

In [1]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 import seaborn as sns
6 from IPython import get_ipython
7 import warnings
8 warnings.filterwarnings("ignore")
```

In [2]:

```
1 data = pd.read_csv('online_education.csv')
```

In [3]:

```
1 data.head()
```

Out[3]:

	Education Level	Institution Type	Gender	Age	Device	IT Student	Location	Financial Condition	Internet Type	Network Type	F
0	University	Private	Male	23	Tab	No	Town	Mid	Wifi	4G	M
1	University	Private	Female	23	Mobile	No	Town	Mid	Mobile Data	4G	M
2	College	Public	Female	18	Mobile	No	Town	Mid	Wifi	4G	M
3	School	Private	Female	11	Mobile	No	Town	Mid	Mobile Data	4G	M
4	School	Private	Female	18	Mobile	No	Town	Poor	Mobile Data	3G	

In [4]:

```
1 data.tail()
```

Out[4]:

	Education Level	Institution Type	Gender	Age	Device	IT Student	Location	Financial Condition	Internet Type	Network Type	F
1200	College	Private	Female	18	Mobile	No	Town	Mid	Wifi	4G	M
1201	College	Private	Female	18	Mobile	No	Rural	Mid	Wifi	4G	M
1202	School	Private	Male	11	Mobile	No	Town	Mid	Mobile Data	3G	
1203	College	Private	Female	18	Mobile	No	Rural	Mid	Wifi	4G	M
1204	School	Private	Female	11	Mobile	No	Town	Poor	Mobile Data	3G	

In [5]:



```
1 data.shape
```

Out[5]:

```
(1205, 11)
```

In [6]:



```
1 data.columns
```

Out[6]:

```
Index(['Education Level', 'Institution Type', 'Gender', 'Age', 'Device',  
      'IT Student', 'Location', 'Financial Condition', 'Internet Type',  
      'Network Type', 'Flexibility Level'],  
      dtype='object')
```

In [7]:



```
1 data.duplicated().sum()
```

Out[7]:

```
980
```

In [8]:



```
1 data.isnull().sum()
```

Out[8]:

```
Education Level      0  
Institution Type     0  
Gender               0  
Age                 0  
Device              0  
IT Student           0  
Location             0  
Financial Condition  0  
Internet Type        0  
Network Type         0  
Flexibility Level    0  
dtype: int64
```

In [9]:



```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1205 entries, 0 to 1204
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Education Level        1205 non-null   object
1   Institution Type        1205 non-null   object
2   Gender                  1205 non-null   object
3   Age                     1205 non-null   int64
4   Device                  1205 non-null   object
5   IT Student              1205 non-null   object
6   Location                1205 non-null   object
7   Financial Condition     1205 non-null   object
8   Internet Type           1205 non-null   object
9   Network Type            1205 non-null   object
10  Flexibility Level       1205 non-null   object
dtypes: int64(1), object(10)
memory usage: 103.7+ KB
```

In [10]:



```
1 data.describe()
```

Out[10]:

	Age
count	1205.000000
mean	17.065560
std	5.830369
min	9.000000
25%	11.000000
50%	18.000000
75%	23.000000
max	27.000000

In [11]:

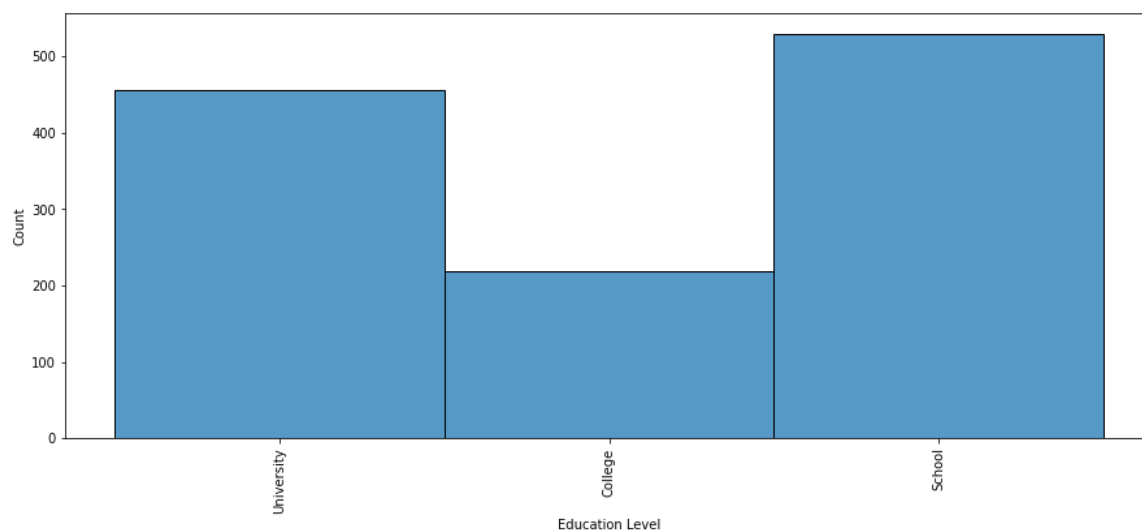
```
1 data.nunique()
```

Out[11]:

```
Education Level      3
Institution Type     2
Gender               2
Age                 6
Device              3
IT Student           2
Location            2
Financial Condition  3
Internet Type        2
Network Type         3
Flexibility Level    3
dtype: int64
```

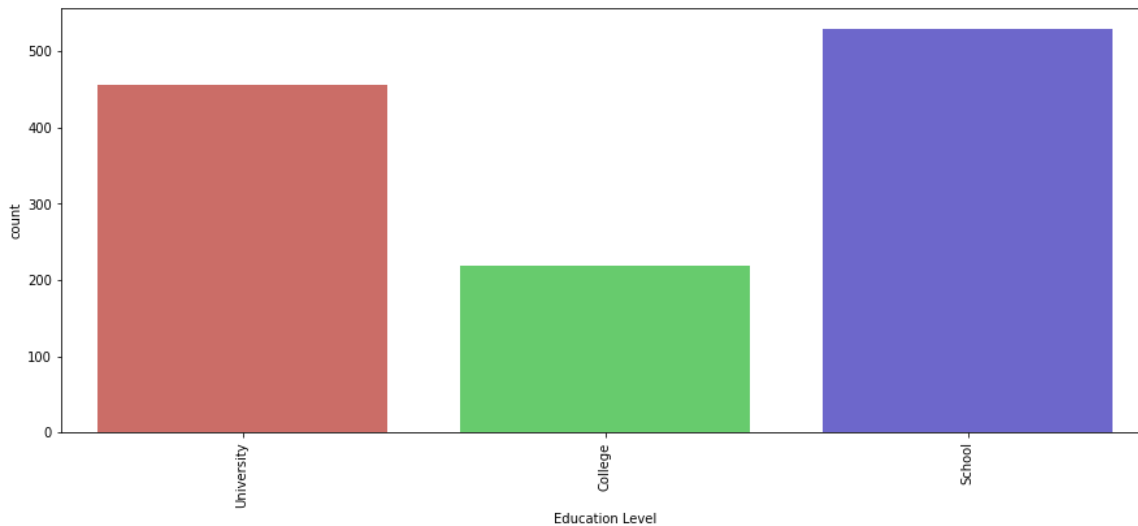
In [12]:

```
1 for i in data.columns:
2     plt.figure(figsize=(15,6))
3     sns.histplot(data[i], bins=10)
4     plt.xticks(rotation = 90)
5     plt.show()
```



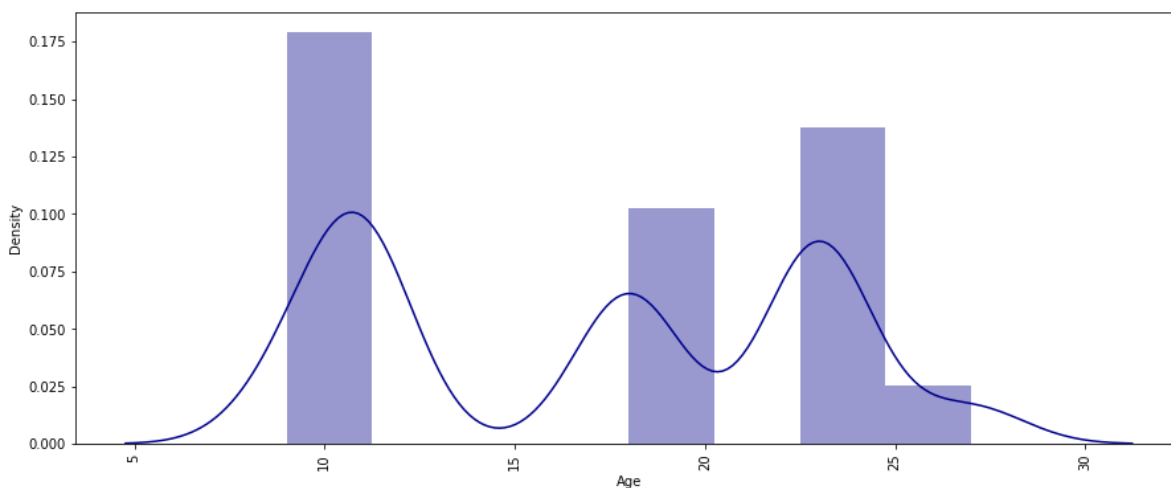
In [13]:

```
1 for i in data.columns:
2     plt.figure(figsize=(15,6))
3     sns.countplot(data[i], data = data,
4                   palette='hls')
5     plt.xticks(rotation = 90)
6     plt.show()
```



In [14]:

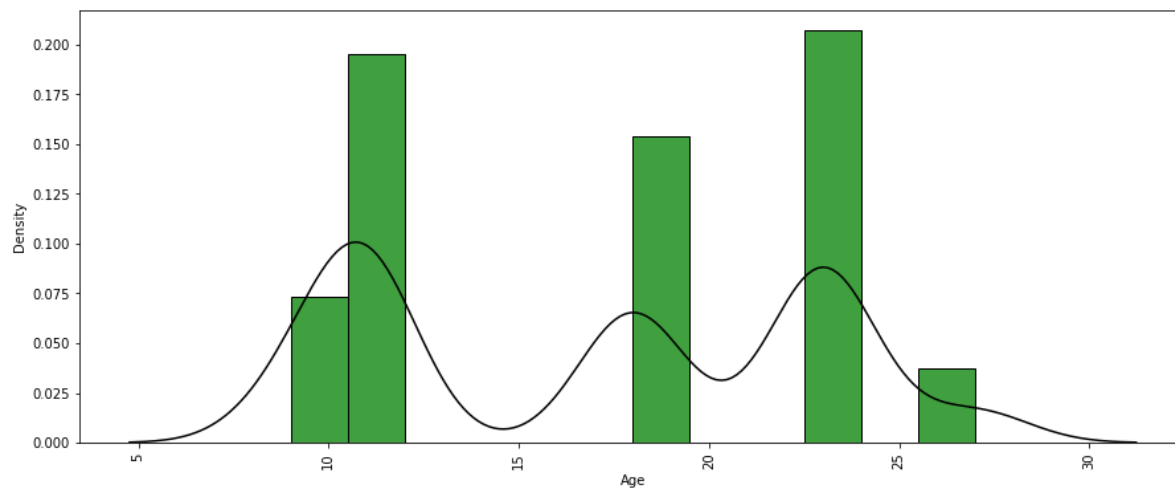
```
1 plt.figure(figsize=(15,6))
2 sns.distplot(data['Age'], kde = True, color = 'Darkblue')
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [15]:



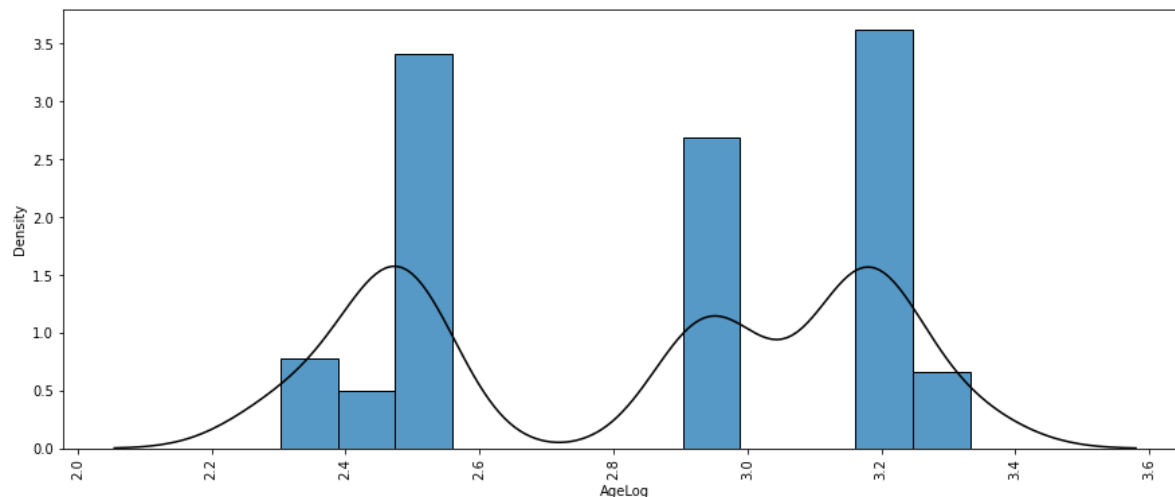
```
1 plt.figure(figsize=(15,6))
2 sns.histplot(data["Age"], stat='density',color='green')
3 sns.kdeplot(data["Age"], color='black')
4 plt.xticks(rotation = 90)
5 plt.show()
```



In [16]:



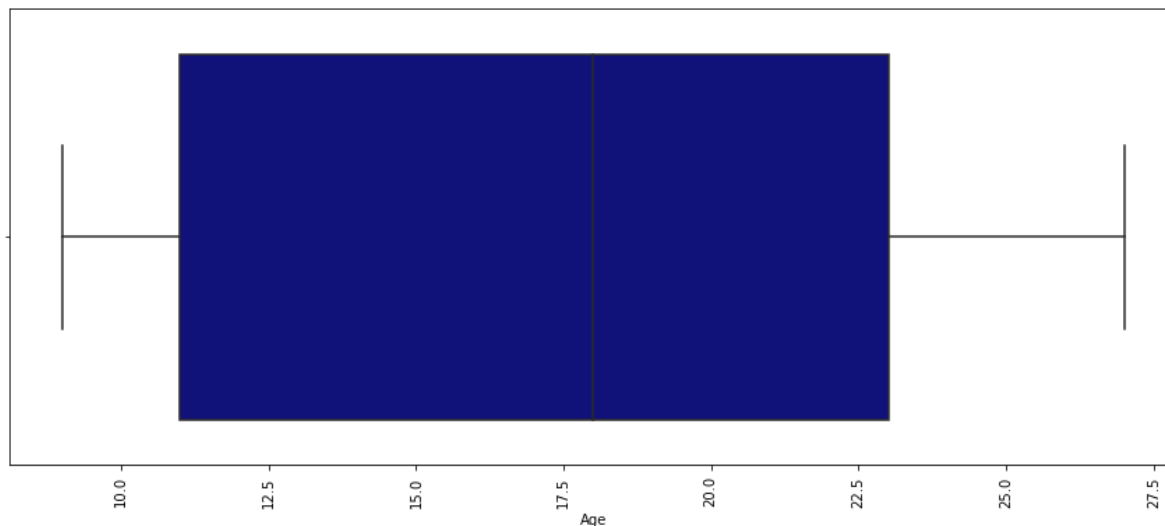
```
1 plt.figure(figsize=(15,6))
2 data['AgeLog'] = np.log(data['Age']+1)
3 sns.histplot(data["AgeLog"], stat='density')
4 sns.kdeplot(data["AgeLog"], color='black')
5 plt.xticks(rotation = 90)
6 plt.show()
```



In [17]:



```
1 plt.figure(figsize=(15,6))
2 sns.boxplot(data['Age'],color = 'Darkblue')
3 plt.xticks(rotation = 90)
4 plt.show()
```



In [18]:



```
1 from sklearn.preprocessing import LabelEncoder
2 le = LabelEncoder()
```

In [19]:



```
1 FlexibilityLevel = le.fit_transform(data['Flexibility Level'])
2 data['Flexibility Level'] = FlexibilityLevel
```

In [20]:



```
1 categorical_features = ['Education Level','Gender','Institution Type',
2                         'Device','IT Student','Location',
3                         'Financial Condition','Internet Type',
4                         'Network Type']
```

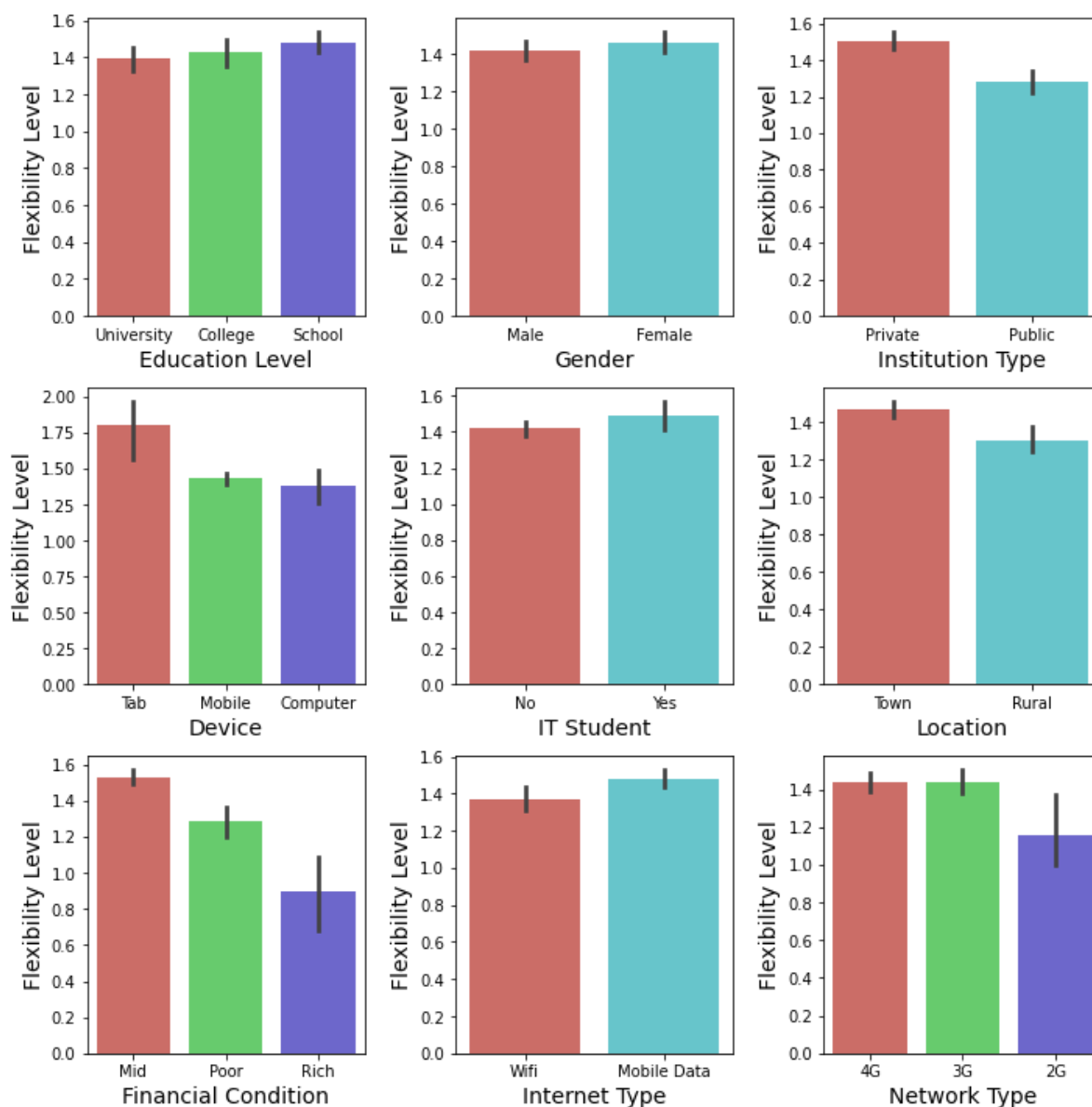
In [21]:



```

1 fig, axes = plt.subplots(3, 3, figsize=(10,10))
2 axes = [ax for axes_row in axes for ax in axes_row]
3 target = 'Flexibility Level'
4
5 for i, c in enumerate(categorical_features):
6     sns.barplot(data[c], data[target], ax=axes[i],
7                 palette = 'hls')
8     axes[i].set_ylabel('Flexibility Level', fontsize=14)
9     axes[i].set_xlabel(c, fontsize=14)
10
11 plt.tight_layout()
12 plt.show()

```



In [22]:

```

1 EducationLevel= le.fit_transform(data['Education Level'])
2 Gender = le.fit_transform(data['Gender'])
3 InstitutionType = le.fit_transform(data['Institution Type'])
4 Device = le.fit_transform(data['Device'])
5 ITStudent= le.fit_transform(data['IT Student'])
6 Location = le.fit_transform(data['Location'])
7 FinancialCondition = le.fit_transform(data['Financial Condition'])
8 InternetType = le.fit_transform(data['Internet Type'])
9 NetworkType = le.fit_transform(data['Network Type'])

```

In [23]:

```

1 data['Education Level'] = EducationLevel
2 data['Gender'] = Gender
3 data['Institution Type'] = InstitutionType
4 data['Device'] = Device
5 data['IT Student'] = ITStudent
6 data['Location'] = Location
7 data['Financial Condition'] = FinancialCondition
8 data['Internet Type'] = InternetType
9 data['Network Type'] = NetworkType

```

In [24]:

```
1 data.head()
```

Out[24]:

	Education Level	Institution Type	Gender	Age	Device	IT Student	Location	Financial Condition	Internet Type	Network Type	F
0	2	0	1	23	2	0	1	0	1	2	
1	2	0	0	23	1	0	1	0	0	2	
2	0	1	0	18	1	0	1	0	1	2	
3	1	0	0	11	1	0	1	0	0	2	
4	1	0	0	18	1	0	1	1	0	1	

In [25]:

```

1 x = data.drop('Flexibility Level',axis=1)
2 y = data['Flexibility Level']

```

In [26]:

```
1 x.shape
```

Out[26]:

```
(1205, 11)
```

In [27]:

```
1 y.shape
```

Out[27]:

```
(1205,)
```

In [28]:

```
1 from sklearn.preprocessing import StandardScaler
```

In [29]:

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(x, y,
3                                                    test_size=0.30,
4                                                    random_state=42)
```

In [30]:

```
1 sc = StandardScaler()
2 X_train = sc.fit_transform(X_train)
3 X_test = sc.fit_transform(X_test)
```

In [31]:

```
1 from sklearn.tree import DecisionTreeClassifier
2 model = DecisionTreeClassifier()
3 model.fit(X_train,y_train)
```

Out[31]:

```
DecisionTreeClassifier()
```

In [32]:

```
1 y_pred = model.predict(X_test)
```

In [33]:

```
1 print("Training Accuracy :", model.score(X_train, y_train))
2 print("Testing Accuracy :", model.score(X_test, y_test))
```

```
Training Accuracy : 0.8623962040332147
```

```
Testing Accuracy : 0.8011049723756906
```

In [34]:



```
1 from sklearn.metrics import classification_report
```

In [35]:

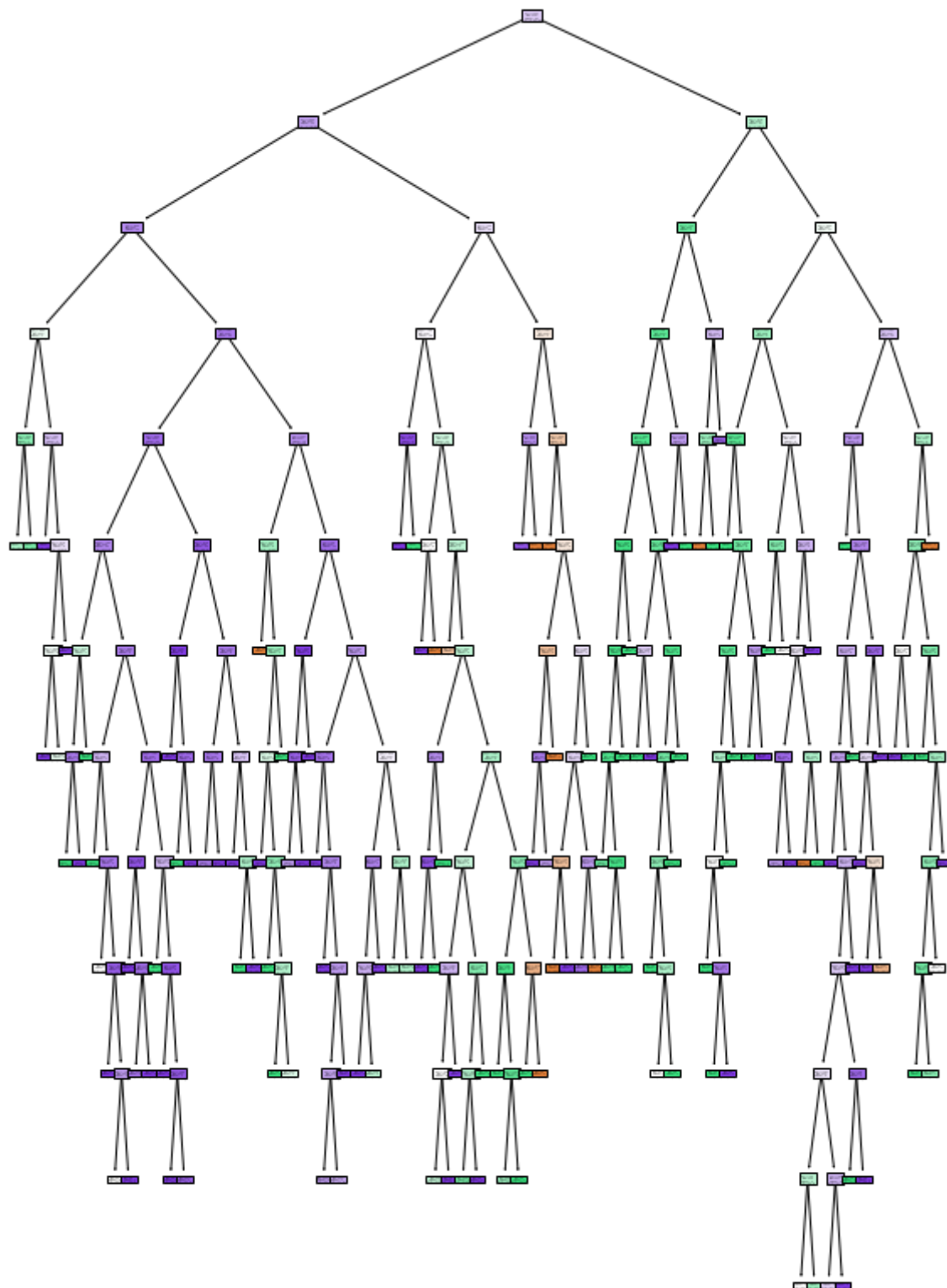


```
1 print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.85	0.56	0.68	39
1	0.82	0.82	0.82	154
2	0.78	0.84	0.81	169
accuracy			0.80	362
macro avg	0.82	0.74	0.77	362
weighted avg	0.80	0.80	0.80	362

In [36]:

```
1 from sklearn import tree
2 plt.figure(figsize=(10,15))
3 tree.plot_tree(model,filled=True)
4 plt.show()
```



In [37]:



```
1 model.feature_importances_
```

Out[37]:

```
array([0.12189951, 0.09626619, 0.1171519 , 0.02145348, 0.04924518,  
       0.06726136, 0.05672186, 0.12951326, 0.06101387, 0.12435672,  
       0.15511667])
```

In [38]:



```
1 from sklearn.ensemble import RandomForestClassifier  
2 classifier= RandomForestClassifier(n_estimators= 10,  
3                                   criterion="entropy")
```

In [39]:



```
1 classifier.fit(X_train, y_train)
```

Out[39]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [40]:



```
1 y_pred = classifier.predict(X_test)
```

In [41]:



```
1 print("Training Accuracy :", classifier.score(X_train, y_train))
2 print("Testing Accuracy :", classifier.score(X_test, y_test))
```

```
Training Accuracy : 0.8588374851720048
```

```
Testing Accuracy : 0.787292817679558
```

In [42]:



```
1 print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.54	0.69	39
1	0.78	0.84	0.81	154
2	0.77	0.80	0.78	169
accuracy			0.79	362
macro avg	0.84	0.72	0.76	362
weighted avg	0.80	0.79	0.78	362

In [43]:



```
1 classifier.feature_importances_
```

Out[43]:

```
array([0.07576819, 0.08319807, 0.11572646, 0.0980481 , 0.05521592,
       0.06144166, 0.06213362, 0.18141846, 0.07569019, 0.08614288,
       0.10521646])
```

In [44]:



```
1 grid_param = {
2     'criterion': ['gini', 'entropy'],
3     'max_depth' : range(2,32,1),
4     'min_samples_leaf' : range(1,10,1),
5     'min_samples_split': range(2,10,1),
6     'splitter' : ['best', 'random']
7 }
```

In [46]:



```
1 from sklearn.model_selection import GridSearchCV
2 grid_search = GridSearchCV(estimator=model,
3                             param_grid=grid_param,
4                             cv=5,
5                             n_jobs =2, verbose=1)
```

In [47]:



```
1 grid_search.fit(X_train,y_train)
```

Fitting 5 folds for each of 8640 candidates, totalling 43200 fits

Out[47]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=2,
             param_grid={'criterion': ['gini', 'entropy'],
                          'max_depth': range(2, 32),
                          'min_samples_leaf': range(1, 10),
                          'min_samples_split': range(2, 10),
                          'splitter': ['best', 'random']},
             verbose=1)
```

In [48]:



```
1 best_parameters = grid_search.best_params_
2 print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 16, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
```

In [49]:



```
1 grid_search.best_score_
```

Out[49]:

```
0.7936038320653704
```

In [51]:



```
1 clf = DecisionTreeClassifier(criterion = 'gini', max_depth = 25,
2                             min_samples_leaf= 1, min_samples_split= 2,
3                             splitter = 'best')
4 clf.fit(X_train,y_train)
```

Out[51]:

```
DecisionTreeClassifier(max_depth=25)
```

In [52]:

```
1 ac2_clf= clf.score(X_test ,y_test)
2 ac2_clf
```

Out[52]:

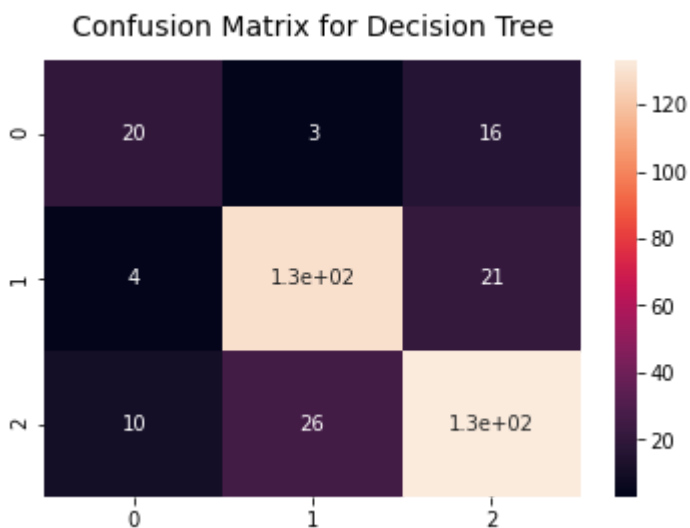
0.7790055248618785

In [53]:

```
1 from sklearn.metrics import confusion_matrix
2 y_pred_clf = clf.predict(X_test)
3 cf_matrix = confusion_matrix(y_test, y_pred_clf)
4 sns.heatmap(cf_matrix, annot=True)
5 plt.title("Confusion Matrix for Decision Tree", fontsize=14,
6           fontname="DejaVu Sans", y=1.03)
```

Out[53]:

Text(0.5, 1.03, 'Confusion Matrix for Decision Tree')



In [55]:

```
1 y_pred_rf = classifier.predict(X_test)
```

In [58]:

```
1 from sklearn.metrics import accuracy_score
2 ac_rf=accuracy_score(y_test,y_pred)
3 ac_rf
```

Out[58]:

0.787292817679558

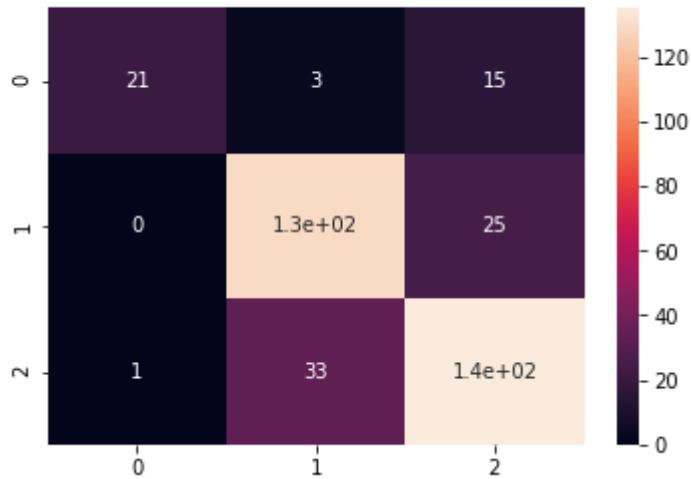
In [56]:

```
1 cf_matrix = confusion_matrix(y_test, y_pred_rf)
2 sns.heatmap(cf_matrix, annot=True)
3 plt.title("Confusion Matrix for RandomForest Classifier",
4           fontsize=14, fontname="DejaVu Sans", y=1.03)
```

Out[56]:

Text(0.5, 1.03, 'Confusion Matrix for RandomForest Classifier')

Confusion Matrix for RandomForest Classifier



In [59]:

```
1 plt.bar(['Decision Tree', 'Random Forest'], [ac2_clf, ac_rf])
2 plt.xlabel("Algorithms")
3 plt.ylabel("Accuracy")
4 plt.show()
```

