_Status'])
df.head(20)
```

```
plt.figure(figsize=(15, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()

#Extracting Input and Output Columns
X = df.drop(['Year','Hostel','On/Off Campus','Package(LPA)','Placement_Status'], axis=1)
y = df['Placement_Status']
X
y
# Getting the shape of the X and Y
print(X.shape)
print(y.shape)
## Data Splitting
# Splitting the dataset into training and testing datasets.
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.33, random_state=42)
# Getting the Shape of all the training and testing dataset
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

## Model Training
from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train, y_train)

## Model Evalution
y_pred= logreg.predict(X_test)
```

```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)

# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.2)
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            annot_kws={"size": 16}, linewidths=0.5, square=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

df1=pd.DataFrame({'Actual': y_test, 'Predict': y_pred})
df1
print("Accuracy Score for Test Dataset is ",model.score(X_test, y_test)*100,"%")
print("Accuracy Score for Train Dataset is",model.score(X_train,y_train)*100,"%")
```

```
## Save the Model
with open("mytrainedplacement_model.pkl", "wb") as model_file:
    pickle.dump(model, model_file)

## Load the Model
with open("mytrainedplacement_model.pkl", "rb") as model_file:
    loaded_model = pickle.load(model_file)
```

Python flask file:

```
from flask import Flask, request, render_template,redirect,url_for

import numpy as np import joblib  import pickle app =

Flask(__name__)

model1 =pickle.load(open('myplacementnow.pkl',"rb"))

ct=joblib.load("myplacementnow.pkl")


@app.route('/') def

hel():

    return render_template("current.html")

@app.route('/login') def

log():

    return render_template("login.html")


@app.route('/sec')
def hello():

    return render_template("index.html")


@app.route('/guest', methods=['GET','POST'])

def Guest():     if request.method == 'POST':

age = request.form['age']         gender =

request.form['gender']         stream =

request.form['stream']         internship =

request.form['internship']         cgpa =

request.form['cgpa']         backlogs =

request.form['backlogs']

internship=request.form['internship']

 return render_template("index2.html")
```

```
@app.route('/y_predict',methods=["POST"]) def
y_predict():
   x_test=[yo for yo in request.form.values()]
predict_output=model1.predict([x_test])     if
predict_output==0:
      return render_template("unsel.html")
else:
     return render_template("sel.html") if
__name__=="__main__":
app.run(debug=True,port=8000)
```