In [1]:

```python
# Data Manipulation
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#Scaling
from sklearn.preprocessing import StandardScaler

#Train Test Split
from sklearn.model_selection import train_test_split

# Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

#Evaluation
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

#Warnings
import warnings
warnings.filterwarnings('ignore')

from traitlets.config import get_config
c = get_config()
c.Exporter.filters = {'escape_html_keep_quotes': 'nbconvert.filters.strings.escape_
```

In [2]:

```python
# Reading the data
student = pd.read_csv('xAPI-Edu-Data.csv')

# Checking the head of the data (First 5 rows)
student.head()

# Show null counts and data types
student.info()
```

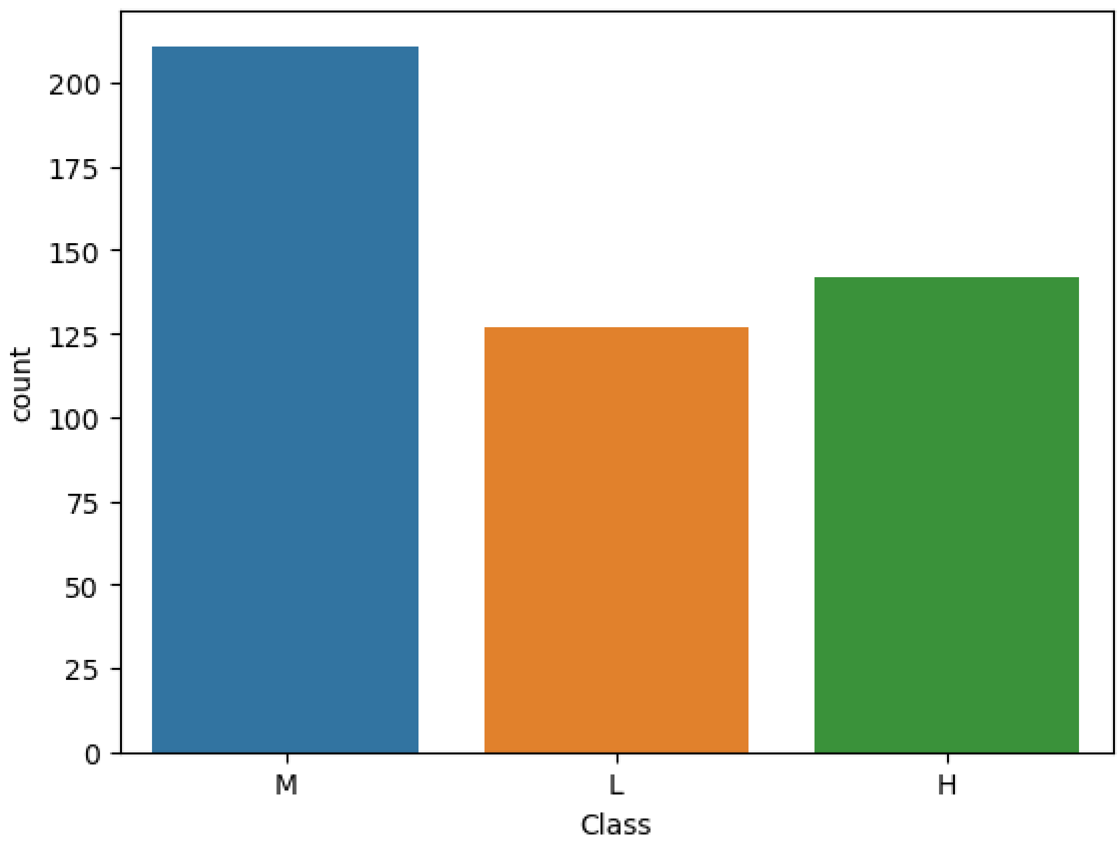
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   gender                    480 non-null    object
 1   NationalITy               480 non-null    object
 2   PlaceofBirth              480 non-null    object
 3   StageID                   480 non-null    object
 4   GradeID                   480 non-null    object
 5   SectionID                 480 non-null    object
 6   Topic                     480 non-null    object
 7   Semester                  480 non-null    object
 8   Relation                  480 non-null    object
 9   raisedhands               480 non-null    int64 
 10  VisITedResources          480 non-null    int64 
 11  AnnouncementsView         480 non-null    int64 
 12  Discussion                480 non-null    int64 
 13  ParentAnsweringSurvey     480 non-null    object
 14  ParentschoolSatisfaction  480 non-null    object
 15  StudentAbsenceDays        480 non-null    object
 16  Class                     480 non-null    object
dtypes: int64(4), object(13)
memory usage: 63.9+ KB
```

In [3]:

```
# Count of students of each class
sns.countplot(student['Class'])
```

Out[3]:

```
<AxesSubplot:xlabel='Class', ylabel='count'>
```

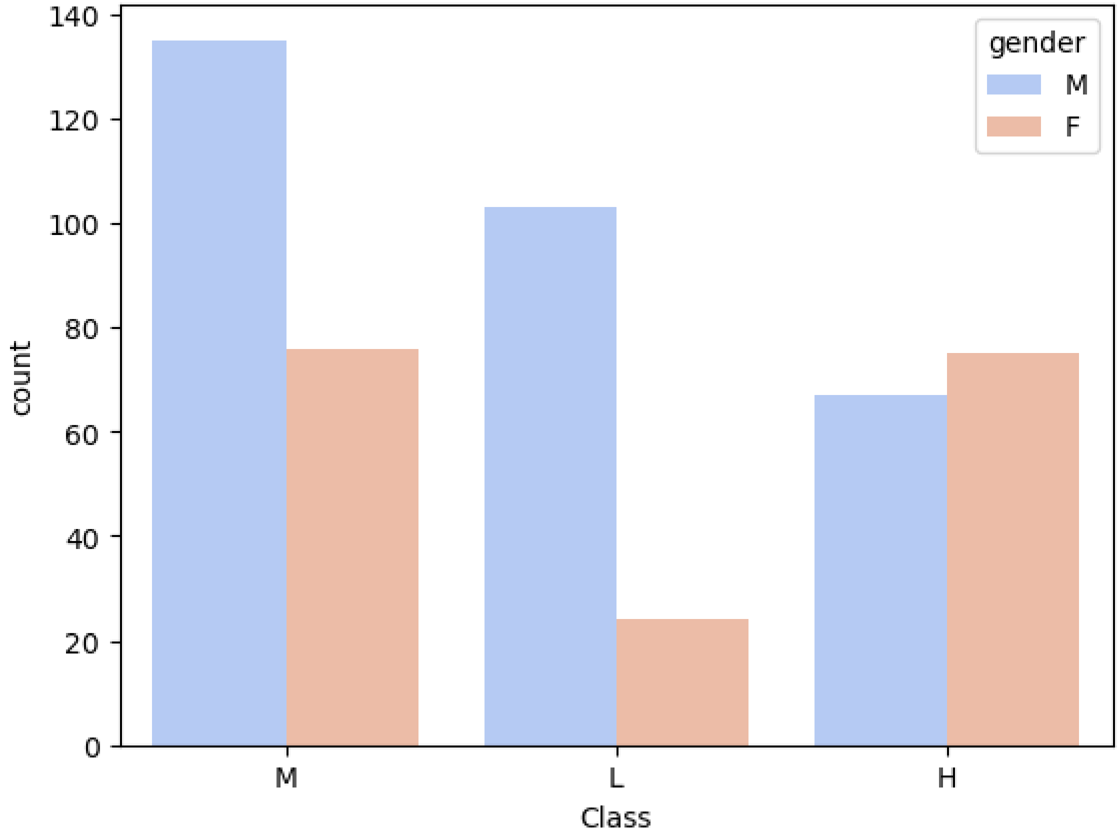


In [4]:

```
# Student class by gender
sns.countplot(x='Class',hue='gender',data=student,palette='coolwarm')
```

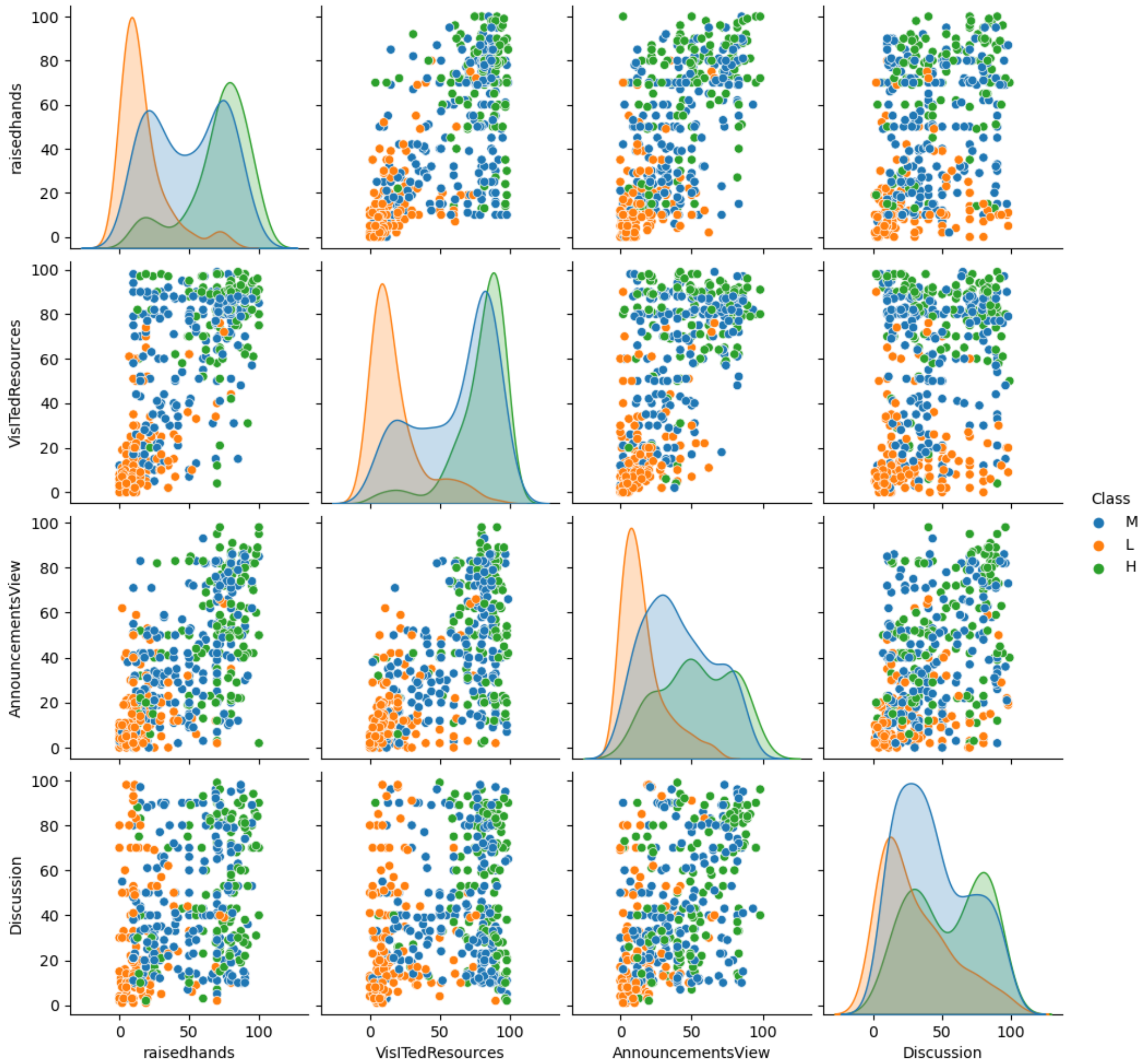
Out[4]:

```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



In [5]:
```
# Countplot based on the student Class
sns.pairplot(student,hue='Class')
```

Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x1d21114d910>
```



In [6]:

```
# Replacing categorical values to numerical

student['gender'].replace('M', 0,inplace=True)
student['gender'].replace('F', 1,inplace=True)

# Or we can use get_dummies to convert categorical values and concatenate them late
nat = pd.get_dummies(student['NationalITy'])
sid = pd.get_dummies(student['StageID'])
gid = pd.get_dummies(student['GradeID'])
secid = pd.get_dummies(student['SectionID'])
topic = pd.get_dummies(student['Topic'])
semester = pd.get_dummies(student['Semester'])
rel = pd.get_dummies(student['Relation'])
pas = pd.get_dummies(student['ParentAnsweringSurvey'])
pss = pd.get_dummies(student['ParentschoolSatisfaction'])
sab = pd.get_dummies(student['StudentAbsenceDays'])
```

In [7]:

```
#Drop useless columns & columns we need to replace with variables above
student.drop(['NationalITy','PlaceofBirth','PlaceofBirth','StageID','GradeID','Sec
              'Relation','ParentAnsweringSurvey','ParentschoolSatisfaction','Stude
```

In [8]:

```
# Concatenating the variables we created above
student = pd.concat([student,nat,sid,gid,secid,topic,semester,rel,pas,pss,sab],axi
```

In [9]:

```
# Check all the columns
student.columns
```

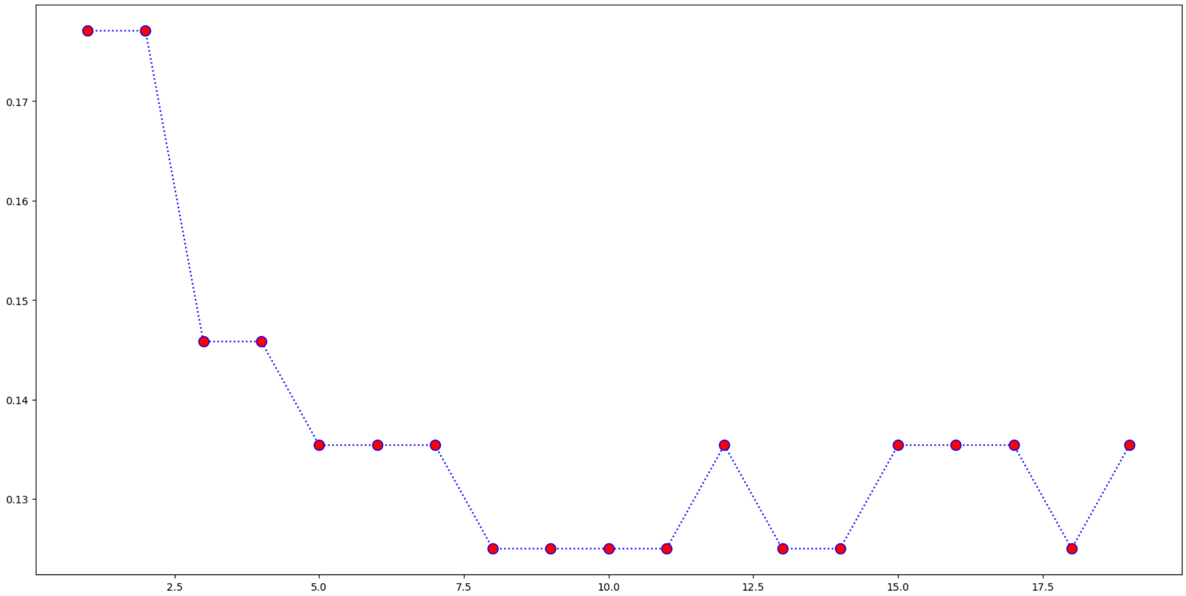
Out[9]:
```
Index(['gender', 'raisedhands', 'VisITedResources', 'AnnouncementsView',
       'Discussion', 'Class', 'Egypt', 'Iran', 'Iraq', 'Jordan', 'KW', 'Lybia',
       'Morocco', 'Palestine', 'SaudiArabia', 'Syria', 'Tunis', 'USA',
       'lebanon', 'venzuela', 'HighSchool', 'MiddleSchool', 'lowerlevel',
       'G-02', 'G-04', 'G-05', 'G-06', 'G-07', 'G-08', 'G-09', 'G-10', 'G-11',
       'G-12', 'A', 'B', 'C', 'Arabic', 'Biology', 'Chemistry', 'English',
       'French', 'Geology', 'History', 'IT', 'Math', 'Quran', 'Science',
       'Spanish', 'F', 'S', 'Father', 'Mum', 'No', 'Yes', 'Bad', 'Good',
       'Above-7', 'Under-7'],
      dtype='object')
```

In [10]:
```
Label = student['Class'] # Class is the value we want to predict

Features = student[['gender', 'raisedhands', 'VisITedResources', 'AnnouncementsVie
       'Discussion','Father', 'Mum', 'No', 'Yes', 'Bad', 'Good',
       'Above-7', 'Under-7']]


# We can also use the following method

#Features = student.drop(['feature a','feature b' .... 'feature n'],axis=1)
```

In [11]:
```
#Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
scaler.fit(Features)
scaled = scaler.transform(Features)
```

In [12]:
```
X = scaled
y = Label
# split the data to 20% test,80% train with random state=42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_s
```

In [13]:
```
err = [] # Array to save all error rates

for i in range(1,20): # Loop to try all error rates from 1 to 40
    rfe = RandomForestClassifier(n_estimators=i*10,random_state=42) # Create rfc w
    rfe.fit(X_train,y_train) # Fit the model
    errpred = rfe.predict(X_test) # Predict the value
    err.append(np.mean(errpred != y_test)) #Add the value to the array


# Plotting the value of estimators error rate using the method we created above to
plt.figure(figsize=(20,10)) # Size of the figure
plt.plot(range(1,20),err,color='blue',linestyle='dotted',marker='o',markerfacecolo
plt.title = 'Number of estimators VS Error Rates' #title
plt.xlabel = 'Estimators' #X label
plt.ylabel= 'Error Rate' # Y label
plt.show()
```

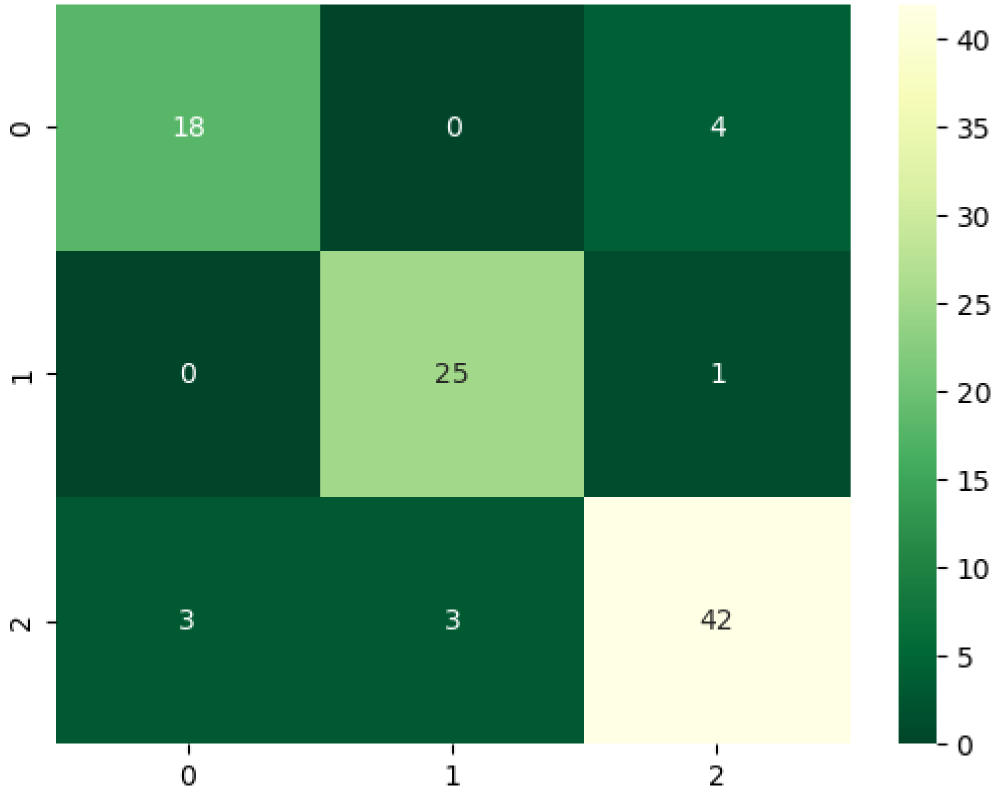


In [14]:
```
rfc = RandomForestClassifier(n_estimators=80,max_features='auto', max_depth=9,min_
                             min_samples_split=2,bootstrap=True, random_state = 42
rfc.fit(X_train,y_train)
rfcpred = rfc.predict(X_test)
```

In [15]:
```
print('Random Forest Classifier' + '\n')
print(classification_report(y_test,rfcpred))

print('\n')

print('Confusion matrix')
sns.heatmap(confusion_matrix(y_test,rfcpred),cmap='YlGn_r',annot=True,fmt='g')
```

```
Random Forest Classifier

              precision    recall  f1-score   support

           H       0.86      0.82      0.84        22
           L       0.89      0.96      0.93        26
           M       0.89      0.88      0.88        48

    accuracy                           0.89        96
   macro avg       0.88      0.88      0.88        96
weighted avg       0.89      0.89      0.88        96



Confusion matrix
```

Out[15]:
```
<AxesSubplot:>
```

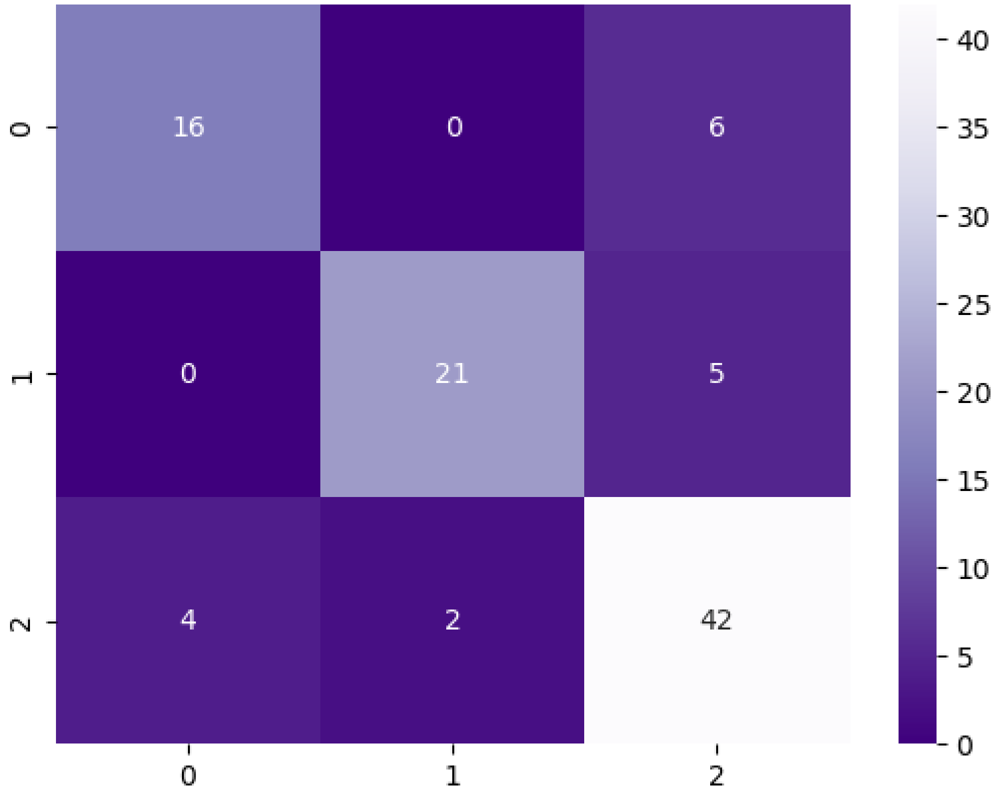


In [16]:
```
svc = SVC(C=100,random_state=42,gamma=1)
svc.fit(X_train,y_train)
svcpred = svc.predict(X_test)
```

In [17]:
```
print('Support Vector Classifier' + '\n')
print(classification_report(y_test,svcpred))

print('\n')

print('Confusion matrix')
sns.heatmap(confusion_matrix(y_test,svcpred),cmap='Purples_r',annot=True,fmt='g')
```

```
Support Vector Classifier

              precision    recall  f1-score   support

           H       0.80      0.73      0.76        22
           L       0.91      0.81      0.86        26
           M       0.79      0.88      0.83        48

    accuracy                           0.82        96
   macro avg       0.84      0.80      0.82        96
weighted avg       0.83      0.82      0.82        96



Confusion matrix
```

Out[17]:
```
<AxesSubplot:>
```

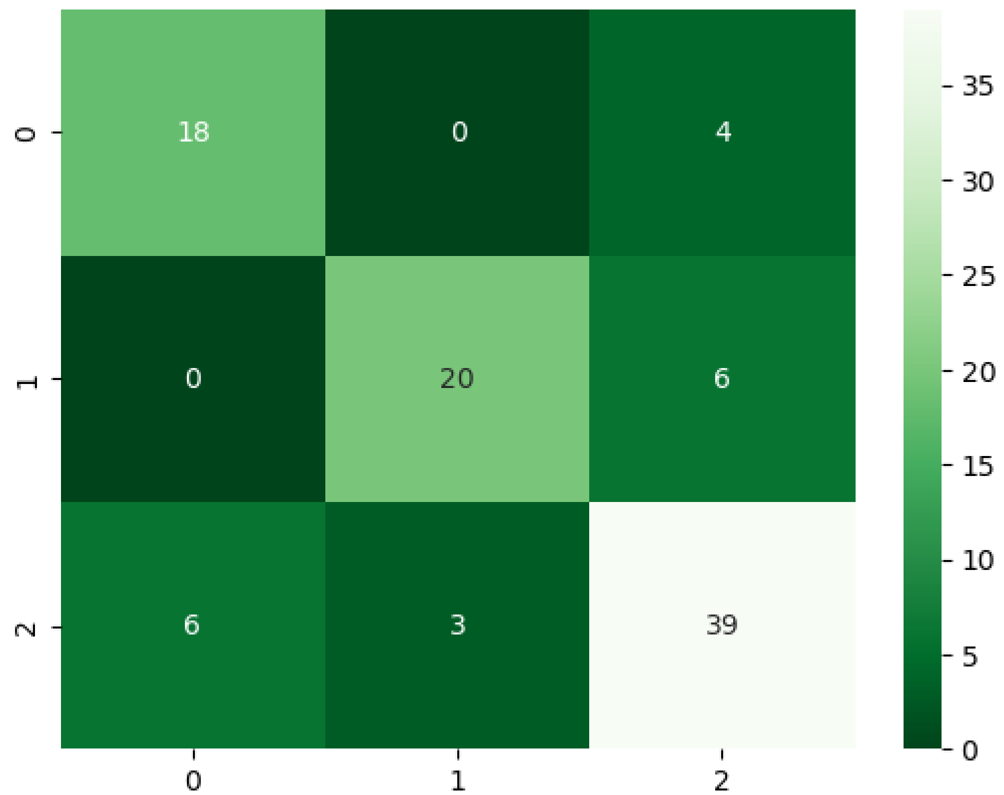


In [18]:
```
dt = DecisionTreeClassifier(max_depth=7, min_samples_split=4, min_samples_leaf=1,
dt.fit(X_train,y_train)
dtpred = dt.predict(X_test)
```

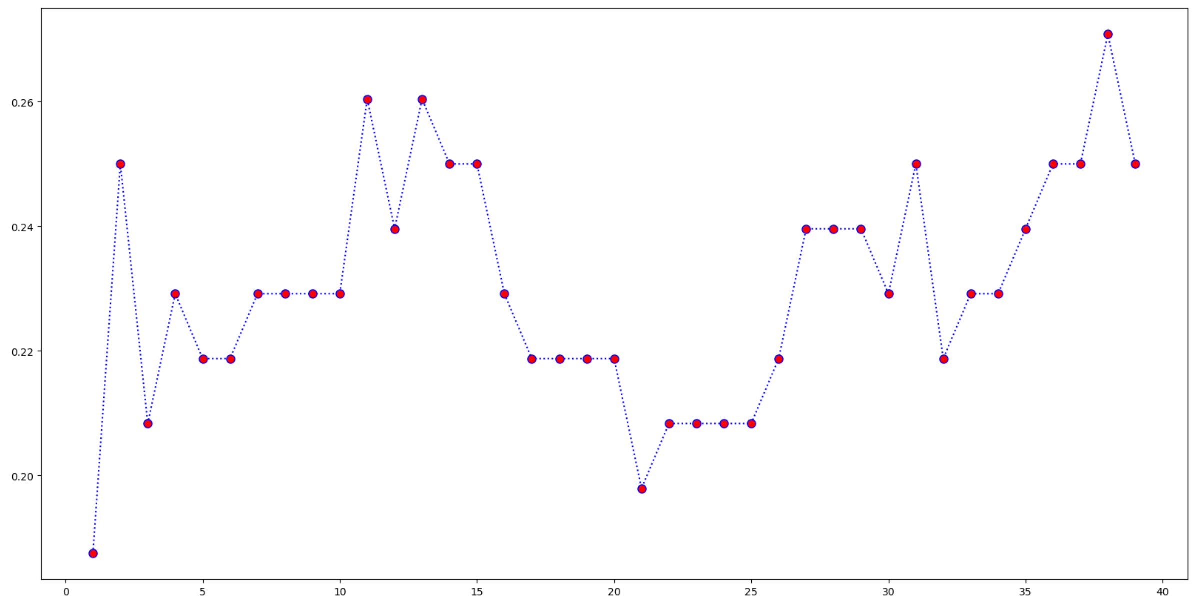
In [19]:
```
print('Desicion Tree Classifier' + '\n')
print(classification_report(y_test,dtpred))

print('\n')

print('Confusion matrix')
sns.heatmap(confusion_matrix(y_test,dtpred),cmap='Greens_r',annot=True,fmt='g')
```

```
Desicion Tree Classifier

              precision    recall  f1-score   support

           H       0.75      0.82      0.78        22
           L       0.87      0.77      0.82        26
           M       0.80      0.81      0.80        48

    accuracy                           0.80        96
   macro avg       0.81      0.80      0.80        96
weighted avg       0.81      0.80      0.80        96



Confusion matrix
<AxesSubplot:>
```

Out[19]:



In [20]:
```
err_rate = [] # Array to save all error rates

for i in range(1,40): # loop to try all error rates from 1 to 40
    knn = KNeighborsClassifier(n_neighbors=i) # create a knn object with number of
    knn.fit(X_train,y_train) # fit the model
    pred_i = knn.predict(X_test) # predict the value
    err_rate.append(np.mean(pred_i != y_test)) #add the value to the array

    # Plotting the value of k error rate using the method we created above to make
plt.figure(figsize=(20,10)) # size of the figure
plt.plot(range(1,40),err_rate,color='blue',linestyle='dotted',marker='o',markerfac
plt.title = 'K Values VS Error Rates' #title
plt.xlabel = 'K Value' #x label
plt.ylabel= 'Error Rate' # y label
plt.show()
```



In [21]:
```
# We didn't choose 1 as it's so sensetive to just rely on 1 neighbor
knn = KNeighborsClassifier(n_neighbors=21,p=10)
```

```
knn.fit(X_train,y_train)
knnpred = knn.predict(X_test)
```

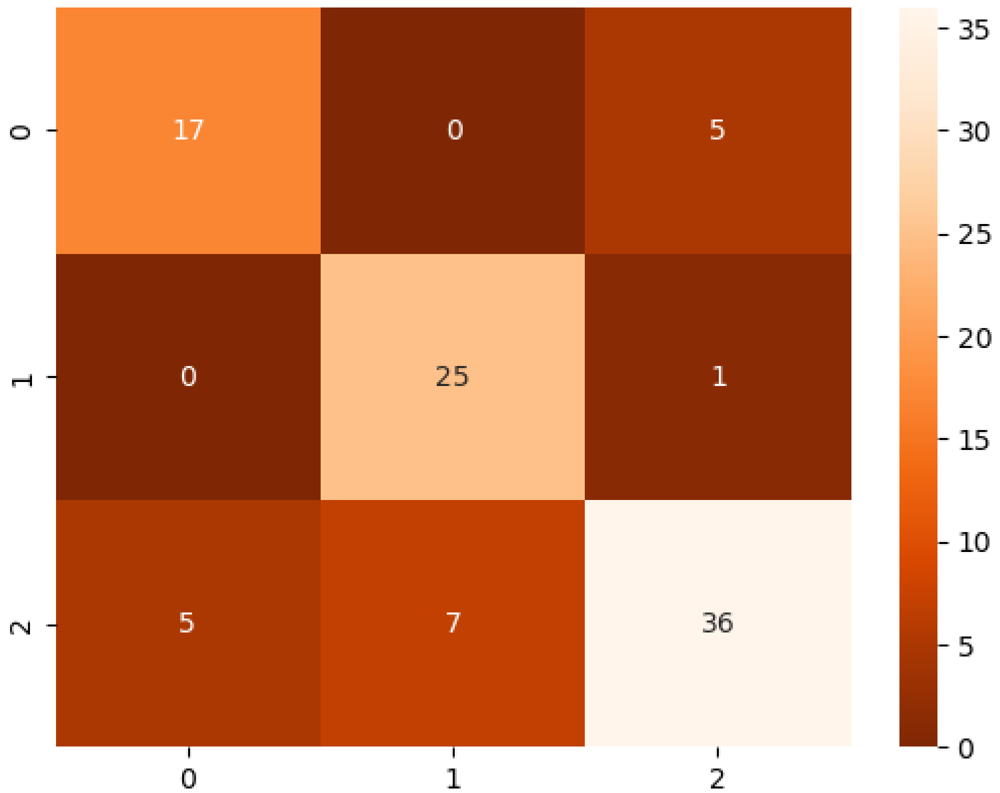
In [22]:

```
print('K Nearest Neighbours' + '\n')
print(classification_report(y_test,knnpred))

print('\n')

print('Confusion matrix')
sns.heatmap(confusion_matrix(y_test,knnpred),cmap='Oranges_r',annot=True,fmt='g')
```

```
K Nearest Neighbours

              precision    recall  f1-score   support

           H       0.77      0.77      0.77        22
           L       0.78      0.96      0.86        26
           M       0.86      0.75      0.80        48

    accuracy                           0.81        96
   macro avg       0.80      0.83      0.81        96
weighted avg       0.82      0.81      0.81        96



Confusion matrix
```

Out[22]:

```
<AxesSubplot:>
```



In [23]:

```
Lr = LogisticRegression(C=1,max_iter=30,multi_class='auto',random_state=1)
Lr.fit(X_train,y_train)
Lrpred = Lr.predict(X_test)
```

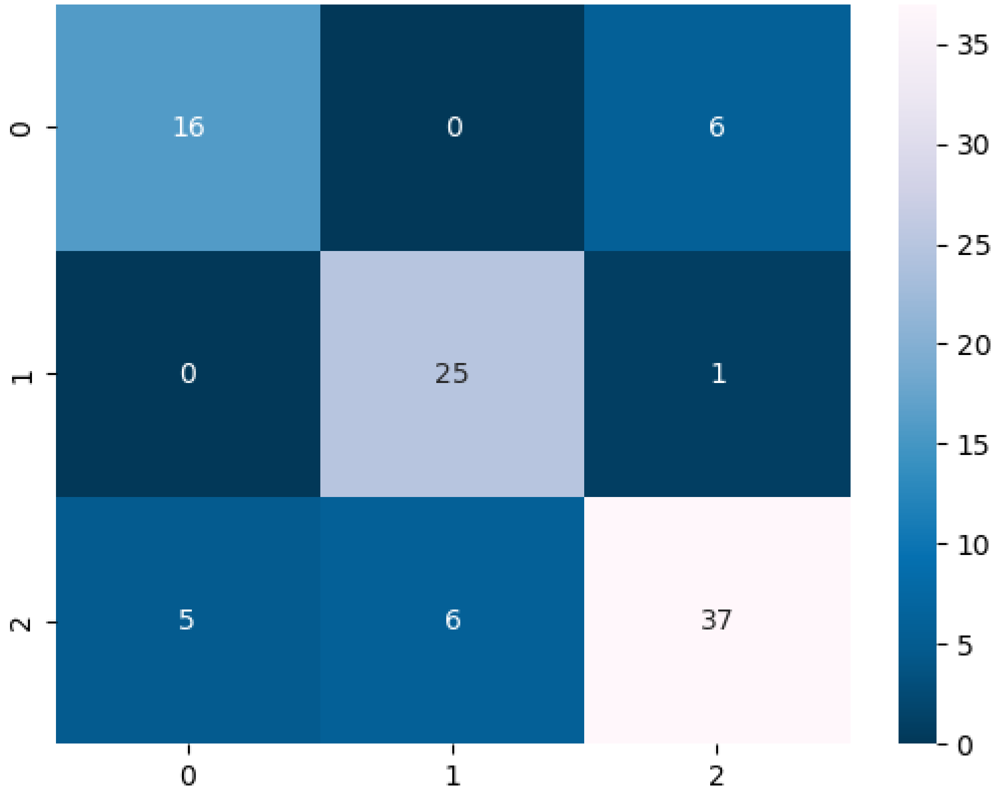
In [24]:

```
print('Logistic Regression' + '\n')
print(classification_report(y_test,Lrpred))

print('\n')
```

```
print('Confusion matrix')
sns.heatmap(confusion_matrix(y_test,Lrpred),cmap='PuBu_r',annot=True,fmt='g')
```

```
Logistic Regression

              precision    recall  f1-score   support

           H       0.76      0.73      0.74        22
           L       0.81      0.96      0.88        26
           M       0.84      0.77      0.80        48

    accuracy                           0.81        96
   macro avg       0.80      0.82      0.81        96
weighted avg       0.81      0.81      0.81        96


Confusion matrix
```

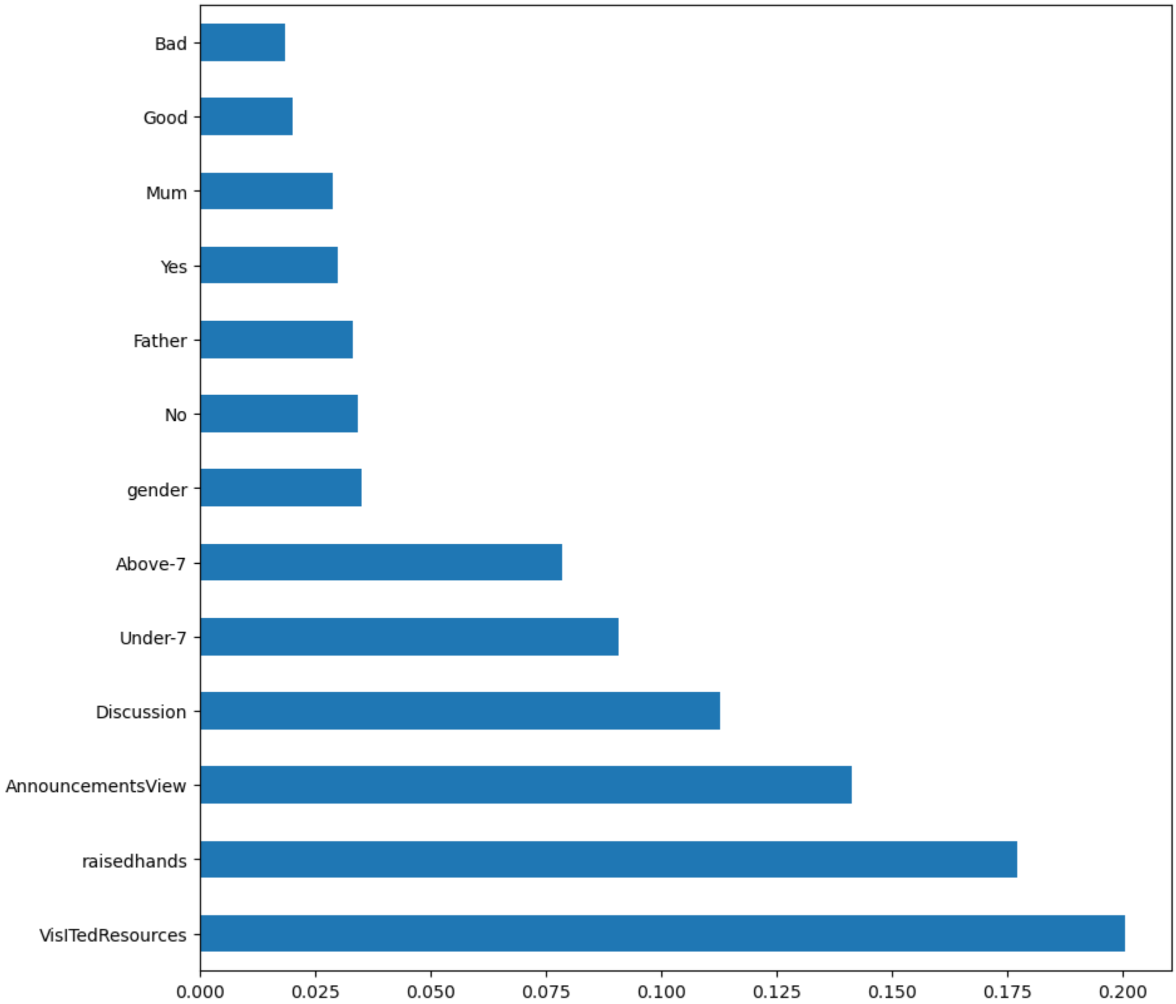
Out[24]: <AxesSubplot:>



In [25]:

```
# Show which features has the most effect on our results so we can modify and tune
# I used Random Forest Classifier to determine the feature importances

plt.figure(figsize=(10,10))
importance = pd.Series(rfc.feature_importances_,index=Features.columns)
importance.nlargest(15).plot(kind='barh')
```

Out[25]: <AxesSubplot:>



Bad
Good
Mum
Yes
Father
No
gender
Above-7
Under-7
Discussion
AnnouncementsView
raisedhands
VisITedResources
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200