sum

January 31, 2020

[155]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[157]:
```
from subprocess import check_output
if ('SAP' not in check_output(['ls', '../../Data']).decode('utf8')):
    raise FileNotFoundError
```

[179]:
```
df_numeric = pd.read_csv('../../Data/SAP_normal.csv')
df_numeric.head()
```

[179]:

	Unnamed: 0	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	Relation	raisedhands	VisITedResources	AnnouncementsView	Discussion	ParentAnsweringSurvey	ParentschoolSatisfaction	StudentAbsenceDays	Class
0	1	1	0.307692	0.307692	0.5	0.111111	0.0	0.636364	0	0	0.15	0.161616	0.020408	0.193878	1	1	1	1.0
1	2	1	0.307692	0.307692	0.5	0.111111	0.0	0.636364	0	0	0.20	0.202020	0.030612	0.244898	1	1	1	1.0
2	3	1	0.307692	0.307692	0.5	0.111111	0.0	0.636364	0	0	0.10	0.070707	0.000000	0.295918	0			
3	4	1	0.307692	0.307692	0.5	0.111111	0.0	0.636364	0	0	0.30	0.252525	0.051020	0.346939	0			
4	5	1	0.307692	0.307692	0.5	0.111111	0.0	0.636364	0	0	0.40	0.505051	0.122449	0.500000	0			

```
2                       0                   0    0.5
3                       0                   0    0.5
4                       0                   0    1.0
```

[158]:
```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(df_numeric.iloc[:, 1:])

explained_variance_ratio = pca.explained_variance_ratio_
cum_explained_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
lst = []
for i in range (0, len(explained_variance_ratio)):
  lst.append([i+1, round(explained_variance_ratio[i],6),␣
 ↪cum_explained_variance_ratio[i]])

pca_predictor = pd.DataFrame(lst)
pca_predictor.columns = ['Component', 'Explained Variance', 'Cumulative␣
 ↪Explained Variance']
pca_predictor
```

[158]:

	Component	Explained Variance	Cumulative Explained Variance
0	1	0.252565	0.252565
1	2	0.116844	0.369409
2	3	0.110032	0.479441
3	4	0.086880	0.566321
4	5	0.071549	0.637871
5	6	0.060843	0.698714
6	7	0.060061	0.758774
7	8	0.048387	0.807162
8	9	0.037112	0.844273
9	10	0.036350	0.880623
10	11	0.032636	0.913259
11	12	0.024941	0.938200
12	13	0.020759	0.958959
13	14	0.015929	0.974889
14	15	0.011472	0.986361
15	16	0.010301	0.996662
16	17	0.003338	1.000000

[159]:
```
df = pd.read_csv('../../Data/SAP.csv')

df['gender'] = df['gender'].astype('category').cat.codes
df['NationalITy'] = df['NationalITy'].astype('category').cat.codes
df['PlaceofBirth'] = df['PlaceofBirth'].astype('category').cat.codes
df['StageID'] = df['StageID'].astype('category').cat.codes
df['GradeID'] = df['GradeID'].astype('category').cat.codes
df['SectionID'] = df['SectionID'].astype('category').cat.codes
```

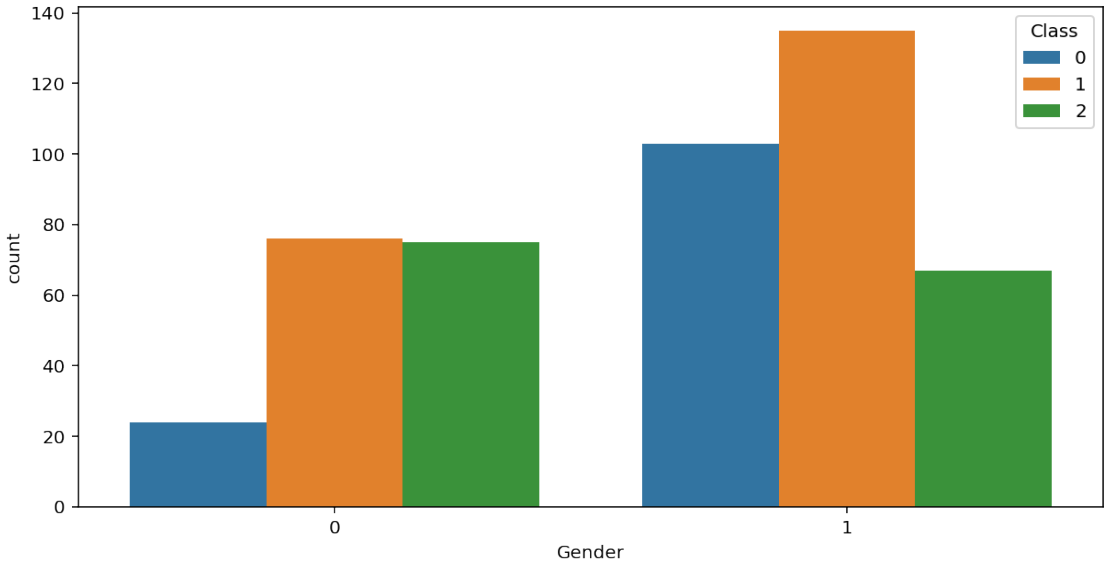
```
df['Topic'] = df['Topic'].astype('category').cat.codes
df['Semester'] = df['Semester'].astype('category').cat.codes
df['Relation'] = df['Relation'].astype('category').cat.codes
df['ParentAnsweringSurvey'] = df['ParentAnsweringSurvey'].astype('category').
 ↪cat.codes
df['ParentschoolSatisfaction'] = df['ParentschoolSatisfaction'].
 ↪astype('category').cat.codes
df['StudentAbsenceDays'] = df['StudentAbsenceDays'].astype('category').cat.codes
df["Class"][df["Class"] == "L"] = 0
df["Class"][df["Class"] == "M"] = 1
df["Class"][df["Class"] == "H"] = 2
df['Class'] = df['Class'].astype(int)

df[df.columns[:]].corr()['Class'][:]
```

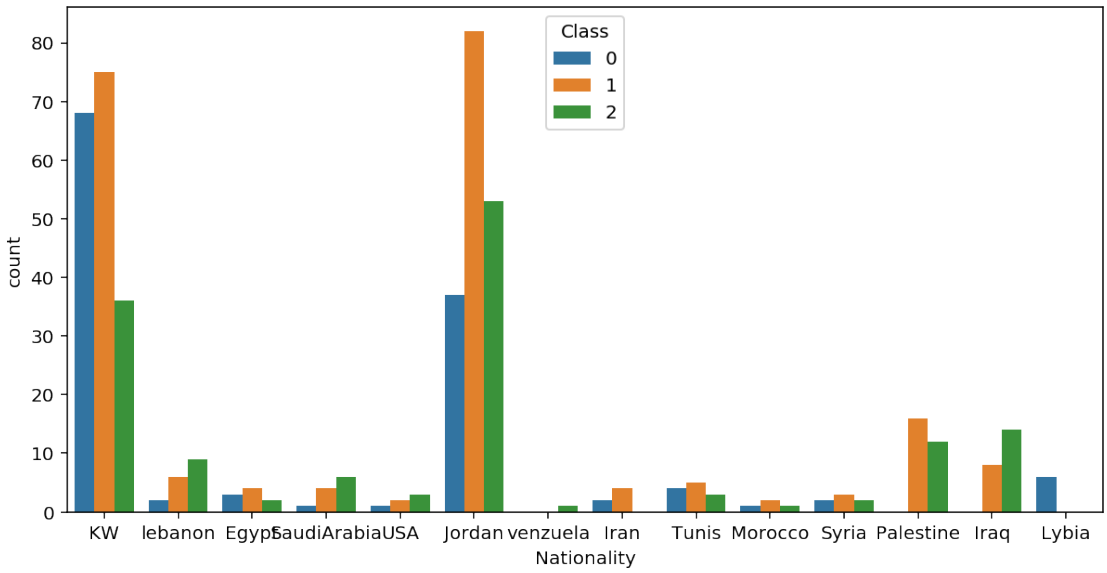
[159]:
```
gender                     -0.263490
NationalITy                 0.094640
PlaceofBirth                0.096571
StageID                    -0.083997
GradeID                     0.067217
SectionID                  -0.036850
Topic                      -0.093681
Semester                    0.126239
Relation                    0.401142
raisedhands                 0.646298
VisITedResources            0.677094
AnnouncementsView           0.527370
Discussion                  0.308183
ParentAnsweringSurvey       0.435495
ParentschoolSatisfaction    0.375901
StudentAbsenceDays          0.671312
Class                       1.000000
Name: Class, dtype: float64
```

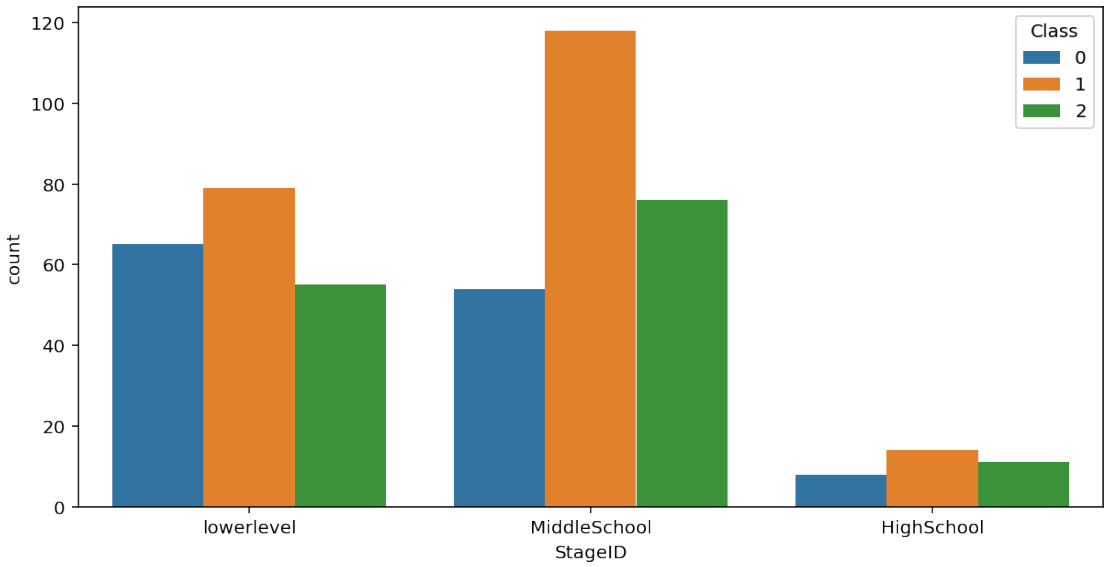
[160]:
```
df = pd.read_csv('../../Data/SAP.csv')
df.rename(index=str, columns={'gender':'Gender', 'NationalITy':
 ↪'Nationality','raisedhands':'RaisedHands', 'VisITedResources':
 ↪'VisitedResources'},inplace=True)

df["Class"][df["Class"] == "L"] = 0
df["Class"][df["Class"] == "M"] = 1
df["Class"][df["Class"] == "H"] = 2
df['Class'] = df['Class'].astype(int)

df['Gender'][df['Gender'] == "F"] = 0
df['Gender'][df['Gender'] == "M"] = 1
df['Gender'] = df['Gender'].astype(int)
```

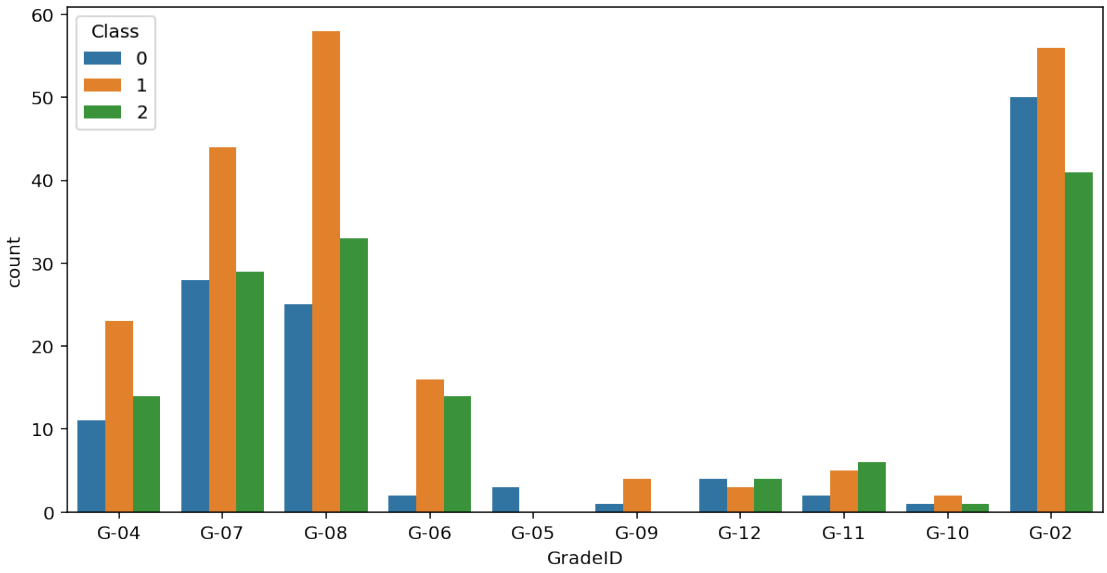
```
df['Semester'][df['Semester'] == "F"] = 0
df['Semester'][df['Semester'] == "S"] = 1
df['Semester'] = df['Semester'].astype(int)

df['Relation'][df['Relation'] == "Father"] = 0
df['Relation'][df['Relation'] == "Mum"] = 1
df['Relation'] = df['Relation'].astype(int)

df['StudentAbsenceDays'][df['StudentAbsenceDays'] == "Under-7"] = 0
df['StudentAbsenceDays'][df['StudentAbsenceDays'] == "Above-7"] = 1
df['StudentAbsenceDays'] = df['StudentAbsenceDays'].astype(int)

df['ParentschoolSatisfaction'][df['ParentschoolSatisfaction'] == "Bad"] = 0
df['ParentschoolSatisfaction'][df['ParentschoolSatisfaction'] == "Good"] = 1
df['ParentschoolSatisfaction'] = df['ParentschoolSatisfaction'].astype(int)

df['ParentAnsweringSurvey'][df['ParentAnsweringSurvey'] == "No"] = 0
df['ParentAnsweringSurvey'][df['ParentAnsweringSurvey'] == "Yes"] = 1
df['ParentAnsweringSurvey'] = df['ParentAnsweringSurvey'].astype(int)

# df.head()
```

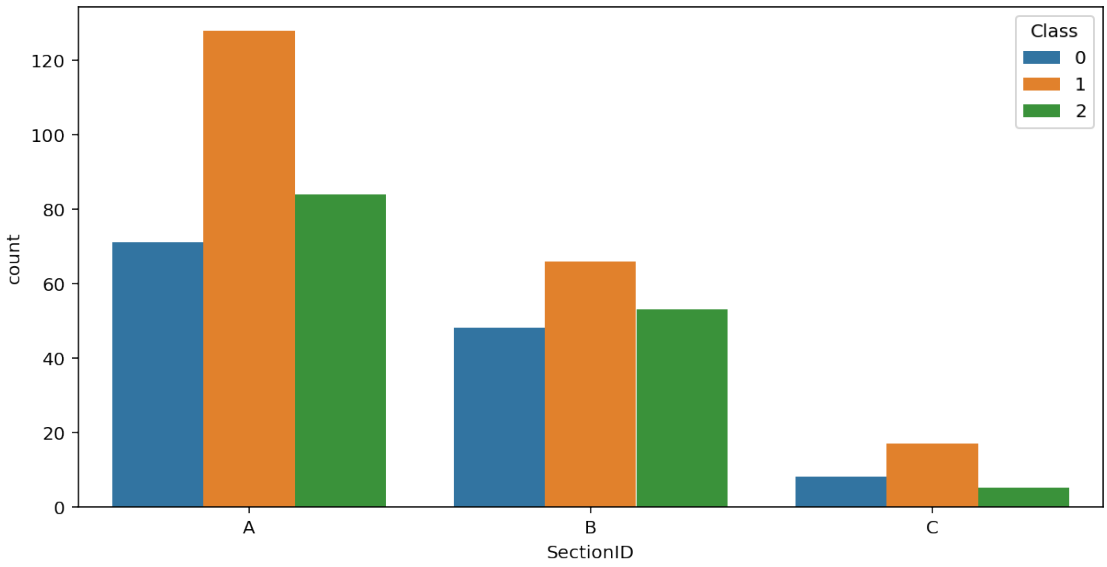
[161]:
```
for col in df.loc[:, ['Gender', 'Nationality', 'StageID', 'GradeID',␣
↪'SectionID', 'Topic', 'Semester']].columns:
    plt.figure(figsize = (10,5))
    ax = sns.countplot(x=df[col], hue=df.Class)
    plt.show()
```

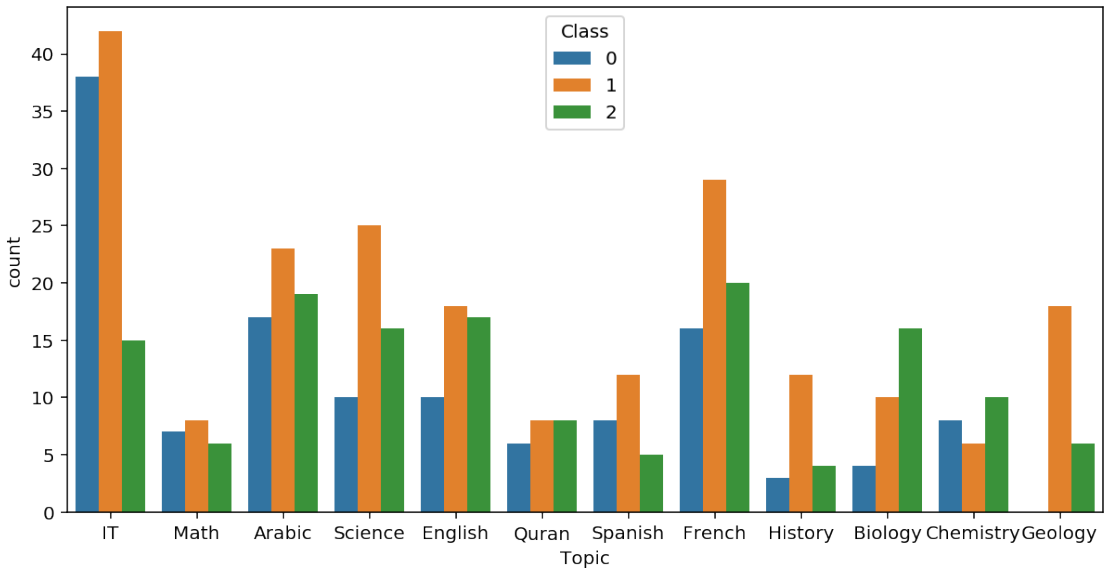


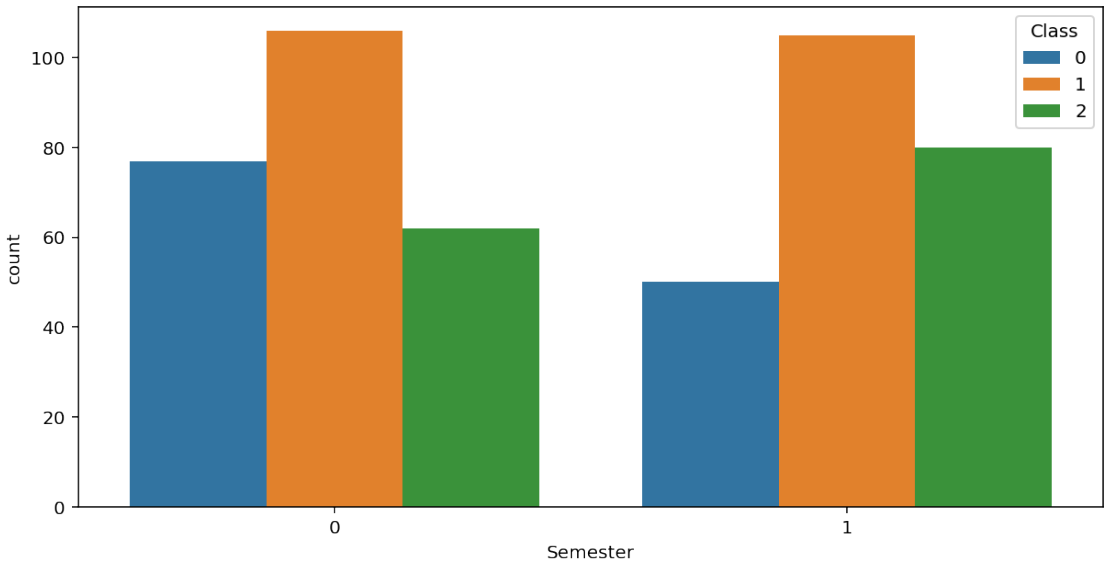

Class
0
1
2
count
Nationality
KW
lebanon
Egypt
SaudiArabia
USA
Jordan
venzuela
Iran
Tunis
Morocco
Syria
Palestine
Iraq
Lybia


Class
0
1
2
count
StageID
lowerlevel
MiddleSchool
HighSchool


Class
0
1
2
count
60
50
40
30
20
10
0
G-04
G-07
G-08
G-06
G-05
G-09
G-12
G-11
G-10
G-02
GradeID

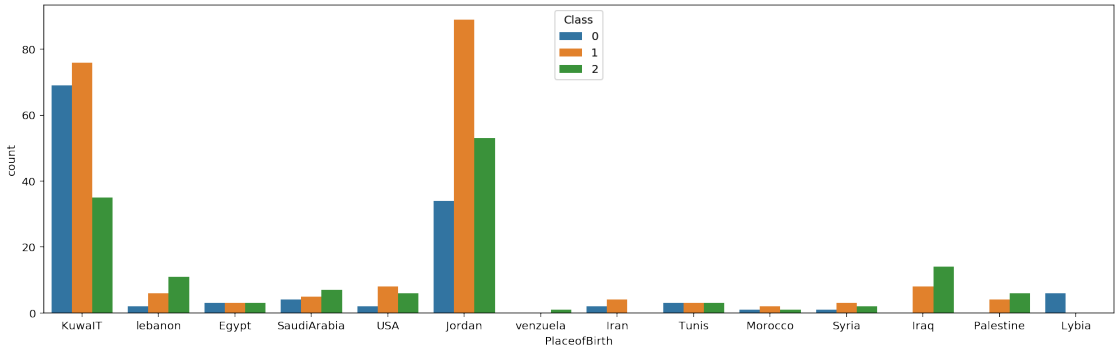

Class
0
1
2
count
120
100
80
60
40
20
0
A
B
C
SectionID





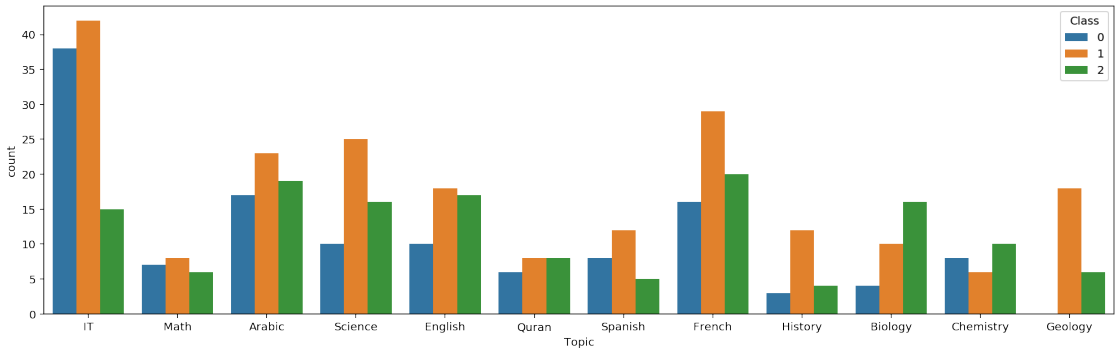
[162]:
```
plt.figure(figsize = (17,5))
sns.countplot(df.PlaceofBirth, hue=df.Class)
```

[162]: <matplotlib.axes._subplots.AxesSubplot at 0x7faa2580ecf8>



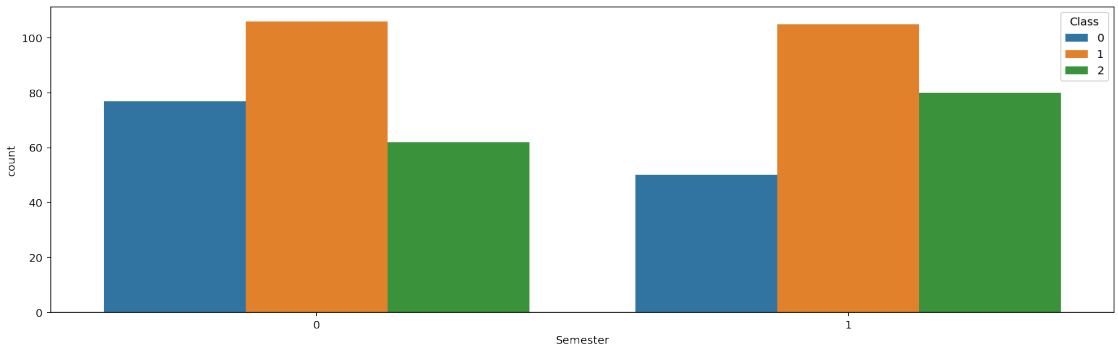
[163]:
```
plt.figure(figsize = (17,5))
sns.countplot(df.Topic, hue=df.Class)
```

[163]: <matplotlib.axes._subplots.AxesSubplot at 0x7faa25747ef0>



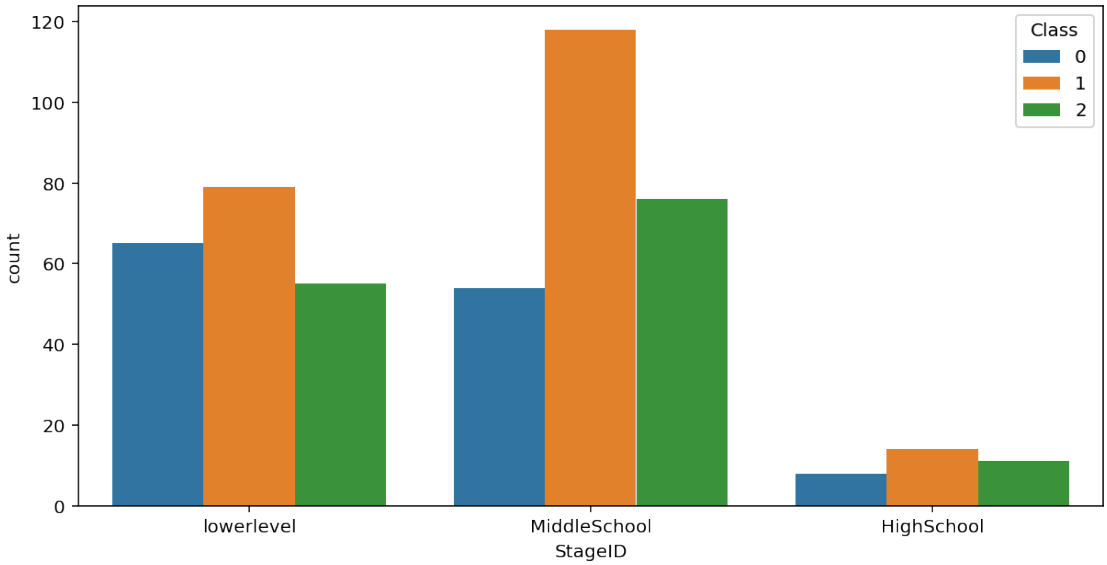
[164]:
```
plt.figure(figsize = (17,5))
sns.countplot(df.Semester, hue=df.Class)
```

[164]: <matplotlib.axes._subplots.AxesSubplot at 0x7faa256a35c0>

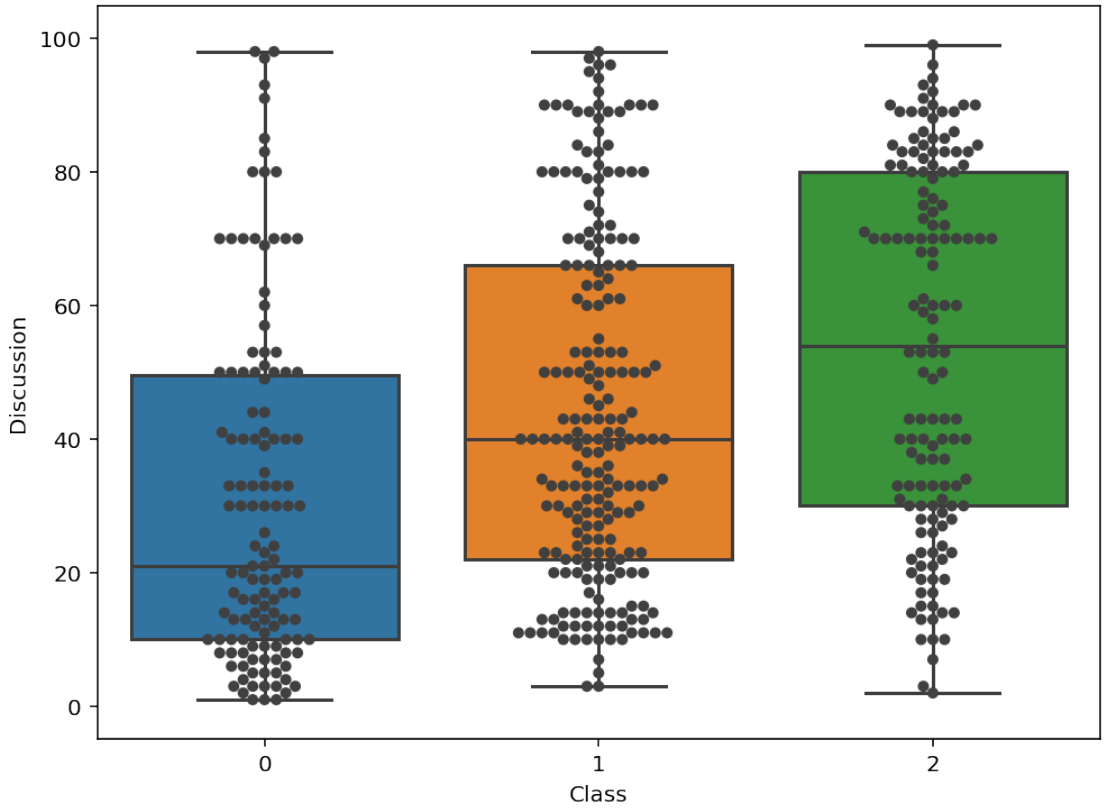


[165]:
```
plt.figure(figsize = (10,5))
sns.countplot(df.StageID, hue=df.Class)
```

[165]: <matplotlib.axes._subplots.AxesSubplot at 0x7faa256882e8>

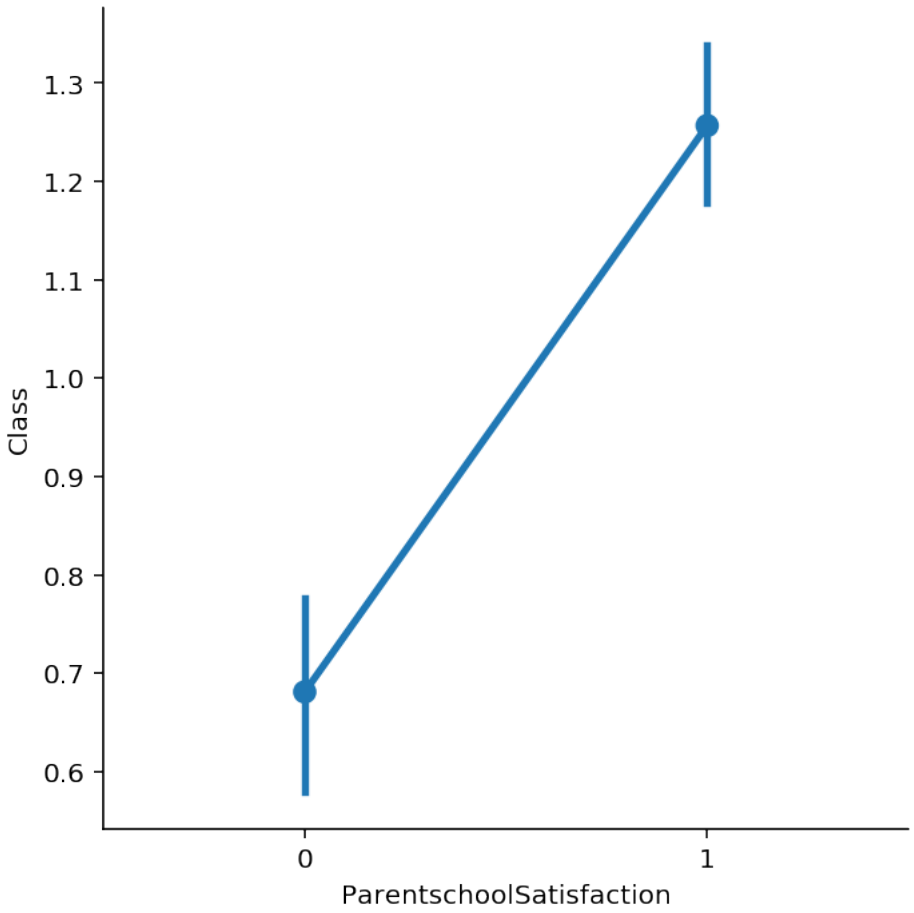


[166]:
```
plt.figure(figsize = (8,6))
ax = sns.boxplot(x='Class', y='Discussion', data=df)
ax = sns.swarmplot(x='Class', y = 'Discussion', data=df, color='.25')
plt.show()
```



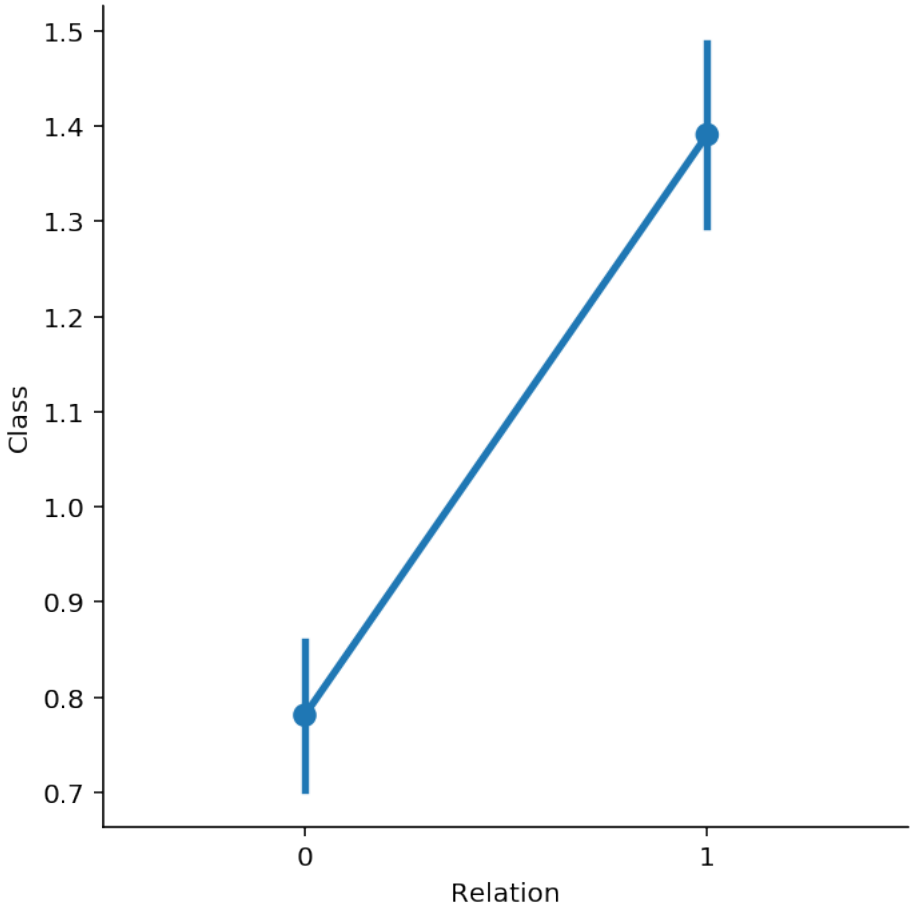
[167]:
```
sns.catplot('ParentschoolSatisfaction','Class', kind='point', data=df)
```

[167]: <seaborn.axisgrid.FacetGrid at 0x7faa25583208>

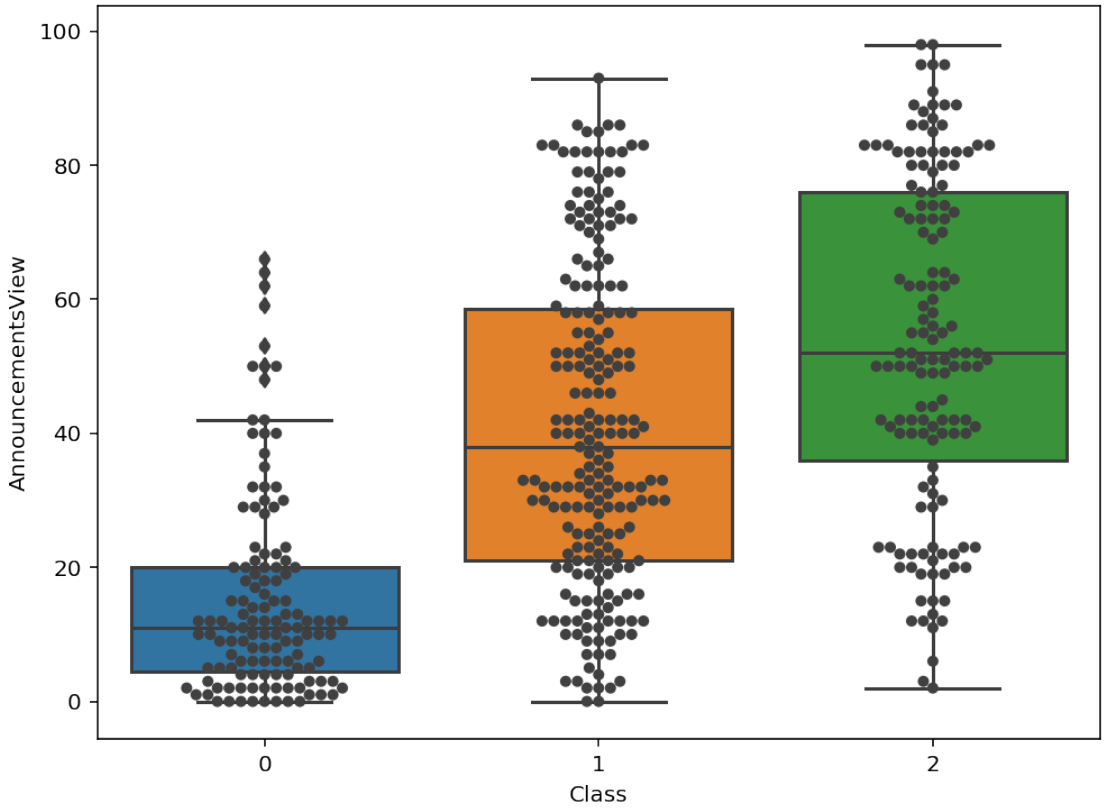


[168]:
```
sns.catplot('Relation','Class', kind='point', data=df)
```

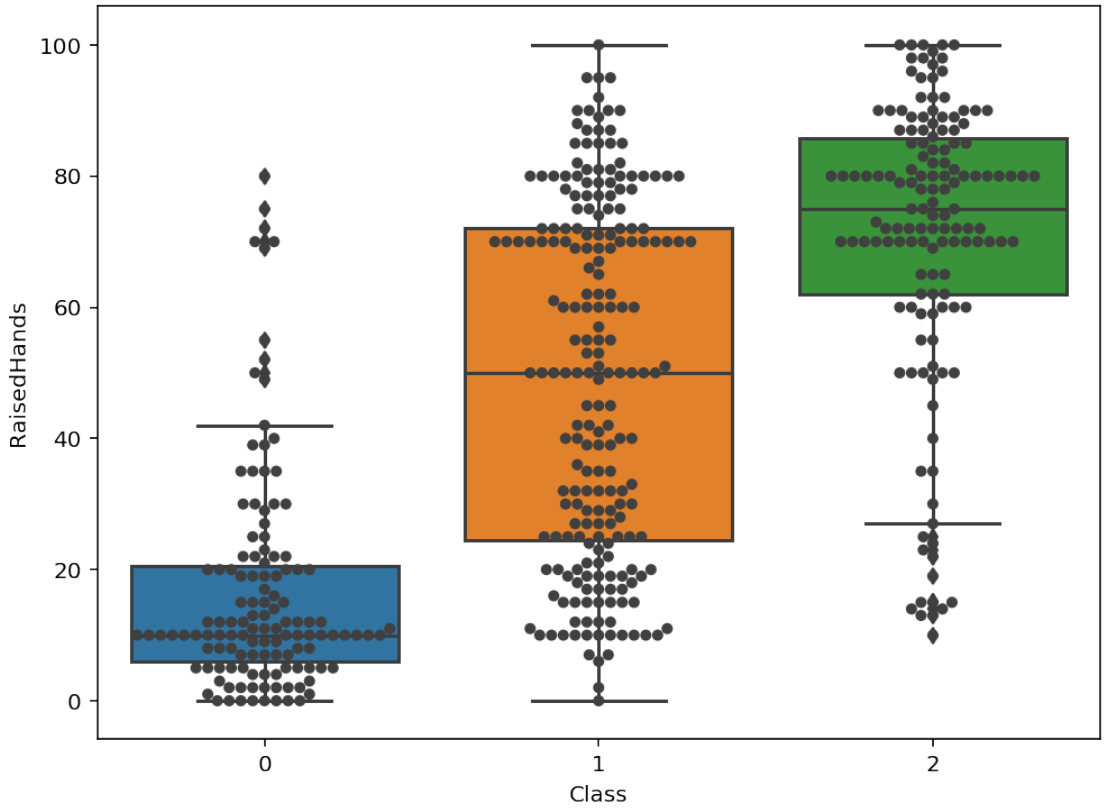
[168]:
```
<seaborn.axisgrid.FacetGrid at 0x7faa2568efd0>
```



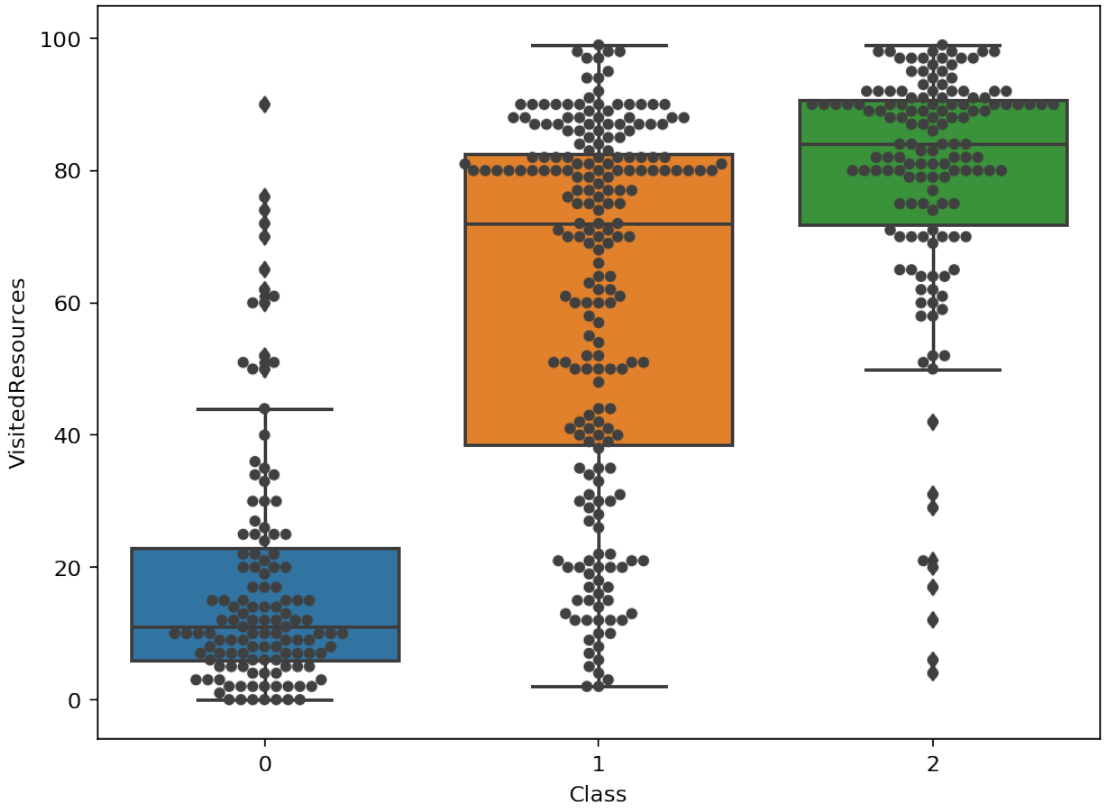
[169]:
```
plt.figure(figsize = (8,6))
ax = sns.swarmplot(x='Class', y='AnnouncementsView', data=df, color='.25')
ax = sns.boxplot(x='Class', y='AnnouncementsView', data=df)
plt.show()
```



[170]:
```
plt.figure(figsize = (8,6))
ax = sns.swarmplot(x='Class', y='RaisedHands', data=df, color='.25')
ax = sns.boxplot(x='Class', y='RaisedHands', data=df)
```

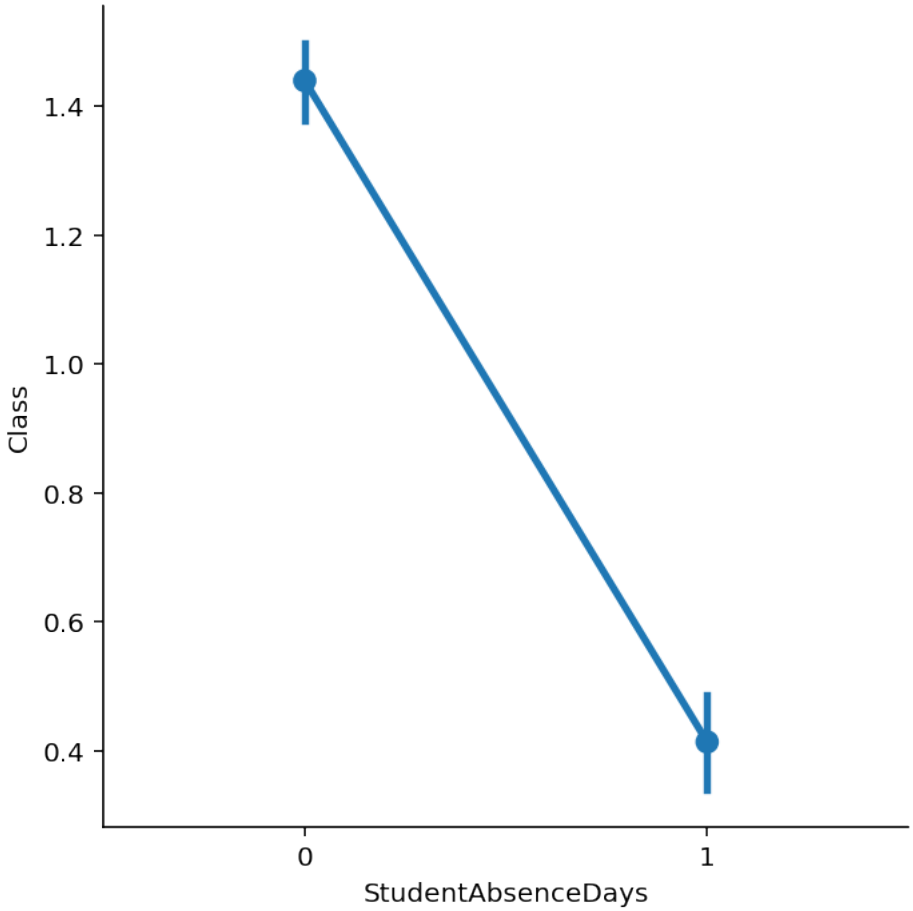


[171]:
```
plt.figure(figsize = (8,6))
ax = sns.boxplot(x='Class', y='VisitedResources', data=df)
ax = sns.swarmplot(x='Class', y = 'VisitedResources', data=df, color='.25')
```



[172]:
```
sns.catplot('StudentAbsenceDays','Class', kind='point', data=df)
```

[172]:
```
<seaborn.axisgrid.FacetGrid at 0x7fa9f3cbbda0>
```



[173]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1], df.iloc[:,␣
↪-1], test_size=.3, random_state=1)
```

[174]:
```
features = X_train[["Gender", "Discussion", "ParentAnsweringSurvey",␣
↪"ParentschoolSatisfaction", "Relation", "AnnouncementsView", "RaisedHands",␣
↪"VisitedResources", "StudentAbsenceDays"]].values

test_features = X_test[["Gender", "Discussion", "ParentAnsweringSurvey",␣
↪"ParentschoolSatisfaction", "Relation", "AnnouncementsView", "RaisedHands",␣
↪"VisitedResources", "StudentAbsenceDays"]].values
```

[175]:
```
from sklearn import tree

tree_clsfr_ovrfit = tree.DecisionTreeClassifier('entropy')
tree_clsfr_ovrfit = tree_clsfr_ovrfit.fit(features, y_train)
```

```
print(tree_clsfr_ovrfit.score(features, y_train))
print(tree_clsfr_ovrfit.score(test_features, y_test))
```

```
1.0
0.7013888888888888
```

[176]:
```
tree_clsfr = tree.DecisionTreeClassifier(criterion='entropy', max_depth = 10,␣
 ↪min_samples_split = 5, random_state = 5)
tree_clsfr = tree_clsfr.fit(features, y_train)

print(tree_clsfr.score(features, y_train))
print(tree_clsfr_ovrfit.score(test_features, y_test))
```

```
0.9375
0.7013888888888888
```

[177]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_forest = RandomForestClassifier(criterion='entropy', max_depth = 7,␣
 ↪min_samples_split=5, n_estimators = 100, random_state = 4)
rnd_forest = rnd_forest.fit(features, y_train)

print(rnd_forest.score(features, y_train))
print(rnd_forest.score(test_features, y_test))
```

```
0.9255952380952381
0.7708333333333334
```

[178]:
```
from sklearn.metrics import classification_report, accuracy_score
print('============= Decision Tree ==============')
print(classification_report(y_test, tree_clsfr.predict(test_features),␣
 ↪target_names=['Low', 'Mid', 'High']))

print('============= Random Forest ==============')
print(classification_report(y_test, rnd_forest.predict(test_features),␣
 ↪target_names=['Low', 'Mid', 'High']))
```

```
============= Decision Tree ==============
              precision    recall  f1-score   support

         Low       0.78      0.88      0.83        33
         Mid       0.62      0.68      0.65        56
        High       0.76      0.64      0.69        55

    accuracy                           0.71       144
   macro avg       0.72      0.73      0.72       144
weighted avg       0.71      0.71      0.71       144
```

```
============= Random Forest ==============
              precision    recall  f1-score   support

         Low       0.85      0.85      0.85        33
         Mid       0.69      0.73      0.71        56
        High       0.81      0.76      0.79        55

    accuracy                           0.77       144
   macro avg       0.78      0.78      0.78       144
weighted avg       0.77      0.77      0.77       144
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
features = X_train[["Gender", "Discussion", "ParentAnsweringSurvey",␣
 ↪"ParentschoolSatisfaction", "Relation", "AnnouncementsView", "RaisedHands",␣
 ↪"VisitedResources", "StudentAbsenceDays"]].values
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

[ ]:
```
data = pd.read_csv('../../Data/SAP_numeric.csv')
data.head()

# Y = data['Class'].values
# data2 = data.drop(['Class'],axis=1)
# x = data2.values

x, Y = data.drop('Class', axis=1), np.asarray(data['Class'])
Xtrain, Xtest, ytrain, ytest = train_test_split(x, Y, test_size=0.20)
```

[ ]:
```
classifiers = [('KNeighborsClassifier', KNeighborsClassifier(n_jobs=-1)),
               ('GaussianNB', GaussianNB()),
```

```
                ('BernoulliNB', BernoulliNB())
               ]

allscores = []
```

[ ]:
```
data = pd.read_csv('../../Data/SAP_numeric.csv')
data.head()

classifiers = [('BernoulliNB', BernoulliNB()),
               ('GaussianNB', GaussianNB()),
               ('DecisionTreeClassifier', DecisionTreeClassifier()),
               ('AdaBoostClassifier', AdaBoostClassifier())
              ]

allscores = []

for name, classifier in classifiers:
    accuracy=  0.0
    f1_total = np.array([])
    scores = np.array([])
    for i in range(20):
        x, Y = data.drop('Class', axis=1), np.asarray(data['Class'])
        Xtrain, Xtest, ytrain, ytest = train_test_split(x, Y, test_size=0.30)

        scaler = StandardScaler()
        Xtrain_scaled = scaler.fit_transform(Xtrain)
        Xtest_scaled = scaler.transform(Xtest)
        #Calculate the class prior probabilities for each continent
        classifier.fit(Xtrain_scaled, ytrain)
        #Make a prediction for the test data
        ypred = classifier.predict(Xtest_scaled)
        #Calculate the accuracy of the data
        scores = np.append(scores, accuracy_score(ytest, ypred))
        accuracy = accuracy + accuracy_score(ytest, ypred)
        f1_total = np.append(f1_total, np.average(f1_score(ytest, ypred,␣
 ↪average=None)))

    new_data = [(name, score) for score in scores]
    allscores.extend(new_data)
    accuracy_avg = accuracy / 20
    f1_avg = np.average(f1_total)
    print(name+" Mean accuracy %.2f and mean f1_score %.2f after 20␣
 ↪Iteration\n" % (accuracy_avg, f1_avg))

temp = pd.DataFrame(allscores, columns=['classifier', 'score'])
#sns.violinplot('classifier', 'score', data=temp, inner=None, linewidth=0.3)
plt.figure(figsize=(15,10))
```

```
sns.catplot(x='classifier',
                y="score",
                data=temp,
                saturation=1,
                kind="box",
                ci=None,
                aspect=1,
                linewidth=1,
                height = 10)
locs, labels = plt.xticks()
plt.setp(labels, rotation=90)
```

[ ]:
```
svm = SVC(kernel='linear', C=2.0, random_state=0)
svm.fit(Xtrain, ytrain)

ypred = svm.predict(Xtest)
print('Misclassified samples: %d' % (ytest != ypred).sum())
print('Accuracy: %.2f' % accuracy_score(ytest, ypred))
print(classification_report(ytest, ypred))
```

[ ]:

[ ]:

[ ]:
```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.metrics.cluster import adjusted_rand_score
```

[ ]:
```
df = pd.read_csv('../../Data/SAP_numeric.csv')
```

[ ]:
```
df.head()
print(df.shape)
df.isnull().sum()
```

[ ]:
```
# Imports
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
%config InlineBackend.figure_format='retina'# Load in the data
```

```
df = pd.read_csv('../../Data/SAP_numeric.csv')# Standardize the data to have a␣
↪mean of ~0 and a variance of 1
X_std = StandardScaler().fit_transform(df)# Create a PCA instance: pca
pca = PCA(n_components=18)
principalComponents = pca.fit_transform(X_std)# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```

[ ]:
```
plt.scatter(PCA_components[0], PCA_components[1], alpha=.1, color='black')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
```

[ ]:
```
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

[ ]:
```
kmeans3 = KMeans(n_clusters=3)
y_kmeans3 = kmeans3.fit_predict(PCA_components)
```

[ ]:
```
plt.scatter(PCA_components[0], PCA_components[1],c=y_kmeans3)
```

[ ]:
```
plt.scatter(PCA_components[0], PCA_components[1],c=df["Class"])
```

[ ]:
```
df["Class"]
```

[ ]:
```
from sklearn.cluster import DBSCAN
from sklearn import metrics

db = DBSCAN(eps=3.1, min_samples=7)
```

```
model = db.fit(PCA_components)
y_db_pred = db.fit_predict(PCA_components)

plt.scatter(PCA_components[0], PCA_components[1],c=y_db_pred)
plt.title("DBSAN")
labels = model.labels_
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print("number are clusters: ",n_clusters)
```

[ ]:
```
from sklearn.cluster import AgglomerativeClustering

hier = AgglomerativeClustering(n_clusters=n_clusters)
y_hier_pred = hier.fit_predict(PCA_components)
plt.scatter(PCA_components[0], PCA_components[1],c=y_hier_pred)
plt.title("Hierarchical")
```

[ ]:
```

```

[ ]:
```
population[1,2:8] = np.array([5,5,5,5,5])
```

[ ]:
```
print(population[1,1:20])
```

[ ]:
```

```