Samash technical Test : Student Data Analytics

In [17]:

```
import pandas as pd
import numpy as np
import csv
from datetime import datetime
import seaborn as sns
import matplotlib.pyplot as plt
```

In [18]:

```
dataSet = pd.read_csv("student_data.csv")
```

In [19]:

```
dataSet.head()
```

Out[19]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees
0	165527	Bryan Rogers	2006-01-19	Computer Science	2020	2017	3	Web Development	155152	19572
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760
2	740021	David Robinson	1997-12-02	Civil Engineering	2017	2022	1	Network Security	55662	5871
3	433076	Susan Miller	1999-10-30	Computer Science	2021	2019	1	Data Science	134955	17284
4	441628	Brittany Martin	1998-01-10	Chemical Engineering	2016	2018	1	Network Security	125934	14871

In [20]:

```
dataSet.shape
```

Out[20]:

```
(200000, 10)
```

In [21]:

```
dataSet.columns
```

Out[21]:

```
Index(['Student ID', 'Student Name', 'Date of Birth', 'Field of Study',
       'Year of Admission', 'Expected Year of Graduation', 'Current Semester',
       'Specialization', 'Fees', 'Discount on Fees'],
      dtype='object')
```

In [22]:

```
dataSet.dtypes
```

Out[22]:

```
Student ID                       int64
Student Name                    object
Date of Birth                   object
Field of Study                  object
Year of Admission                int64
Expected Year of Graduation      int64
Current Semester                 int64
Specialization                  object
Fees                             int64
Discount on Fees                 int64
dtype: object
```

In [23]:

```
missing_values = dataSet.isnull().sum()
missing_values
```

Out[23]:

```
Student ID                      0
Student Name                    0
Date of Birth                   0
Field of Study                  0
Year of Admission               0
Expected Year of Graduation     0
Current Semester                0
Specialization                  0
Fees                            0
Discount on Fees                0
dtype: int64
```

Cleaning & data quality

Parsing dates

In [26]:

```
#Verify the date formate using : to_date and handeling errors
#The errors ='coerce' parameter handles any invalid or unparsable dates by converting them to NaT (Not a
#We would like to have this formate %Y-%m-%d for all the data attributes

def clean_dates(data, col):
    data[col] = pd.to_datetime(data[col], errors='coerce')
    return data


cleaned_data = clean_dates(dataSet, 'Date of Birth')

cleaned_data.head()
cleaned_data.dtypes
```

Out[26]:

```
Student ID                                int64
Student Name                             object
Date of Birth                    datetime64[ns]
Field of Study                           object
Year of Admission                         int64
Expected Year of Graduation               int64
Current Semester                          int64
Specialization                           object
Fees                                      int64
Discount on Fees                          int64
dtype: object
```

In [27]:

```
def clean_parse_string(data):
    object_columns = data.select_dtypes(include='object').columns
    data[object_columns] = data[object_columns].apply(lambda x: x.astype(str))
    return data
cleaned_data = clean_parse_string(cleaned_data)
cleaned_data.infer_objects().dtypes
```

Out[27]:

```
Student ID                                int64
Student Name                             object
Date of Birth                    datetime64[ns]
Field of Study                           object
Year of Admission                         int64
Expected Year of Graduation               int64
Current Semester                          int64
Specialization                           object
Fees                                      int64
Discount on Fees                          int64
dtype: object
```

Adjusting admission year and graduation year column values

In [28]:

```
# Function to compare Admission and graduation Years. the second column (graduation Year) must contain gre
def compare_columns(df, column1, column2, result_column):
    df[result_column] = df[column1] > df[column2]
    mask = df[column1] > df[column2]
    df.loc[mask, [column1, column2]] = df.loc[mask, [column2, column1]].values
    return df

# Compare columns and add the result to a new column
compared_data = compare_columns(dataSet, 'Year of Admission', 'Expected Year of Graduation', 'result_colu

# Print the modified DataFrame
compared_data.head(2)
```

Out[28]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resul
0	165527	Bryan Rogers	2006-01-19	Computer Science	2017	2020	3	Web Development	155152	19572	
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760	

Getting Expected Education duration for each student

In [29]:

```
df=compared_data
df['Expected Student Education Duration'] = df['Expected Year of Graduation'] - df['Year of Admission'] +
df.head(2)
```

Out[29]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resul
0	165527	Bryan Rogers	2006-01-19	Computer Science	2017	2020	3	Web Development	155152	19572	
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760	

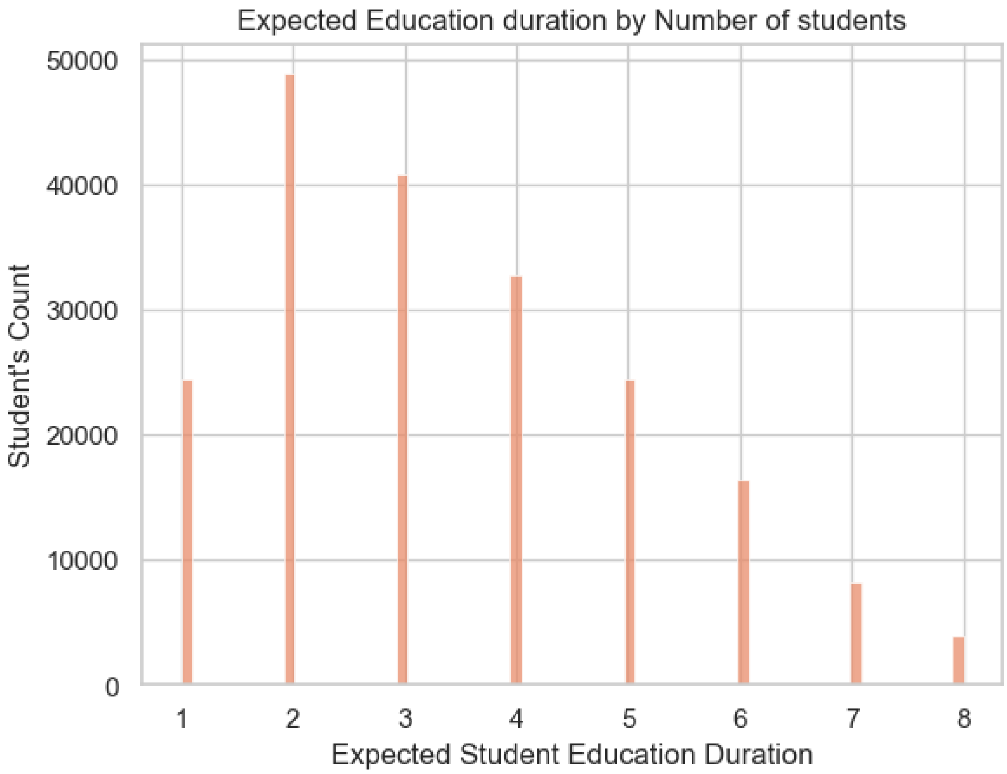
In [74]:

```
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid", palette="flare")
sns.histplot(data=df, x="Expected Student Education Duration")
plt.title("Expected Education duration by Number of students")
axes = plt.gca()
axes.set_ylabel("Student's Count")

"""for bar in axes.patches:
    # Get the height of the bar
    height = bar.get_height()

    # Add text annotation on top of the bar
    axes.annotate(f'{height:.2f}',
                  (bar.get_x() + bar.get_width() / 2, height),
                  ha='center', va='bottom')"""

plt.show()
```



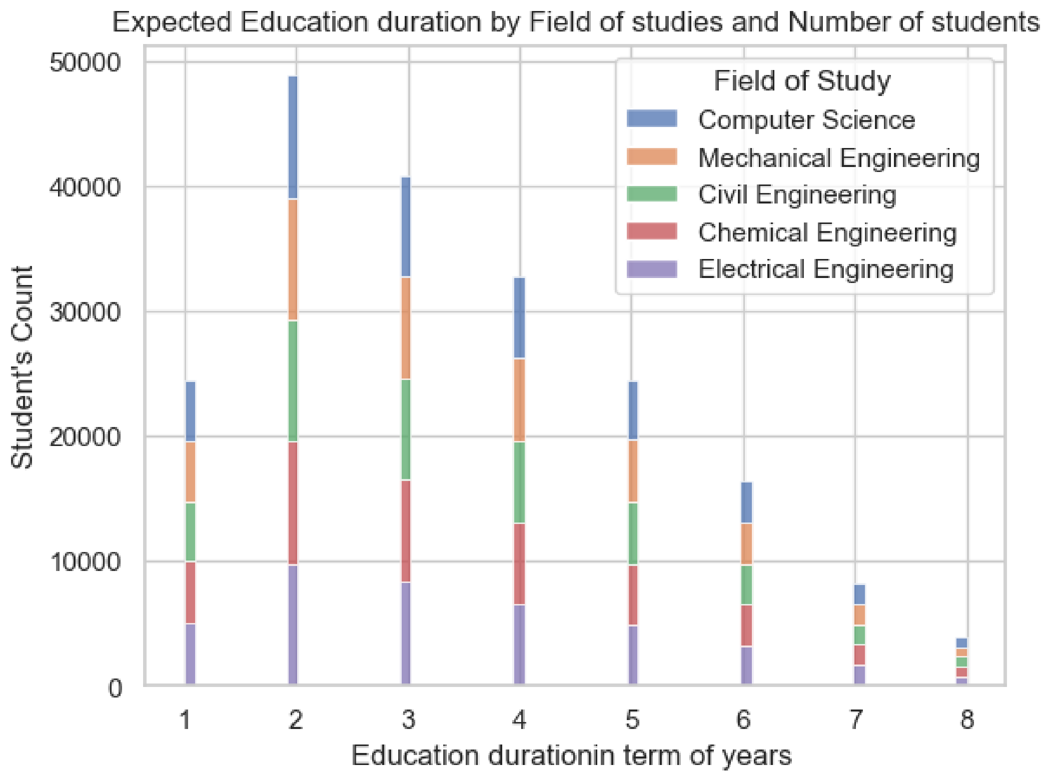
In [62]:

```
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid", palette="deep")
sns.histplot(data=df, x="Expected Student Education Duration",  hue="Field of Study", multiple="stack")
axes = plt.gca()
axes.set_ylabel("Student's Count")
axes.set_xlabel("Education durationin term of years ")
plt.title("Expected Education duration by Field of studies and Number of students")

plt.show
```

Out[62]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Adjusting semester dispatching for student expected to graduate the same admission year

In [198]:

```
#We noticed that in some rows, the year of admission equals year of graduation
filtered_df = df[df['Year of Admission'] == df['Expected Year of Graduation']]
filtered_df.head(2)
```

Out[198]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resul
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760	
6	268329	Erica Owens	2003-06-20	Mechanical Engineering	2020	2020	1	Artificial Intelligence	52994	5231	

In [199]:

```
#we can notice that if in the same year the student's current semester  is greater then 2, the value is il
#to minimize the error, if the expected year isn't 2023 then we set the value at 2 (graduated)


df.loc[df['Year of Admission'] == df['Expected Year of Graduation'] , 'Current Semester'] = 2
df.head(2)
```

Out[199]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resul
0	165527	Bryan Rogers	2006-01-19	Computer Science	2017	2020	3	Web Development	155152	19572	
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760	

In [132]:

```
#Since we're in the first semester of 2023, we turn the value of undergraduated student to semester 1
duplication = filtered_df [filtered_df ['Expected Year of Graduation'] == 2023]
print(duplication)
#there's no such condition, we won't apply changes on df which is the copy of the data source
```

```
Empty DataFrame
Columns: [Student ID, Student Name, Date of Birth, Field of Study, Year of Admission, Expec
ted Year of Graduation, Current Semester, Specialization, Fees, Discount on Fees, result_co
lumn, Expected Student Education Duration]
Index: []
```

In [133]:

```
#the portion of student expected to graduate the same year they are admitted in the total data set

portion = len(filtered_df) / len(df)
print("portion shape is", filtered_df.shape)
portionPourcentage = len(filtered_df) / len(df) * 100
print("pourcentage of duplication", portionPourcentage)
```

```
portion shape is (24498, 12)
pourcentage of duplication 12.249
```

```
In [81]:
```

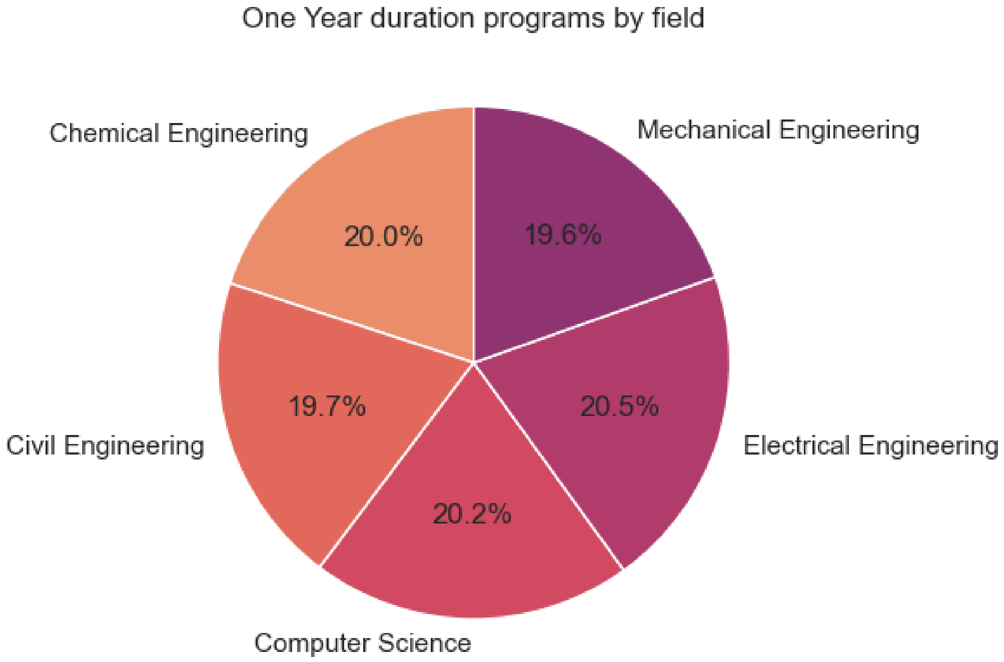
```
#We will assume that  the proportion of students within each field of study
#who are expected to graduate in the same year they were admitted

grouped_df = df.groupby('Field of Study').apply(lambda x: len(x[x['Year of Admission']
                                                                == x['Expected Year of Graduation']]) / l
print(" Pourcentage of One Year duration field compared to the other periods per field: \n", grouped_df)
grouped_df.plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('One Year duration programs by field')
```

```
 Pourcentage of One Year duration field compared to the other periods per field:
 Field of Study
Chemical Engineering      12.241379
Civil Engineering         12.093327
Computer Science          12.389890
Electrical Engineering    12.531944
Mechanical Engineering    11.985178
dtype: float64
```

```
Out[81]:
```

```
Text(0.5, 1.0, 'One Year duration programs by field')
```



Pourcentage of ungraduate students (still in Uni) and their age

In [82]:

```
# Selecting rows with graduation year >= 2023
age = df[df['Expected Year of Graduation'] >= 2023]
# get current year
current_year = datetime.now().year
age['Age of Undergraduated student'] = current_year - pd.DatetimeIndex(age['Date of Birth']).year
age.head(2)
#age.shape
```

```
C:\Users\asus\AppData\Local\Temp\ipykernel_8836\1112744852.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gui
de/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy)
  age['Age of Undergraduated student'] = current_year - pd.DatetimeIndex(age['Date of Birt
h']).year
```

Out[82]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resu
18	439048	Gary Garcia	2005-07-21	Chemical Engineering	2020	2023	3	Machine Learning	88470	292	
22	400958	Nicole Grimes	1998-11-30	Civil Engineering	2021	2023	4	Artificial Intelligence	152798	19868	

In [165]:

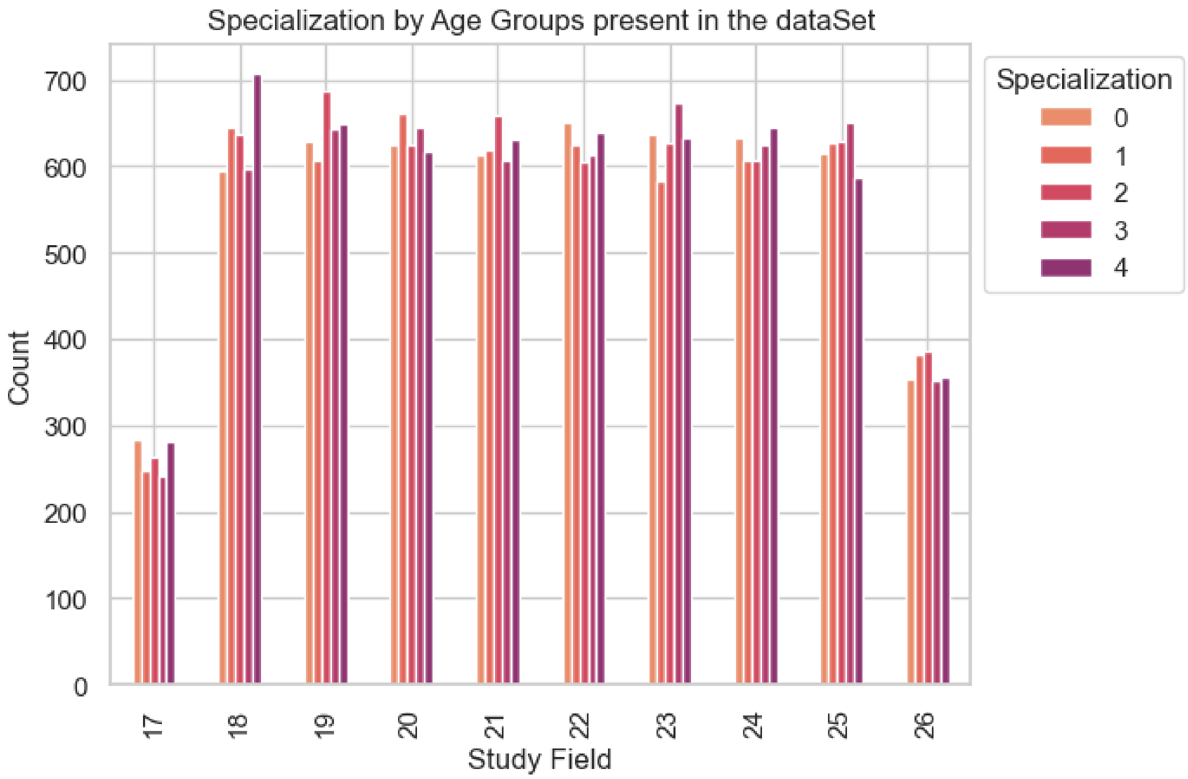
```


grouped_students = age.groupby('Age of Undergraduated student')[ "Specialization"].value_counts().unstack
ax = grouped_students.plot(kind='bar')


# Set the x-axis label
plt.xlabel('Study Field')

# Set the y-axis label
plt.ylabel('Count')

# Set the title of the plot
plt.title('Specialization by Age Groups present in the dataSet')
sns.move_legend(ax, "upper left", bbox_to_anchor=(1, 1))
# Display the plot
plt.show()
```



Graduate vs Ungraduate students pourcentage

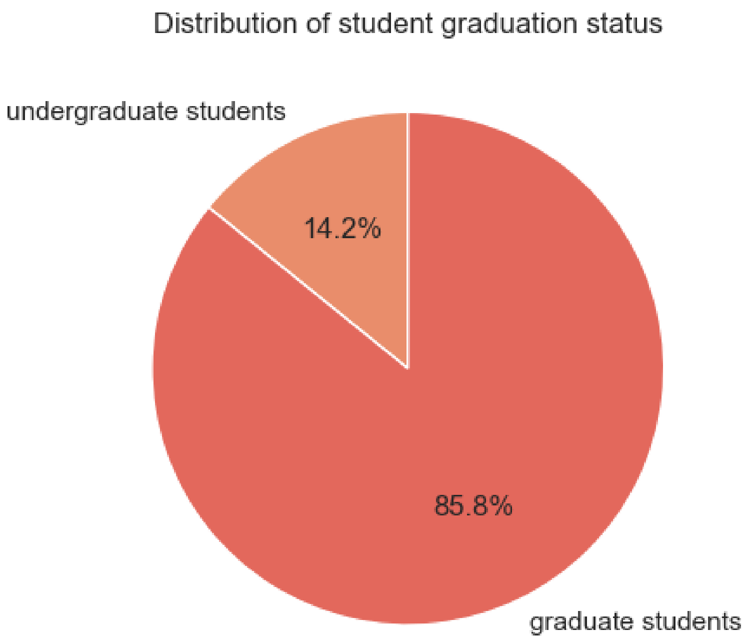
In [108]:

```
# we want to see the diffrent pourcentage of student graduation statu in the total data
undergraduate_Pourcentage = len(age) / len(df) * 100
graduate_Pourcentage = 100 - undergraduate_Pourcentage
print("pourcentage of undergraduate student present in the dataset", undergraduate_Pourcentage,"%")
print("pourcentage of graduated student  present in the dataset",graduate_Pourcentage,"%")
```

```
pourcentage of undergraduate student present in the dataset 14.180000000000001 %
pourcentage of graduated student  present in the dataset 85.82 %
```

In [109]:

```
plt.pie([undergraduate_Pourcentage,graduate_Pourcentage], labels=['undergraduate students','graduate stud
plt.title('Distribution of student graduation status')
plt.show()
```



Fees analysis based on Field of studies using Kmeans

In [202]:

```
unique_fields= df['Field of Study'].unique()
print('Number of main fields of studies is : ', len(unique_fields))
unique_fields
```

```
Number of main fields of studies is :  5
```

Out[202]:

```
array(['Computer Science', 'Mechanical Engineering', 'Civil Engineering',
       'Chemical Engineering', 'Electrical Engineering'], dtype=object)
```

In [203]:

```
df['Fees after reduction'] = df['Fees'] - df['Discount on Fees']
df.head(2)
```

Out[203]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	resul
0	165527	Bryan Rogers	2006-01-19	Computer Science	2017	2020	3	Web Development	155152	19572	
1	635763	James Hogan	1999-05-23	Mechanical Engineering	2020	2020	2	Machine Learning	157870	14760	

In [112]:

```python

from sklearn.cluster import KMeans

from sklearn.preprocessing import LabelEncoder


#string to numeric value

def convert_string_columns(data):

    encoded_data = data.copy()

    for column in encoded_data.select_dtypes(include='object'):

        label_encoder = LabelEncoder()

        encoded_data[column] = label_encoder.fit_transform(encoded_data[column])

    return encoded_data

# date to numeric value
try :
    age['Reduction pourcentage'] = age['Discount on Fees'] / age['Fees'] * 100
except:
    print("problem of assignment")

age['Date of Birth'] = pd.to_datetime(df['Date of Birth'])

age['Date of Birth'] = (pd.to_datetime('2023-06-17') - age['Date of Birth'] ).dt.total_seconds().abs()

age = convert_string_columns(age)

age.head(2)
```

Out[112]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	re
18	439048	8435	565056000.0	0	2020	2023	3	2	88470	292	
22	400958	18189	774576000.0	1	2021	2023	4	0	152798	19868	

In [163]:

```
    kmeans = KMeans(n_clusters=5)

    kmeans.fit(age)

    kmeans.fit(age[['Age of Undergraduated student','Reduction pourcentage','Field of Study']])

    # Get the cluster labels
    cluster_labels = kmeans.labels_

    # Add the cluster labels to the data frame
    age['Cluster'] = cluster_labels

    age.head()
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: T
he default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init
` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: T
he default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init
` explicitly to suppress the warning
  warnings.warn(
```

Out[163]:

	Student ID	Student Name	Date of Birth	Field of Study	Year of Admission	Expected Year of Graduation	Current Semester	Specialization	Fees	Discount on Fees	res
18	439048	8435	565056000.0	0	2020	2023	3	2	88470	292	
22	400958	18189	774576000.0	1	2021	2023	4	0	152798	19868	
25	328657	18036	767836800.0	4	2019	2023	1	2	87936	2612	
28	989259	19293	813196800.0	1	2019	2023	2	2	135043	993	
38	200648	7856	562032000.0	4	2018	2023	2	3	194644	36082	

In [128]:

```
df_total = dataSet[['Student ID', 'Field of Study']]
df_age = age[['Student ID', 'Field of Study']]

# Merge the selected data frames based on ID
result_merge = pd.merge(df_total, df_age, on='Student ID', how='inner')
sorted_df = result_merge.sort_values('Field of Study_x')
sorted_df = sorted_df.drop_duplicates(subset='Field of Study_x')
sorted_df
```

Out[128]:

	Student ID	Field of Study_x	Field of Study_y
0	439048	Chemical Engineering	0
19737	957902	Civil Engineering	1
26593	991070	Computer Science	2
9190	527903	Electrical Engineering	3
13019	849558	Mechanical Engineering	4

In [129]:

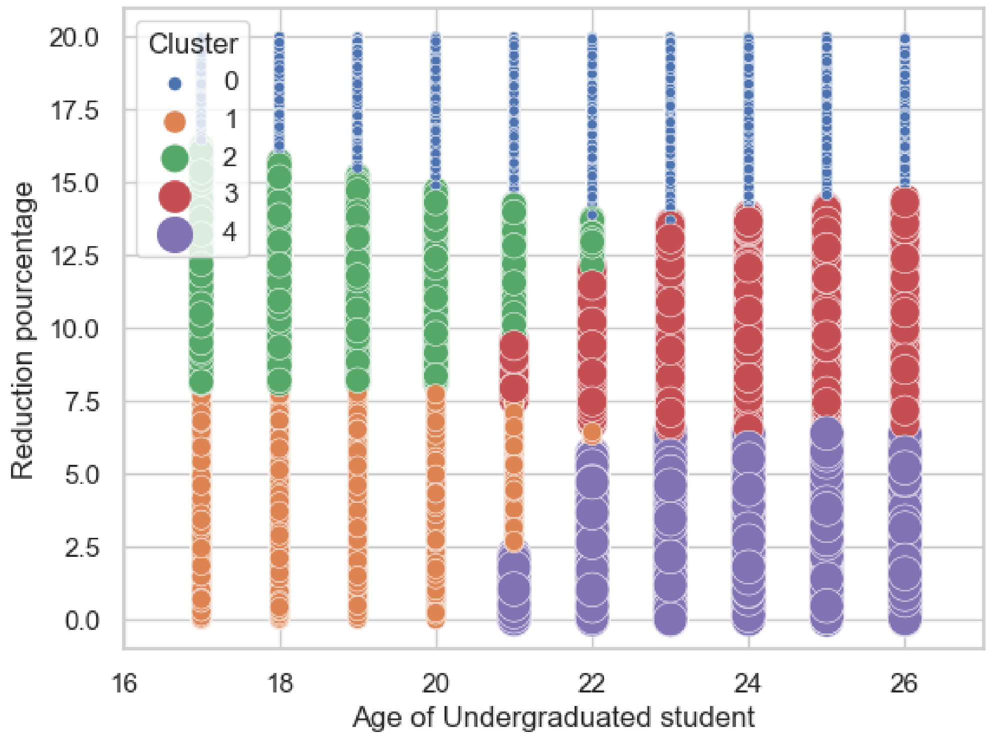
```
sns.scatterplot(data=age, x='Age of Undergraduated student', y='Reduction pourcentage',
                size = "Cluster",sizes=(20, 200),hue='Cluster', palette='deep')

min_age = age['Age of Undergraduated student'].min()

max_age = age['Age of Undergraduated student'].max()

plt.xlim(xmin=min_age - 1, xmax=max_age + 1, emit=True, auto=False)


plt.show()
```


Cluster
Reduction pourcentage
Age of Undergraduated student