Imports

In [13]:

```
# Install tpot on the server
!pip install tpot
```

```
Requirement already satisfied: tpot in /usr/local/lib/python3.6/dist-packages (0.11.5)
Requirement already satisfied: deap>=1.2 in /usr/local/lib/python3.6/dist-packages (from tpot) (1.3.
1)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.6/dist-packages (from tpot) (1
.4.1)
Requirement already satisfied: stopit>=1.1.1 in /usr/local/lib/python3.6/dist-packages (from tpot) (
1.1.2)
Requirement already satisfied: update-checker>=0.16 in /usr/local/lib/python3.6/dist-packages (from
tpot) (0.17)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.6/dist-packages (from tpot)
(1.0.5)
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.6/dist-packages (from tpot) (
1.18.5)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.6/dist-packages (from tpot)
(0.15.1)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.6/dist-packages (from tpot) (4
.41.1)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.6/dist-packages (from
tpot) (0.22.2.post1)
Requirement already satisfied: requests>=2.3.0 in /usr/local/lib/python3.6/dist-packages (from updat
e-checker>=0.16->tpot) (2.23.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=
0.24.2->tpot) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-packages (fro
m pandas>=0.24.2->tpot) (2.8.1)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from req
uests>=2.3.0->update-checker>=0.16->tpot) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests
>=2.3.0->update-checker>=0.16->tpot) (2.9)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/d
ist-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from re
quests>=2.3.0->update-checker>=0.16->tpot) (2020.6.20)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateu
til>=2.6.1->pandas>=0.24.2->tpot) (1.12.0)
```

In [14]:

```
#  General
from os.path import join
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Utility
from scipy import stats
from scipy.stats import norm
from sklearn.pipeline import make_pipeline
import warnings
from itertools import cycle

# Preprocessing
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import learning_curve,ShuffleSplit,validation_curve, train_test_split
from sklearn.model_selection import GridSearchCV,learning_curve
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import cross_val_score, cross_validate, KFold

# Models
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from tpot import TPOTClassifier
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier


# Metrics
from sklearn.metrics import classification_report, confusion_matrix,f1_score,make_scorer,accuracy_score,roc_curve
, auc, accuracy_score
from sklearn.metrics import r2_score
from sklearn.metrics import make_scorer, accuracy_score

# Graphs
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap
from mlxtend.plotting import plot_decision_regions


warnings.filterwarnings('ignore')
%matplotlib inline
```

Retrieve dataset

In [15]:

```
DS_URL = "https://raw.githubusercontent.com/clintonyeb/ml-dataset/master/BEPS.csv"
FIG_SIZE=(12, 6)
```

In [16]:

```
beps = pd.read_csv(DS_URL, names=["id", "vote", "age", "nat_cond", "hhold_cond", "labor_lead_assmnt", "cons_lead_assmnt", "democ_lead_assmnt", "euro_intg_attud", "political_knowledge", "gender"], index_col="id", header=0)
beps.head(10)
```

Out[16]:

	vote	age	nat_cond	hhold_cond	labor_lead_assmnt	cons_lead_assmnt	democ_lead_assmnt	euro_intg_attud	po
id									
1	Liberal Democrat	43	3	3	4	1	4	2	
2	Labour	36	4	4	4	4	4	5	
3	Labour	35	4	4	5	2	3	3	
4	Labour	24	4	2	2	1	3	4	
5	Labour	41	2	2	1	1	4	6	
6	Labour	47	3	4	4	4	2	4	
7	Liberal Democrat	57	2	2	4	4	2	11	
8	Labour	77	3	4	4	1	4	1	
9	Labour	39	3	3	4	4	4	11	
10	Labour	70	3	2	5	1	1	11	

Exploratory Data Analysis (EDA)

We are using British Election Panel Study (https:/https://vincentarelbundock.github.io/Rdatasets/doc/carData/BEPS.html/) dataset.

Description
These data are drawn from the 1997-2001 British Election Panel Study (BEPS).

Format
A data frame with 1525 observations on the following 10 variables.

vote (vote)
Party choice: Conservative, Labour, or Liberal Democrat

age (age)
in years

economic.cond.national (nat_cond)
Assessment of current national economic conditions, 1 to 5.

economic.cond.household (hhold_cond)
Assessment of current household economic conditions, 1 to 5.

Blair (labor_lead_assmnt)
Assessment of the Labour leader, 1 to 5.

Hague (cons_lead_assmnt)
Assessment of the Conservative leader, 1 to 5.

Kennedy (democ_lead_assmnt)
Assessment of the leader of the Liberal Democrats, 1 to 5.

Europe (euro_intg_attud)
an 11-point scale that measures respondents' attitudes toward European integration. High scores represent ‘Eurosceptic’ sentiment.

political.knowledge (political_knowledge)
Knowledge of parties' positions on European integration, 0 to 3.

gender (gender)
female or male.

References
J. Fox and R. Andersen (2006) Effect displays for multinomial and proportional-odds logit models. Sociological Methodology 36, 225–255.

In [17]:

```
print("Number of records: ", len(beps))
print("Shape: ", beps.shape)
# Checks if there are any missing values
print("\nMissing data?")
beps.isnull().sum()
```

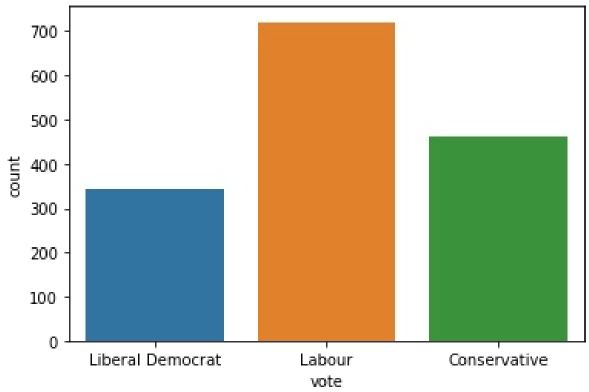
```
Number of records:  1525
Shape:  (1525, 10)

Missing data?
```

Out[17]:

```
vote                       0
age                        0
nat_cond                   0
hhold_cond                 0
labor_lead_assmnt          0
cons_lead_assmnt           0
democ_lead_assmnt          0
euro_intg_attud            0
political_knowledge        0
gender                     0
dtype: int64
```

In [18]:

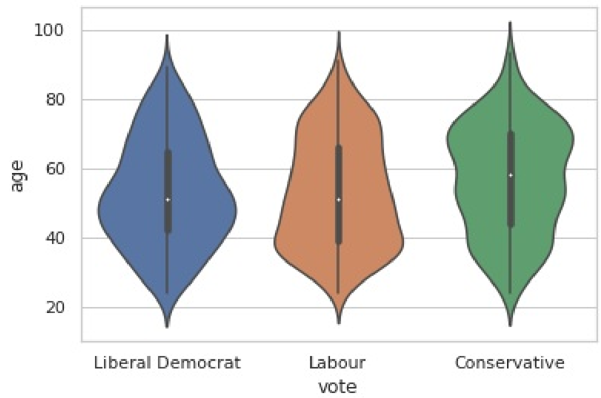
```
sns.countplot(x="vote", data=beps);
```



The Labor party won that election. This might be the reason why it's more represented here!

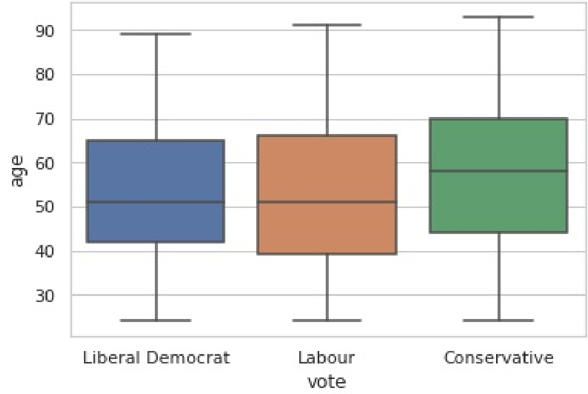
In [19]:

```
sns.set(style="whitegrid")
sns.violinplot(x="vote", y="age", data=beps);
```



In [20]:

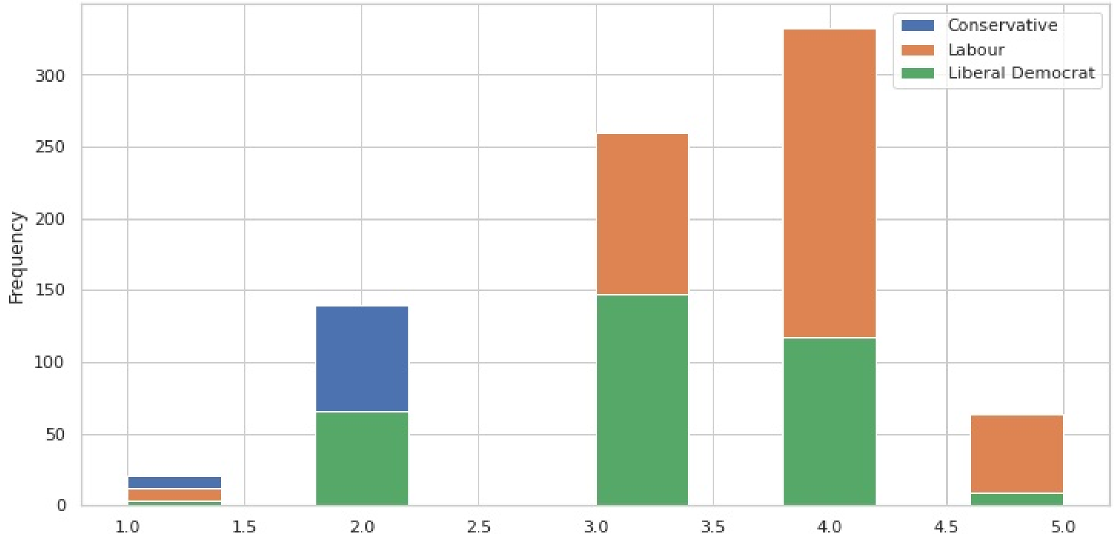
```
sns.boxplot(x="vote", y="age", data=beps);
```



We can tell from the above two graphs that the Conservate party voter's typical age is higher than that of the two other parties

In [21]:

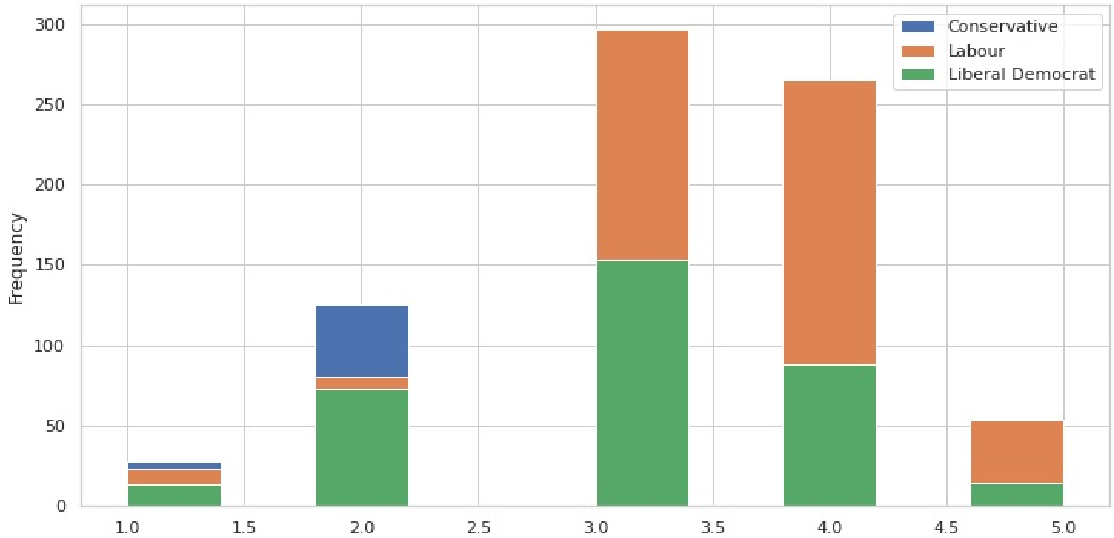
```
beps.groupby('vote')['nat_cond'].plot.hist(legend=True, figsize=FIG_SIZE);
```



It seems like the Labor's party voters were happier with the national economic conditions than the others, followed by the Liberal Democrat's

In [22]:

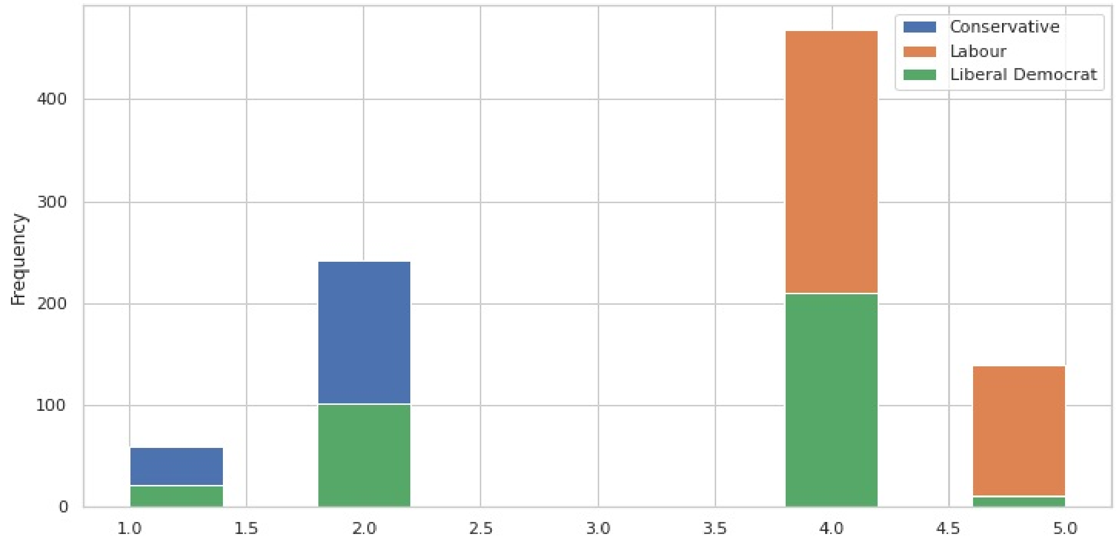
```
beps.groupby('vote')['hhold_cond'].plot.hist(legend=True, figsize=FIG_SIZE);
```



The public attitude towards household economic conditions reflects that towards national economic conditions

In [23]:

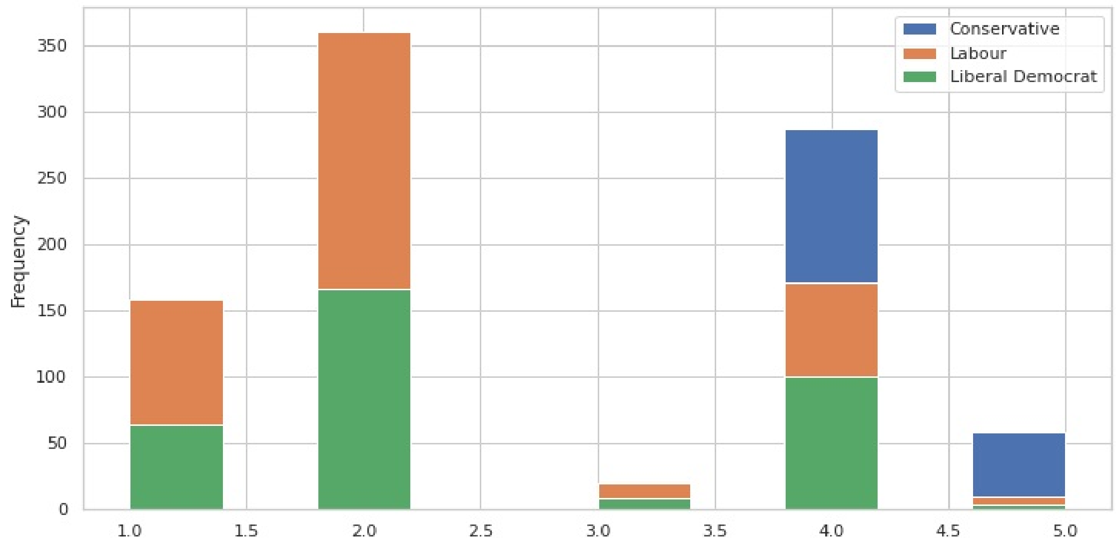
```
beps.groupby('vote')['labor_lead_assmnt'].plot.hist(legend=True, figsize=FIG_SIZE);
```



It seems like the Labor's leader (i.e. Tony Blair) was just fine, but the voters might wanted more, because even among the Labor's voters there were way more 4s than 5s. Also, it seems like he was more popular among the Libral Democrats than the Conservatives.

In [24]:

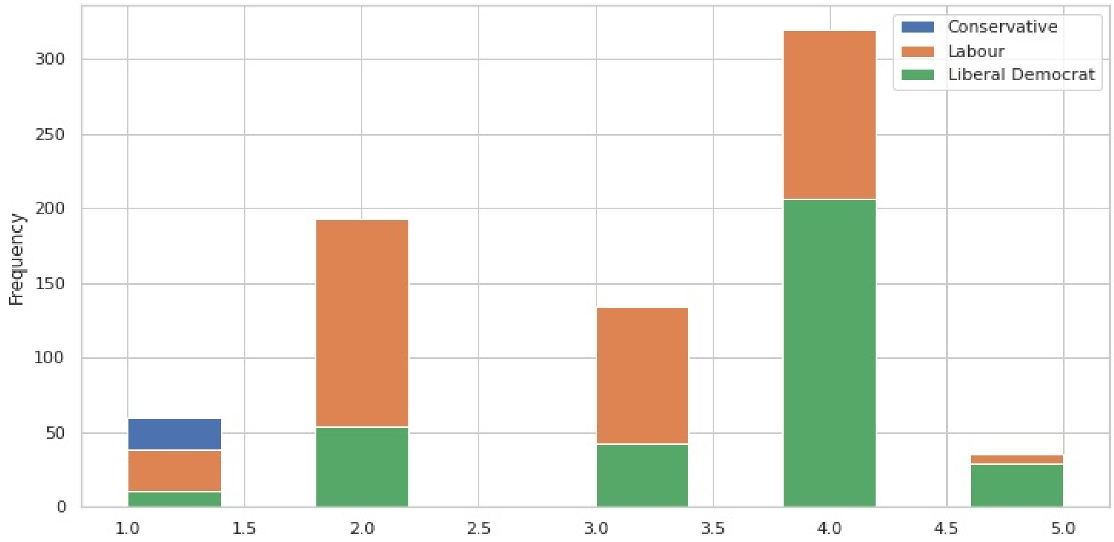
```
beps.groupby('vote')['cons_lead_assmnt'].plot.hist(legend=True, figsize=FIG_SIZE);
```



It doesn't seem like the conservative's leader (i.e. John Major) was more popular among Labour's voters than the Labour's leader was among the Conservatives! But the Liberal Democrats seemed more into the Labour's leader than the Conservative's leader.

In [25]:

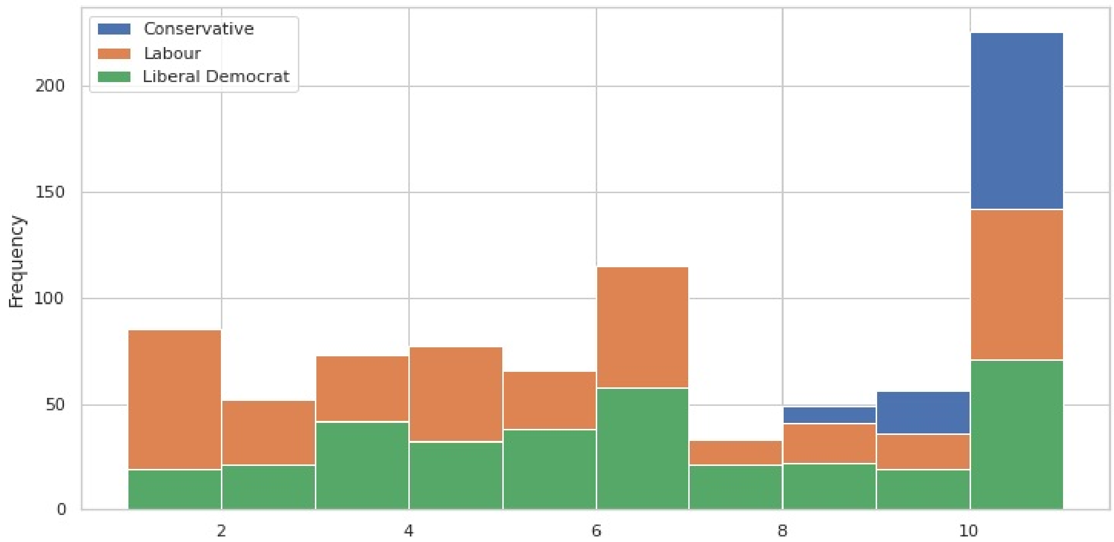
```
beps.groupby('vote')['democ_lead_assmnt'].plot.hist(legend=True, figsize=FIG_SIZE);
```



The Liberal Democrat's leader (i.e. Paddy Ashdown) seemed just fine, but not so popular even among Liberal Democrats or the Labour's voters. But it's obvious that the Conservatives didn't like him at all.

In [26]:

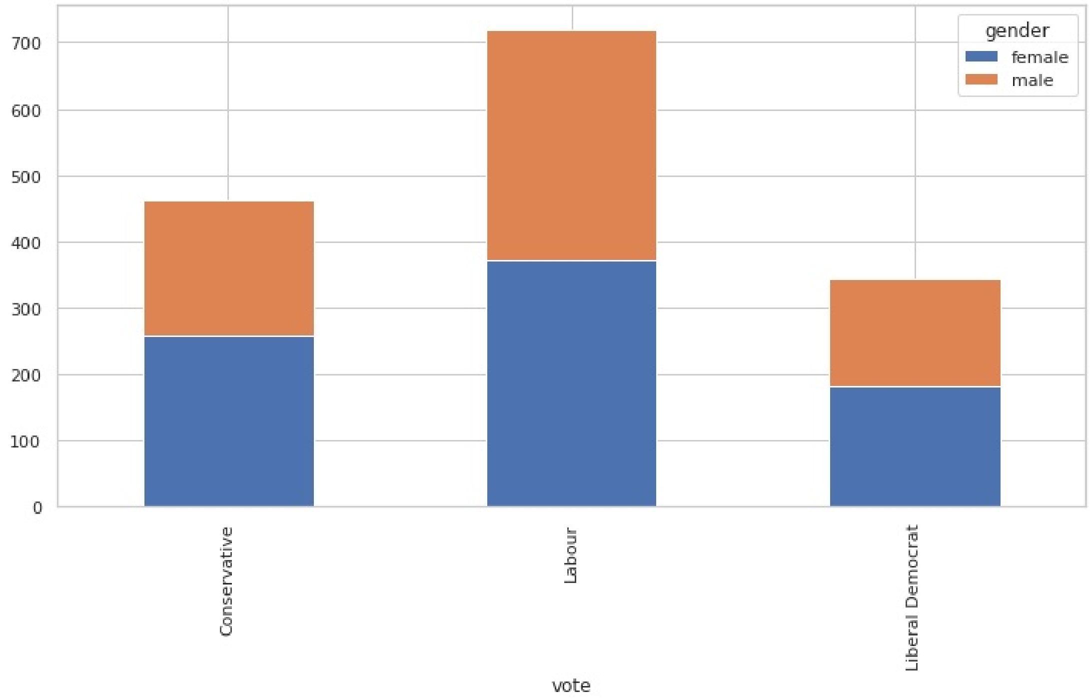
```
beps.groupby('vote')['euro_intg_attud'].plot.hist(legend=True, figsize=FIG_SIZE);
```



The most prominent attitude was the Conservatives attitude! They seemed very Eurosceptic!

In [27]:

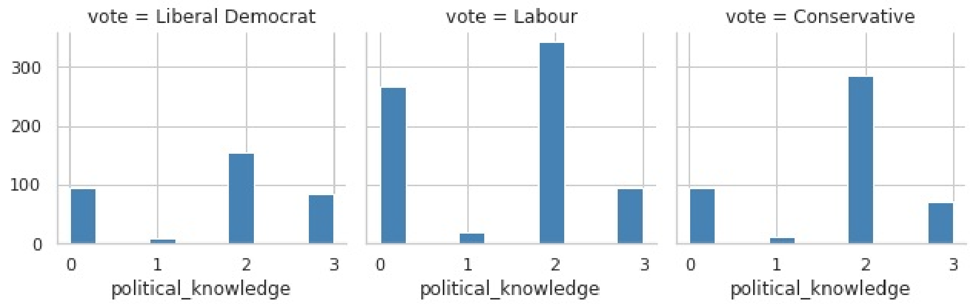
```
beps.groupby(['vote', 'gender'])['vote'].count().unstack('gender').plot.bar(stacked=True, figsize=FIG_SIZE);
```



The number of female voters in almost all the parties was almost half the number of male voters!

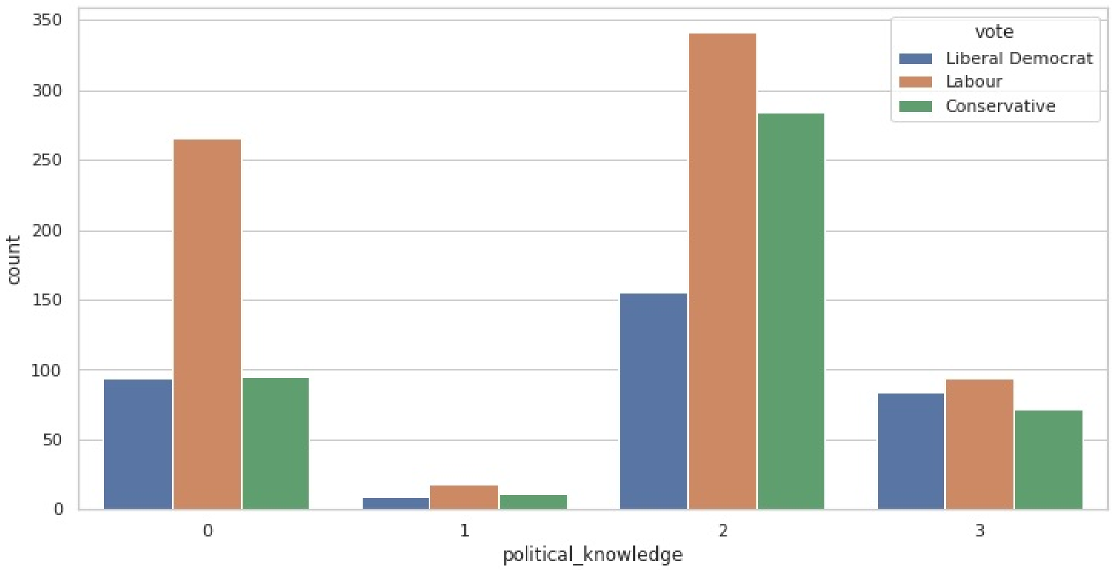
In [28]:

```
g = sns.FacetGrid(beps, col="vote", margin_titles=True)
g.map(plt.hist, "political_knowledge", color="steelblue");
```



In [29]:

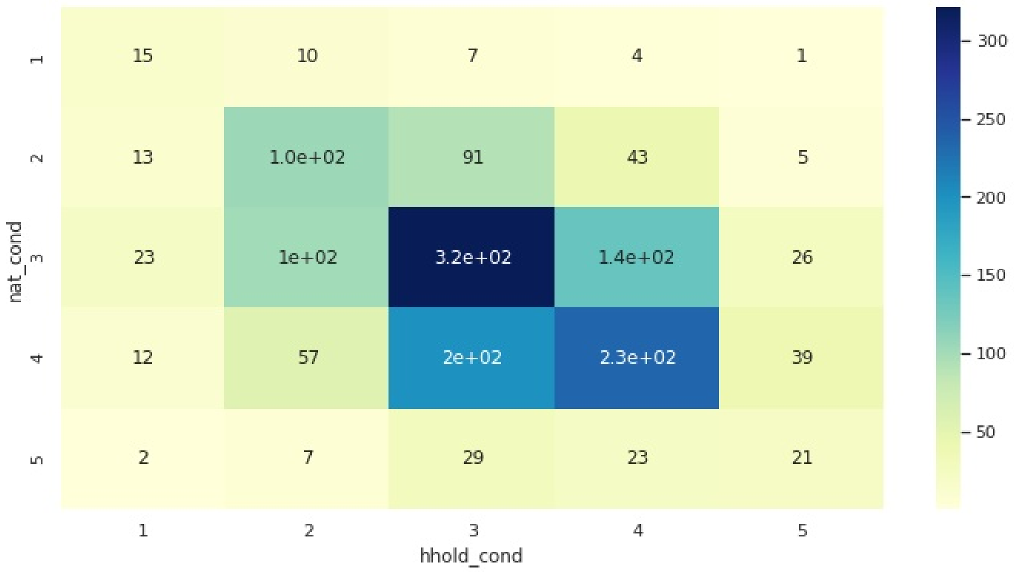
```
plt.figure(figsize=FIG_SIZE)
sns.countplot(x='political_knowledge', hue='vote', data=beps);
```



We can vaguely say that the Conservatives tend to report higher knowledge of parties' positions on European integration than the other parties' voters tend to do!

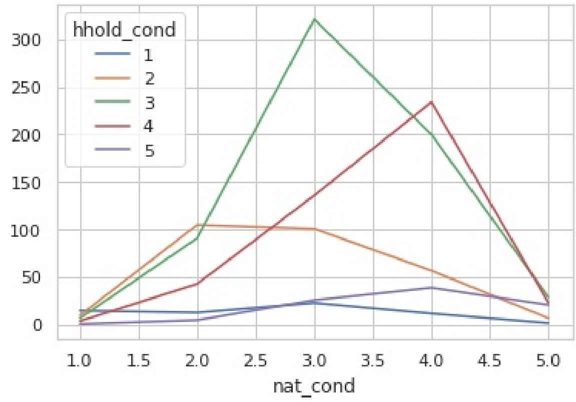
In [30]:

```
nat_hhold = beps.groupby(["nat_cond", "hhold_cond"])["nat_cond"].count()
plt.figure(figsize=FIG_SIZE)
sns.heatmap(nat_hhold.unstack("hhold_cond"), annot=True, cmap="YlGnBu");
```



In [31]:

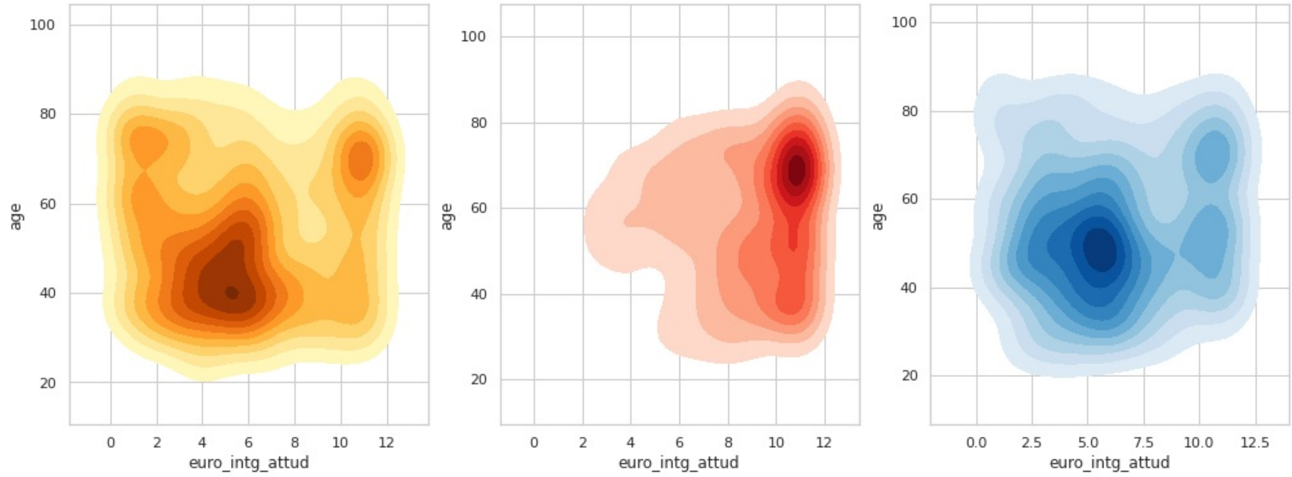
```
nat_hhold.unstack().plot();
```



The relationhsip between voters' assessment of current national vs. household economic conditions is not linear! It's more like a bell shape skewed to the right. Voters were half-half satisfied about both!

In [32]:

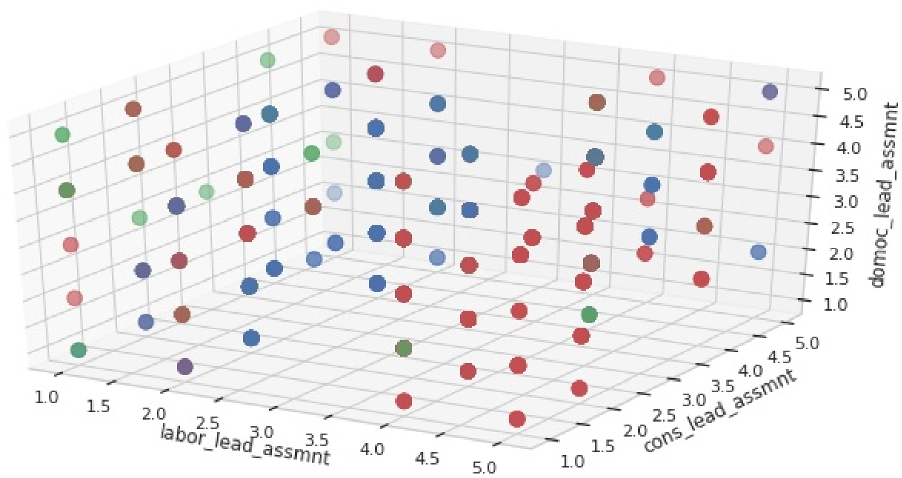
```
plt.figure(figsize=(17, 6))
vote_lab = beps.loc[beps.vote == 'Labour']
vote_cons = beps.loc[beps.vote == 'Conservative']
vote_democ = beps.loc[beps.vote == 'Liberal Democrat']
plt.subplot(131)
sns.kdeplot(vote_lab['euro_intg_attud'], vote_lab['age'], cmap="YlOrBr", shade=True, shade_lowest=False)
plt.subplot(132)
sns.kdeplot(vote_cons['euro_intg_attud'], vote_cons['age'], cmap="Reds", shade=True, shade_lowest=False)
plt.subplot(133)
sns.kdeplot(vote_democ['euro_intg_attud'], vote_democ['age'], cmap="Blues", shade=True, shade_lowest=False);
```



The trend of older and more Eurosceptic Conservatives is obvious once more!

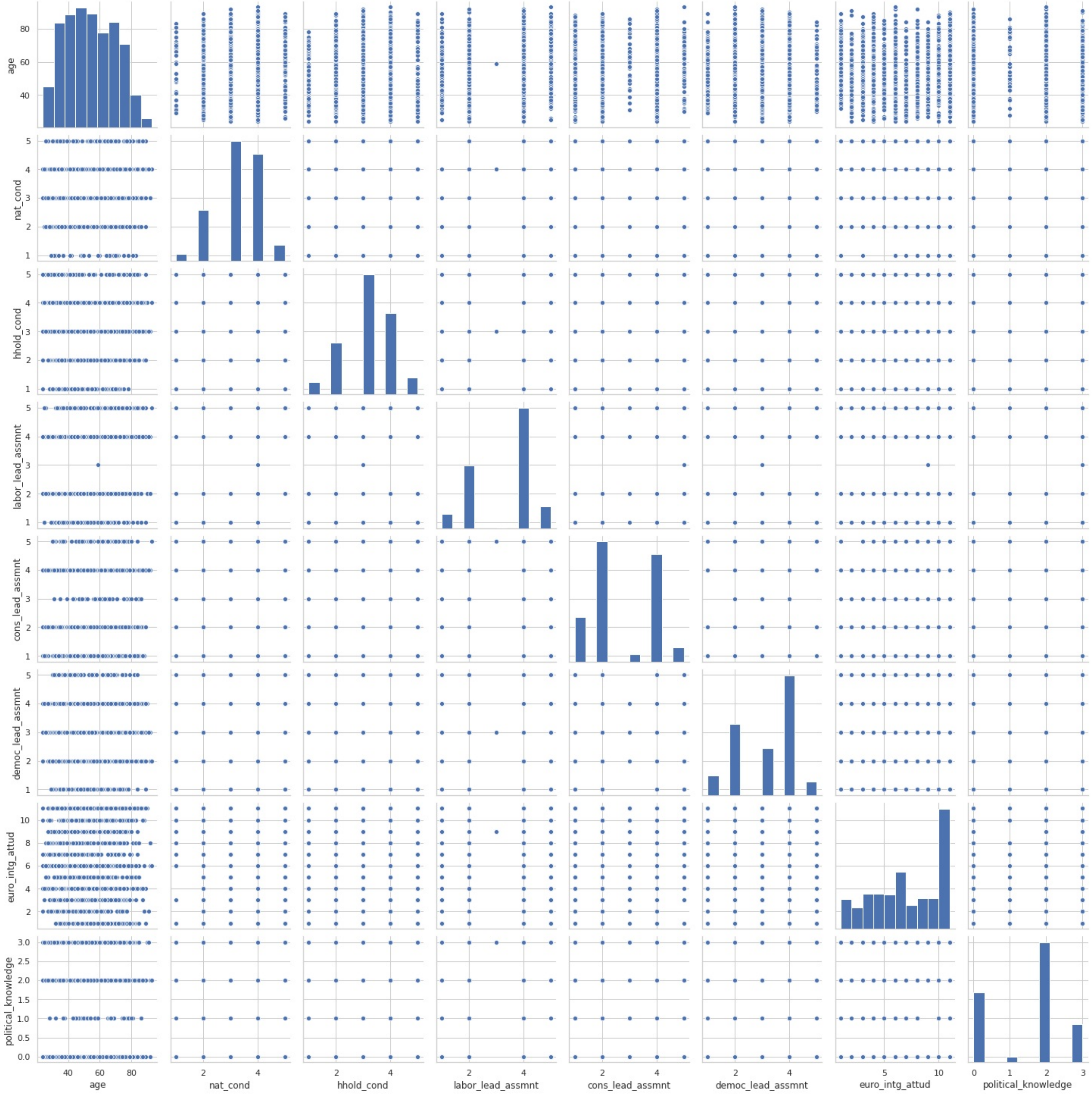
In [33]:

```
fig = plt.figure(figsize=FIG_SIZE)
ax = fig.add_subplot(111, projection='3d')
beps_c = beps['vote'].map({'Labour':'r', 'Conservative':'b', 'Liberal Democrat':'g'})
ax.scatter(beps['labor_lead_assmnt'], beps['cons_lead_assmnt'], beps['democ_lead_assmnt'], s = 90, c=beps_c)
ax.set_xlabel('labor_lead_assmnt')
ax.set_ylabel('cons_lead_assmnt')
ax.set_zlabel('domoc_lead_assmnt')
plt.show()
```



In [34]:

```
sns.pairplot(beps[['age', 'nat_cond', 'hhold_cond', 'labor_lead_assmnt', 'cons_lead_assmnt', 'democ_lead_assmnt',
'euro_intg_attud', 'political_knowledge']]);
```



There is no **linear** correlation between any pair of variables! Even between variables like age and attitudes toward European integration for example, or age and political knowledge!

Model Selection

In [35]:

```
#changing gender column to 1 =female 0=male
beps.gender.replace(['female','male'],[1,0],inplace=True)
beps.vote.replace(['Labour','Conservative','Liberal Democrat'],[0,1,2], inplace=True)
```

In [36]:

```
#Working on models
# Separating target columns
X = beps.drop('vote', axis='columns')
y = beps.vote
```

In [37]:

```
#Will train with (X_train,y_train)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 7)
```

In [38]:

```
#f1_scorer for the gridSearch
f1_scorer = make_scorer(f1_score, average='micro')
```

In [39]:

```
len(X_train)
```

Out[39]:

1220

In [40]:

```
len(X_test)
```

Out[40]:

305

Helper Functions

In [41]:

```
#Method for ploting learning curve and Validation Curve
def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 1, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes[0].fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes[0].plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes[0].plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes[0].legend(loc="best")

    #plot Validation curve

    param_range = np.logspace(-6, -1, 5)
    trainVC_scores, testVC_scores = validation_curve(
        estimator, X, y, param_name="gamma", param_range=param_range,
        scoring="accuracy", n_jobs=1)
    trainVC_scores_mean = np.mean(trainVC_scores, axis=1)
    trainVC_scores_std = np.std(trainVC_scores, axis=1)
    testVC_scores_mean = np.mean(testVC_scores, axis=1)
    testVC_scores_std = np.std(testVC_scores, axis=1)

    plt.title("Validation Curve with SVM")
    plt.xlabel(r"$\gamma$")
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.semilogx(param_range, trainVC_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, trainVC_scores_mean - trainVC_scores_std,
                     trainVC_scores_mean + trainVC_scores_std, alpha=0.2,
                     color="darkorange", lw=lw)
    plt.semilogx(param_range, testVC_scores_mean, label="Cross-validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, testVC_scores_mean - testVC_scores_std,
                     testVC_scores_mean + testVC_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")

    return plt
```

In [42]:

```
#Compare the testing data and prediction results by heatmap
def plot_confusion_matrix(y_test, y_preds):
    cm = confusion_matrix(y_test, y_preds)
    plt.figure(figsize=(10,7))
    sns.heatmap(cm,annot=True,fmt='d',cmap="Blues",linewidths=.08)
    plt.xlabel("Predicted")
    plt.ylabel("Truth")
    plt.show()
    print("Classification Report")
    print(classification_report(y_test, y_preds))
```

In [43]:

```python
#Method for ploting learning curve and Validation Curve
def learningPlot_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 1, figsize=(20, 5))

    axes.set_title(title)
    if ylim is not None:
        axes.set_ylim(*ylim)
    axes.set_xlabel("Training examples")
    axes.set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes.grid()
    axes.fill_between(train_sizes, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.1,
                         color="r")
    axes.fill_between(train_sizes, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.1,
                         color="g")
    axes.plot(train_sizes, train_scores_mean, 'o-', color="r",
                 label="Training score")
    axes.plot(train_sizes, test_scores_mean, 'o-', color="g",
                 label="Cross-validation score")
    axes.legend(loc="best")

    return plt
```

In [44]:

```
#Validation Curve
def validationPlot_curve(estimator, title, X, y,param,paramRange, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 1, figsize=(20, 5))

    if ylim is not None:
        axes[0].set_ylim(*ylim)

    #plot Validation curve

    param_range = paramRange
    #np.logspace(-6, -1, 5)
    trainVC_scores, testVC_scores = validation_curve(
        estimator, X, y, param_name=param, param_range=param_range,
        scoring="accuracy", n_jobs=1)
    trainVC_scores_mean = np.mean(trainVC_scores, axis=1)
    trainVC_scores_std = np.std(trainVC_scores, axis=1)
    testVC_scores_mean = np.mean(testVC_scores, axis=1)
    testVC_scores_std = np.std(testVC_scores, axis=1)

    plt.title("Validation Curve")
    plt.xlabel(r"$\gamma$")
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.semilogx(param_range, trainVC_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, trainVC_scores_mean - trainVC_scores_std,
                     trainVC_scores_mean + trainVC_scores_std, alpha=0.2,
                     color="darkorange", lw=lw)
    plt.semilogx(param_range, testVC_scores_mean, label="Cross-validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, testVC_scores_mean - testVC_scores_std,
                     testVC_scores_mean + testVC_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")

    return plt
```

In [45]:

```
def plot_curves(model, param_name, param_range):
    cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=7)
    validationPlot_curve(model, title, X, y,param_name,param_range,cv=cv, n_jobs=4)
    plt.show()
    learningPlot_curve(model, title, X, y,cv=cv, n_jobs=4)
    plt.show()
```

Support Vector Machine Model (SVM) Analysis

In [46]:

```
#Creating model
model = SVC()
```

In [47]:

```
model.fit(X_train, y_train)
```

Out[47]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [48]:

```
model.score(X_test, y_test)
```

Out[48]:

```
0.639344262295082
```

In [49]:

```
#HyperParameter Tuning
param_grid = {'C': [1,50,100],
              'gamma': [0.1,0.01,0.001,'scale'],
              'kernel': ['rbf','linear']}
```

In [50]:

```
grid = GridSearchCV(
    estimator=SVC(),
    param_grid=param_grid,
    cv=5,
    return_train_score=False,
    scoring=f1_scorer,
    n_jobs=-1,
    verbose=2)
svm_grid=grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:    6.6s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed: 11.3min finished
```

In [51]:

```
svc_best_params = grid.best_params_
```

In [52]:

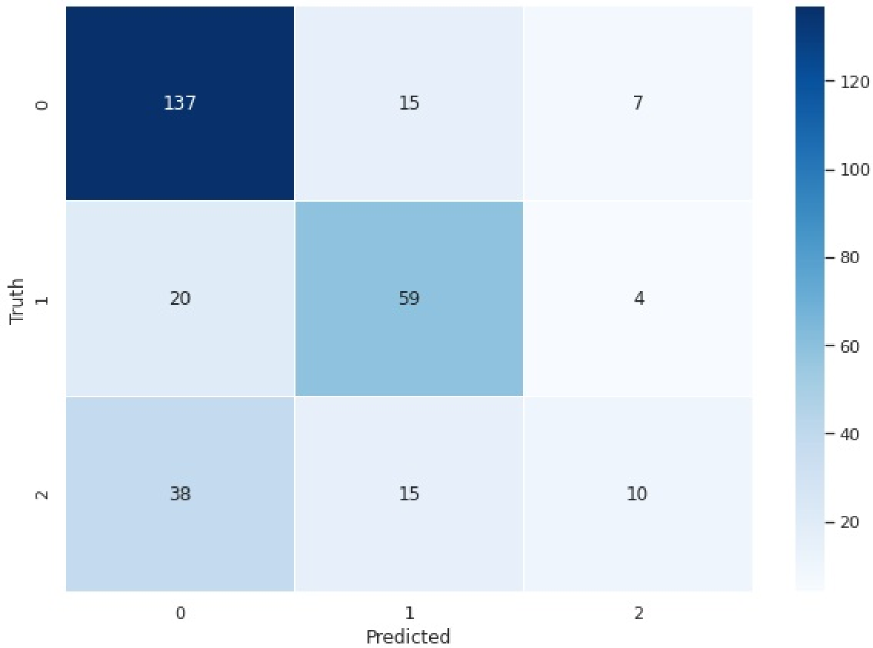
```
grid.best_score_
```

Out[52]:

```
0.6680327868852459
```

In [53]:

```
grid_predictions = grid.predict(X_test)
plot_confusion_matrix(y_test, grid_predictions)
```




```
Classification Report
              precision    recall  f1-score   support

           0       0.70      0.86      0.77       159
           1       0.66      0.71      0.69        83
           2       0.48      0.16      0.24        63

    accuracy                           0.68       305
   macro avg       0.61      0.58      0.57       305
weighted avg       0.65      0.68      0.64       305
```

In [54]:

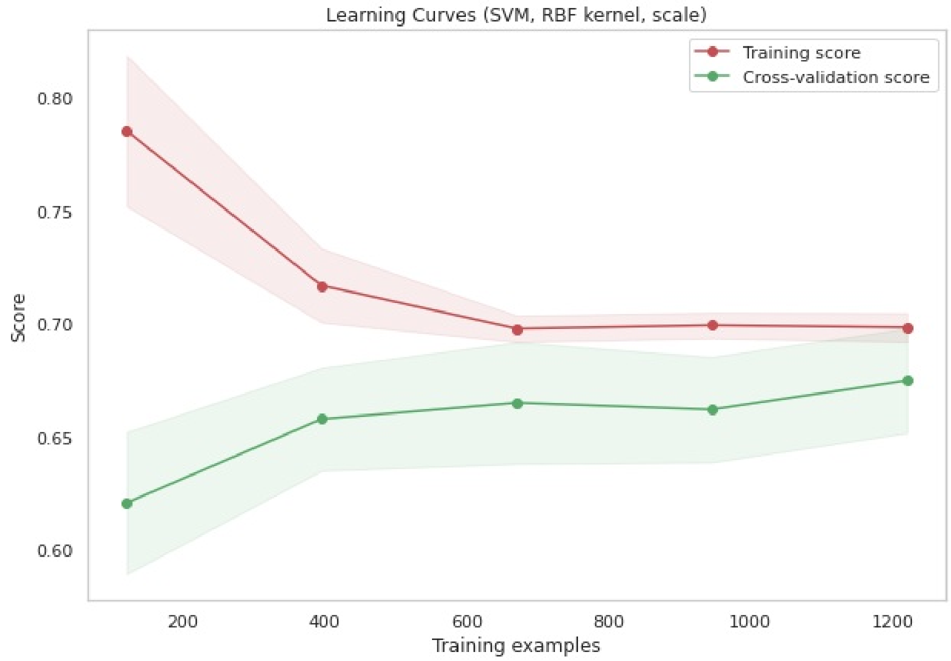
```
# Learn to predict each class against the other
#ROC curve using the best params found in gridSearch
classifier = OneVsRestClassifier(SVC(C=100,kernel='rbf', probability=True,
                                 gamma='scale'))
```

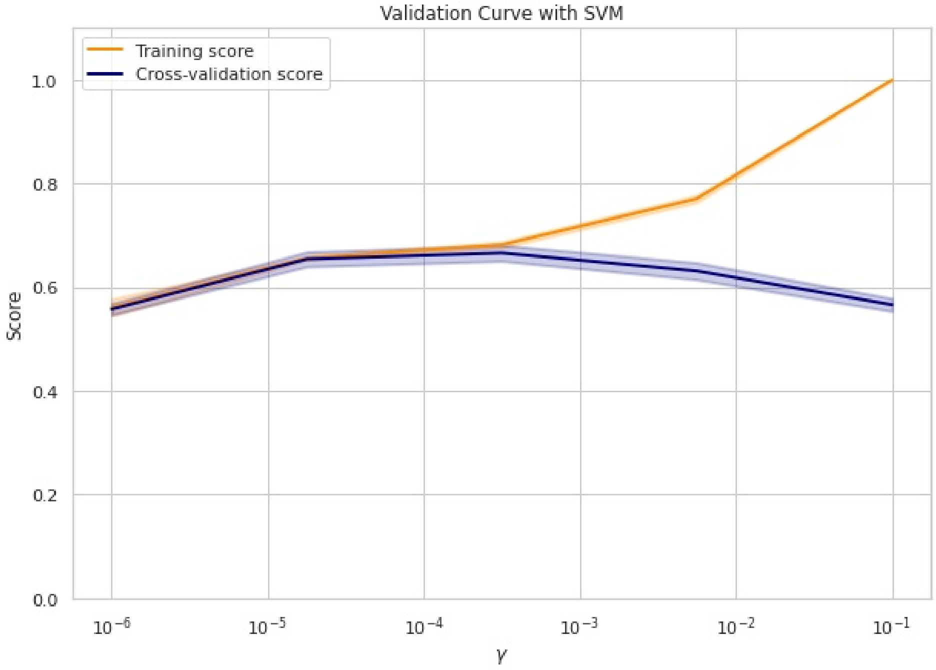
In [55]:

```
#Generating new yscore with the binarize data
svm_y_score = classifier.fit(X_train, y_train).decision_function(X_test)
```

In [56]:

```
fig, axes = plt.subplots(2, 1, figsize=(10, 15))
title = r"Learning Curves (SVM, RBF kernel, scale)"
# SVC is more expensive so we do a lower number of CV iterations:
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
estimator =SVC(**svc_best_params)
plot_learning_curve(estimator, title, X, y, axes=axes,
                    cv=cv, n_jobs=4)

plt.show()
```





From the learning curve we can see that as the dataset is increacing the training accuracy is decreasing exponentially ,while the cross-validation score is increasing until they both reaches a point where they start converging to a lower value. We can conclude that our dataset is really complexe and won't benefit from adding more dataset. From the validation curve

Random Forest Model Analysis

In [57]:

```
param_grid = {'n_estimators': [10, 100, 150, 200, 250, 300],
              'max_depth': [3, 4, 5, 6, 7, 8, 9],
              'max_features': [2, 4, 5, 6, 7, 9]
              }
```

In [58]:

```
#Using GridSearchCV for HPTuning
grid = GridSearchCV(
    estimator=RandomForestClassifier(),
    param_grid=param_grid,
    cv=5,
    return_train_score=False,
    scoring=make_scorer(f1_score, average='micro'),
    n_jobs=-1,
    verbose=2)
```

In [59]:

```
rf_grid = grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 252 candidates, totalling 1260 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  53 tasks      | elapsed:   10.9s
[Parallel(n_jobs=-1)]: Done 174 tasks      | elapsed:   40.2s
[Parallel(n_jobs=-1)]: Done 377 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 660 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 1025 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 1260 out of 1260 | elapsed:  5.8min finished
```

In [60]:

```
rf_y_score = rf_grid.predict_proba(X_test)
```

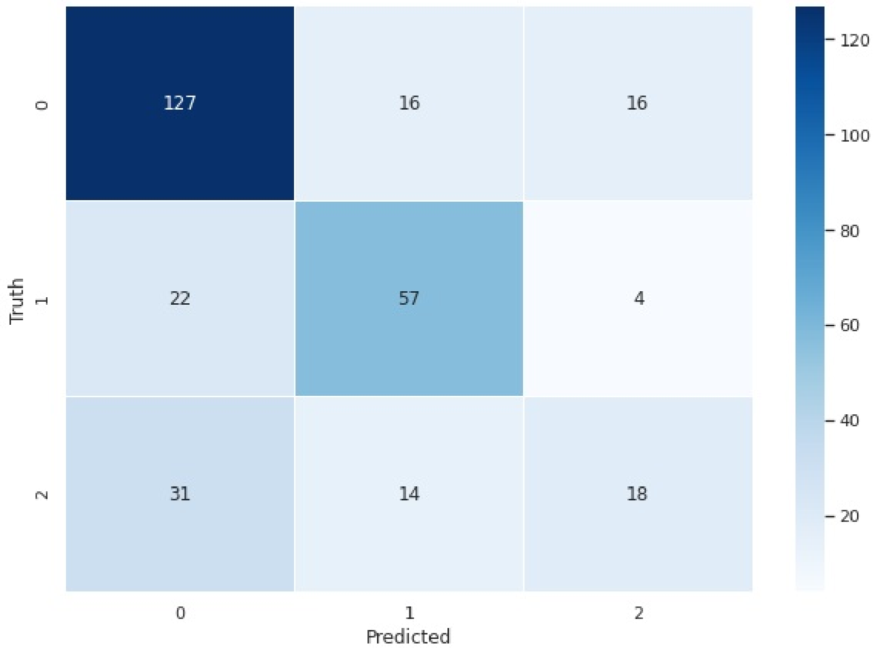
In [61]:

```
#Find Best Estimator
rf_best_params = rf_grid.best_params_
rf_best_est = rf_grid.best_estimator_
print("Best estimator:")
print(rf_best_est)
```

```
Best estimator:
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=9, max_features=2,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [62]:

```
rf_grid_predictions = rf_best_est.predict(X_test)
plot_confusion_matrix(y_test, rf_grid_predictions)
```




```
Classification Report
              precision    recall  f1-score   support

           0       0.71      0.80      0.75       159
           1       0.66      0.69      0.67        83
           2       0.47      0.29      0.36        63

    accuracy                           0.66       305
   macro avg       0.61      0.59      0.59       305
weighted avg       0.64      0.66      0.65       305
```

In [63]:

```
#Importance  feature List: labor_lead_assmnt is the most importance
feature_list = list(X)
feature_imp= pd.Series(rf_best_est.feature_importances_,index = feature_list).sort_values(ascending = False)
print(feature_imp)
```

```
cons_lead_assmnt       0.170774
euro_intg_attud        0.159727
labor_lead_assmnt      0.157849
age                    0.146965
nat_cond               0.087679
democ_lead_assmnt      0.087164
political_knowledge    0.085692
hhold_cond             0.073668
gender                 0.030483
dtype: float64
```

In [64]:

```
n_estimators = [10, 30, 50, 70, 100, 150, 200, 250, 300, 350]
max_depths = [2, 3, 4, 5, 6, 7, 8, 9]
acc_scores = []
val_scores = []
for mx_dep in max_depths:
    model_randomForest = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=mx_dep, max_features=2,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
    model_randomForest.fit(X_train,y_train)
    acc_scores.append(100 * (1 - model_randomForest.score(X_train, y_train)))
    preds = model_randomForest.predict(X_test)
    val_scores.append(100 * (1 - accuracy_score(y_test, preds)))
```

In [65]:

```
plt.style.use('seaborn')
plt.plot(max_depths, acc_scores, label = 'Training error')
plt.plot(max_depths, val_scores, label = 'Validation error')
plt.ylabel('Error', fontsize = 14)
plt.xlabel('Max Depth', fontsize = 14)
plt.title('Learning curves for a Random Forest model', fontsize = 18, y = 1.03)
plt.legend()
plt.ylim(0,100)
```

Out[65]:

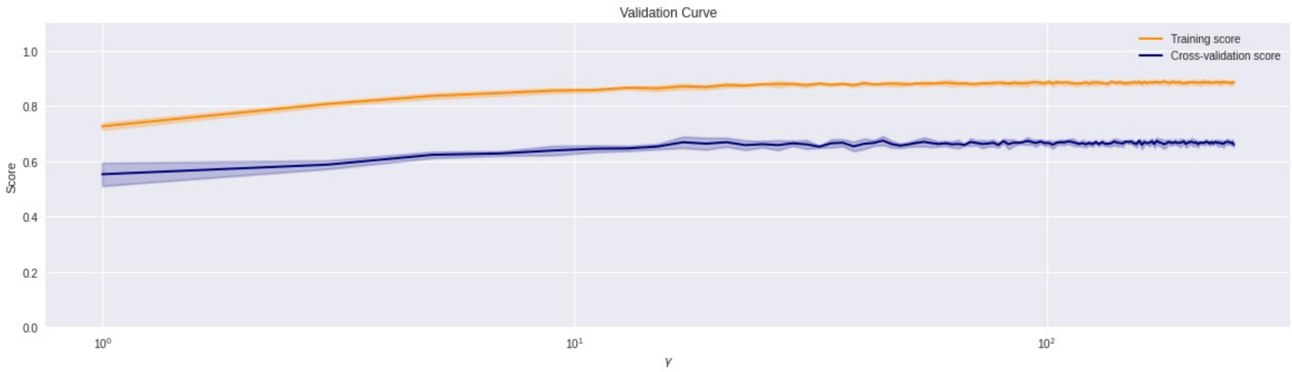
```
(0.0, 100.0)
```

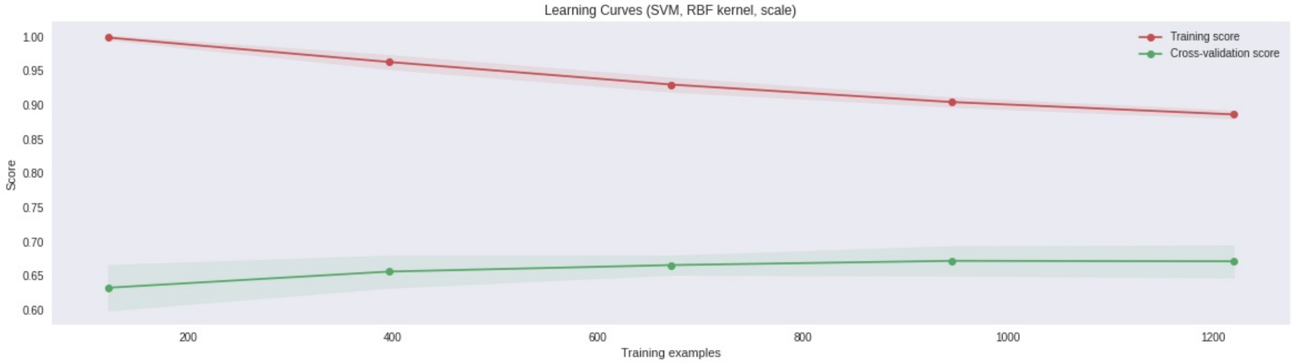
In [66]:

```
best_rf = RandomForestClassifier(**rf_best_params)
```

In [67]:

```
param_range = np.arange(1, 250, 2)
plot_curves(best_rf, "n_estimators", param_range)
```





Multi-layer Perceptron Classifier (MLP) Model Analsysis

In [68]:

```
# Parameter tuning
from sklearn.neural_network import MLPClassifier
parameters = {
    'hidden_layer_sizes': [(100,), (200,), (300, ), (100, 2), (200, 2)],
    'solver': ['adam', 'lbfgs'],
    'alpha': [0.001, 0.0001, 10, 100],
    'activation': ['relu'],
    'learning_rate': ['adaptive']
}

clf = GridSearchCV(estimator=MLPClassifier(max_iter=30000), param_grid=parameters, cv=5, scoring=f1_scorer, n_job
s=-1, verbose=2)
```

In [69]:

```
mlp_model = clf.fit(X_train, y_train)
# mlp_y_score = mlp_model.decision_function(X_test)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 11.5min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 13.0min finished
```

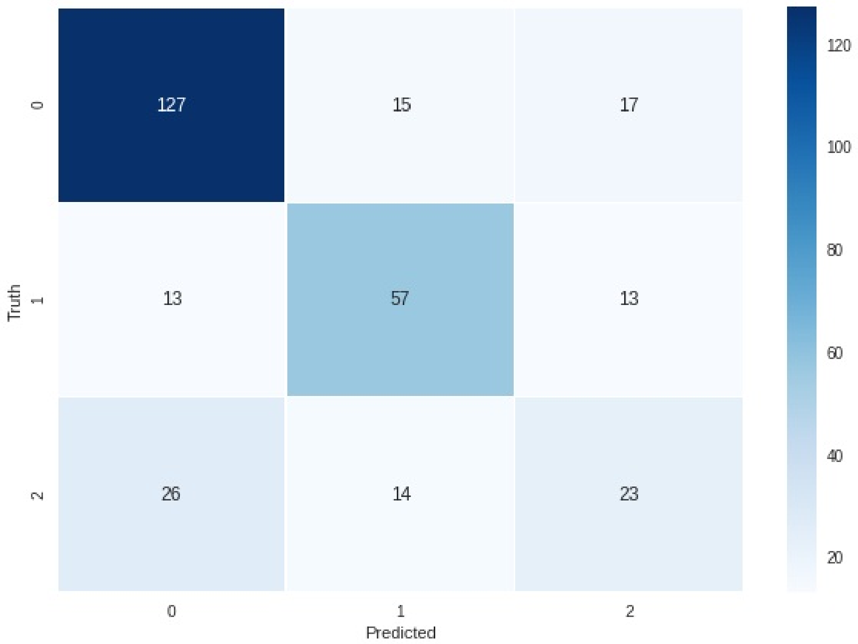
In [70]:

```
print(clf.best_score_)
print(clf.best_params_)
mlp_best_params = clf.best_params_
```

```
0.6786885245901639
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', '
solver': 'adam'}
```

In [71]:

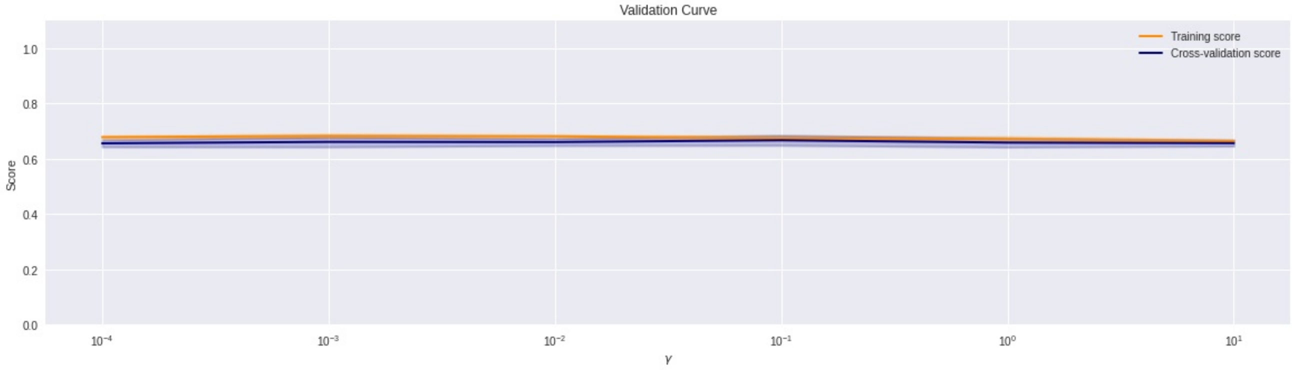
```
# confusion matrix
predictions = mlp_model.predict(X_test)
plot_confusion_matrix(y_test, predictions)
```

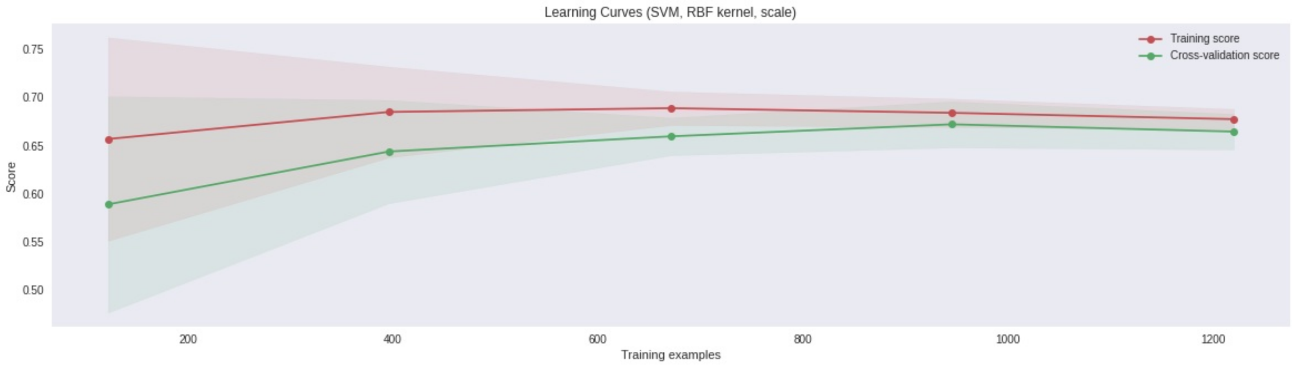



```
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.80      0.78       159
           1       0.66      0.69      0.67        83
           2       0.43      0.37      0.40        63

    accuracy                           0.68       305
   macro avg       0.62      0.62      0.62       305
weighted avg       0.67      0.68      0.67       305
```

In [72]:

```
best_mlp = MLPClassifier(**mlp_best_params)
param_range = np.array([0.0001, 0.001, 0.01, 0.1, 1, 10])
plot_curves(best_mlp, "alpha", param_range)
```





Logistical Regression Model Analysis

In [73]:

```
# basic
clf = LogisticRegressionCV(cv = 5).fit(X_train, y_train)
print(clf)
print("Train Error: ", clf.score(X_train, y_train))
print("Test Error: ", clf.score(X_test, y_test))
```

```
LogisticRegressionCV(Cs=10, class_weight=None, cv=5, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='auto', n_jobs=None,
                     penalty='l2', random_state=None, refit=True, scoring=None,
                     solver='lbfgs', tol=0.0001, verbose=0)
Train Error:  0.6778688524590164
Test Error:  0.6754098360655738
```

In [74]:

```
# first
clf = LogisticRegression()
acc_scorer = make_scorer(accuracy_score)
parameters = {'C': [0.01, 0.03, 0.05, 0.07],
              'solver': ['newton-cg', 'lbfgs']}

# Grid Search (use the default 5-fold cross validation)
grid_obj = GridSearchCV(clf, parameters, acc_scorer, cv = 5)
grid_obj = grid_obj.fit(X_train, y_train)
logit_y_score = grid_obj.decision_function(X_test)

# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_
logit_best_params = grid_obj.best_params_
print(clf)

print("Best Score: ", grid_obj.best_score_)
print("Train Error: ", clf.score(X_train, y_train))
print("Test Error: ", clf.score(X_test, y_test))
```

```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
Best Score:  0.6762295081967213
Train Error:  0.6836065573770492
Test Error:  0.6754098360655738
```

In [75]:

```
# second
clf = LogisticRegressionCV(cv = 5)

acc_scorer = make_scorer(accuracy_score)
parameters = {'Cs': [5, 10, 15],
              'solver': ['newton-cg', 'lbfgs']}

# Grid Search (use the default 5-fold cross validation)
grid_obj = GridSearchCV(clf, parameters, acc_scorer, cv = 10)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_
print(clf)

print("Best Score: ", grid_obj.best_score_)
print("Train Error: ", clf.score(X_train, y_train))
print("Test Error: ", clf.score(X_test, y_test))
```

```
LogisticRegressionCV(Cs=10, class_weight=None, cv=5, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='auto', n_jobs=None,
                     penalty='l2', random_state=None, refit=True, scoring=None,
                     solver='newton-cg', tol=0.0001, verbose=0)
Best Score:  0.6688524590163935
Train Error:  0.6819672131147541
Test Error:  0.6754098360655738
```

In [76]:

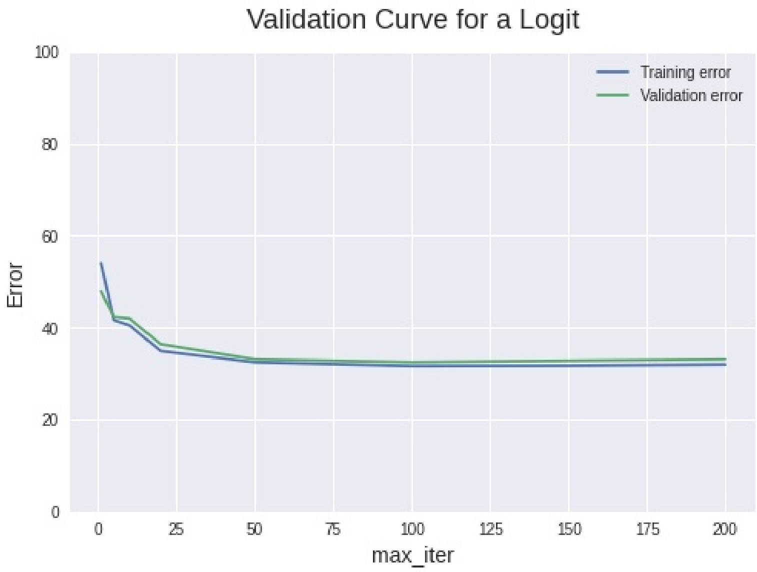
```
# logit
max_iter_values = [1, 5, 10, 20, 50, 100, 150, 200]

acc_scores = []
val_scores = []
for x in max_iter_values:
    model = LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,
                               intercept_scaling=1, l1_ratio=None, max_iter=x,
                               multi_class='auto', n_jobs=None, penalty='l2',
                               random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                               warm_start=False)

    model.fit(X_train, y_train)
    acc_scores.append(100 * (1 - model.score(X_train, y_train)))
    preds = model.predict(X_test)
    val_scores.append(100 * (1 - accuracy_score(y_test, preds)))
```

In [77]:

```
# using?
darw_acc = max_iter_values

plt.style.use('seaborn')
plt.plot(darw_acc, acc_scores, label = 'Training error')
plt.plot(darw_acc, val_scores, label = 'Validation error')
plt.ylabel('Error', fontsize = 14)
plt.xlabel('max_iter', fontsize = 14)
plt.title('Validation Curve for a Logit', fontsize = 18, y = 1.03)
plt.legend()
plt.ylim(0,100)
```

Out[77]:

```
(0.0, 100.0)
```

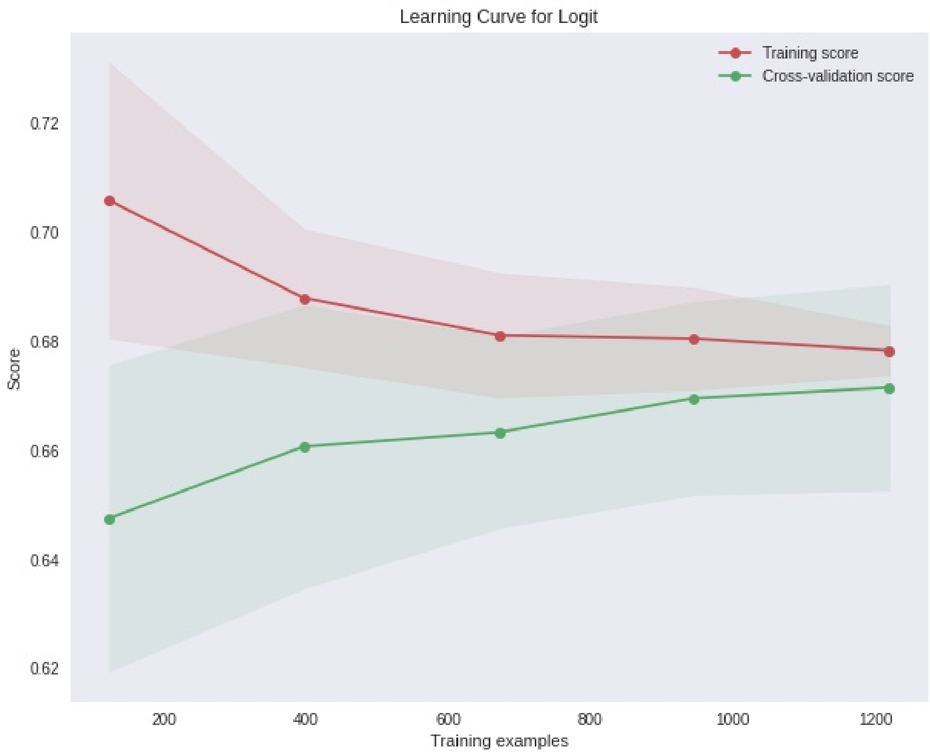


In [78]:

```
# Method for ploting learning curve and Validation Curve
def plot_learning_curve_2(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 1, figsize=(20, 5))

    axes.set_title(title)
    if ylim is not None:
        axes.set_ylim(*ylim)
    axes.set_xlabel("Training examples")
    axes.set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes.grid()
    axes.fill_between(train_sizes, train_scores_mean - train_scores_std,
                      train_scores_mean + train_scores_std, alpha=0.1,
                      color="r")
    axes.fill_between(train_sizes, test_scores_mean - test_scores_std,
                      test_scores_mean + test_scores_std, alpha=0.1,
                      color="g")
    axes.plot(train_sizes, train_scores_mean, 'o-', color="r",
              label="Training score")
    axes.plot(train_sizes, test_scores_mean, 'o-', color="g",
              label="Cross-validation score")
    axes.legend(loc="best")

    return plt
```

In [79]:

```
# Logit
fig, axes = plt.subplots(1, 1, figsize=(10,8))
title = "Learning Curve for Logit"

cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=7)

estimator = LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

plot_learning_curve_2(estimator, title, X, y, cv = cv, axes=axes)

plt.show()
```



K-Nearest Neigbor Model Analysis

In [80]:

```
# Parameter tuning
from sklearn.neighbors import KNeighborsClassifier
parameters = {
    'n_neighbors': [1, 2, 5, 10, 15, 20, 25, 30],
    'weights': ['uniform', 'distance'],
}

clf = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parameters, cv=5, scoring=f1_scorer, n_jobs=-1, v
erbose=2)
```

In [81]:

```
knn_model = clf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    0.9s finished
```

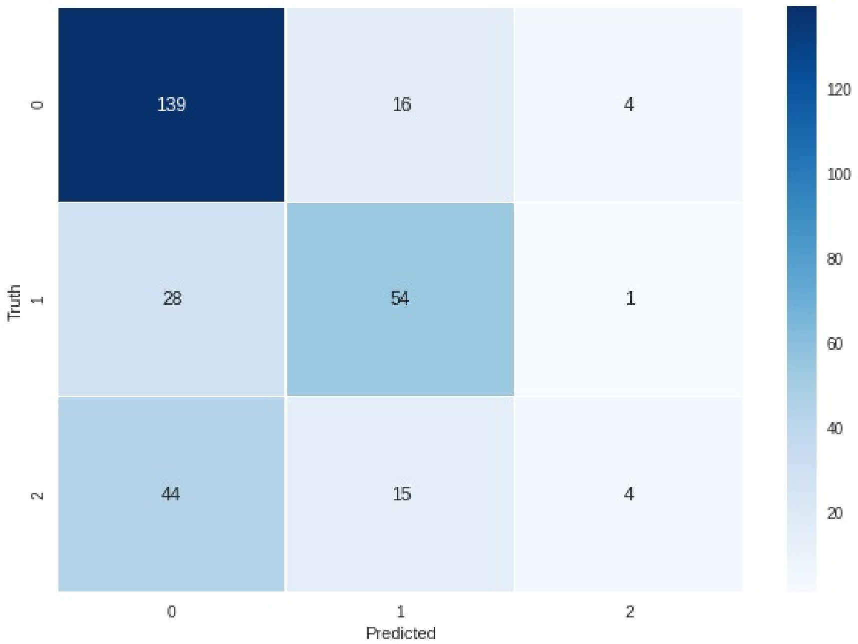
In [82]:

```
print(clf.best_score_)
print(clf.best_params_)
knn_best_params = clf.best_params_
```

```
0.6360655737704918
{'n_neighbors': 25, 'weights': 'uniform'}
```

In [83]:

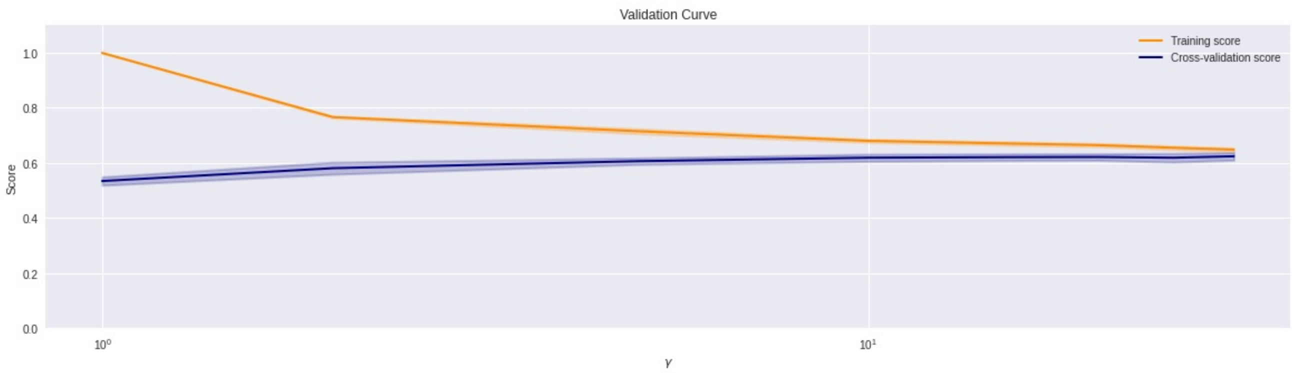
```
# confusion matrix
predictions = knn_model.predict(X_test)
plot_confusion_matrix(y_test, predictions)
```

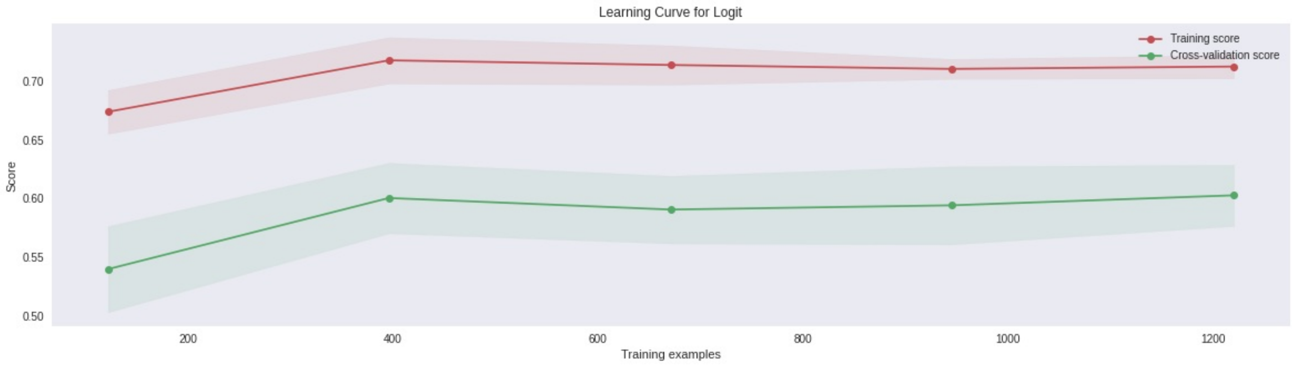



```
Classification Report
              precision    recall  f1-score   support

           0       0.66      0.87      0.75       159
           1       0.64      0.65      0.64        83
           2       0.44      0.06      0.11        63

    accuracy                           0.65       305
   macro avg       0.58      0.53      0.50       305
weighted avg       0.61      0.65      0.59       305
```

In [84]:

```
best_knn = KNeighborsClassifier()
param_range = np.array([1, 2, 5, 10, 15, 20, 25, 30])
plot_curves(best_knn, "n_neighbors", param_range)
```





Naive Bayes Model Analysis

In [86]:

```
# third
clf = GaussianNB()
acc_scorer = make_scorer(accuracy_score)
parameters = {'var_smoothing': [1e-10, 1e-09, 1e-08]}

# Grid Search (use the default 5-fold cross validation)
grid_obj = GridSearchCV(clf, parameters, acc_scorer, cv = 5)
grid_obj = grid_obj.fit(X_train, y_train)
# naive_y_score = grid_obj.decision_function(X_test)

# Set the clf to the best combination of parameters
clf = grid_obj.best_estimator_
naive_best_params = grid_obj.best_params_
print(clf)

print("Best Score: ", grid_obj.best_score_)
print("Train Error: ", clf.score(X_train, y_train))
print("Test Error: ", clf.score(X_test, y_test))
```

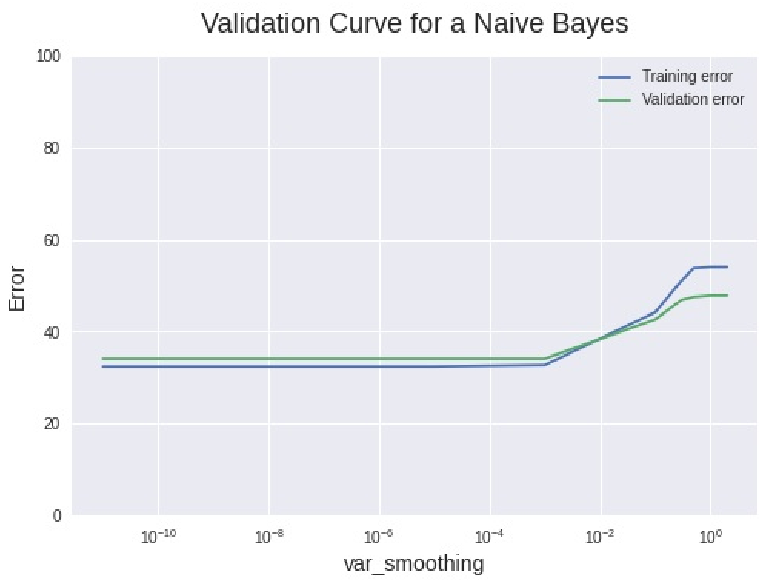
```
GaussianNB(priors=None, var_smoothing=1e-10)
Best Score:  0.6713114754098362
Train Error:  0.6762295081967213
Test Error:  0.659016393442623
```

In [87]:

```
# naive bayes
var_smoothing_values = [2, 1, 0.5, 0.3, 1e-1, 1e-3, 1e-5, 1e-7, 1e-9, 1e-10, 1e-11]

acc_scores = []
val_scores = []
for x in var_smoothing_values:
    model = GaussianNB(priors=None, var_smoothing=x)

    model.fit(X_train, y_train)
    acc_scores.append(100 * (1 - model.score(X_train, y_train)))
    preds = model.predict(X_test)
    val_scores.append(100 * (1 - accuracy_score(y_test, preds)))
```

In [88]:

```
# using?
darw_acc = var_smoothing_values

plt.style.use('seaborn')
plt.plot(darw_acc, acc_scores, label = 'Training error')
plt.plot(darw_acc, val_scores, label = 'Validation error')
plt.ylabel('Error', fontsize = 14)
plt.xlabel('var_smoothing', fontsize = 14)
plt.title('Validation Curve for a Naive Bayes', fontsize = 18, y = 1.03)
plt.legend()
plt.ylim(0, 100)
plt.xscale("log")
```

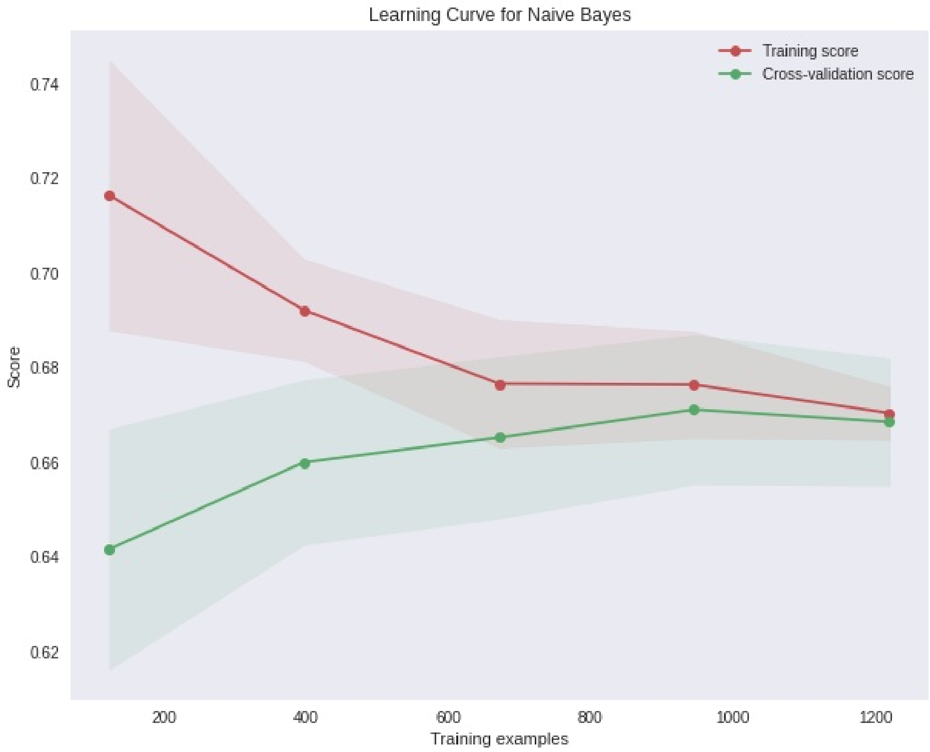


In [89]:

```
# Method for ploting learning curve and Validation Curve
def plot_learning_curve(estimator, title, X, y, axes=None, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    if axes is None:
        _, axes = plt.subplots(1, 1, figsize=(20, 5))

    axes.set_title(title)
    if ylim is not None:
        axes.set_ylim(*ylim)
    axes.set_xlabel("Training examples")
    axes.set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = \
        learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
                       train_sizes=train_sizes,
                       return_times=True)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)

    # Plot learning curve
    axes.grid()
    axes.fill_between(train_sizes, train_scores_mean - train_scores_std,
                      train_scores_mean + train_scores_std, alpha=0.1,
                      color="r")
    axes.fill_between(train_sizes, test_scores_mean - test_scores_std,
                      test_scores_mean + test_scores_std, alpha=0.1,
                      color="g")
    axes.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    axes.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    axes.legend(loc="best")

    return plt
```

In [90]:

```
# Naive Bayes
fig, axes = plt.subplots(1, 1, figsize=(10,8))
title = "Learning Curve for Naive Bayes"

cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=7)

estimator = GaussianNB(priors=None, var_smoothing=1e-10)

plot_learning_curve(estimator, title, X, y, cv = cv, axes=axes)

plt.show()
```



One-vs-Rest

In [91]:

```
nb_ovr = GaussianNB()
```

In [92]:

```
ovr_clf = OneVsRestClassifier(nb_ovr)
ovr_clf.fit(X_train, y_train);
```

In [93]:

```
print("OVR Training Accuracy: %.4f" % ovr_clf.score(X_train, y_train))
print("OVR Validation Accuracy: %.4f" % ovr_clf.score(X_test, y_test))
```

```
OVR Training Accuracy: 0.6721
OVR Validation Accuracy: 0.6623
```

One-vs-One

In [94]:

```
nb_ovo = GaussianNB()
ovo_clf = OneVsOneClassifier(nb_ovo)
ovo_clf.fit(X_train, y_train);
```

In [95]:

```
print("OVO Training Accuracy: %.4f" % ovo_clf.score(X_train, y_train))
print("OVO Validation Accuracy: %.4f" % ovo_clf.score(X_test, y_test))
```

```
OVO Training Accuracy: 0.6762
OVO Validation Accuracy: 0.6590
```

Manual One-vs-One

In [96]:

```
nb_lab_con = GaussianNB()
```

In [97]:

```
beps_lab_con = beps[beps.vote != 2]
y_lab_con = beps_lab_con.vote
X_lab_con = beps_lab_con.drop('vote', axis='columns')
```

In [98]:

```
X_lab_con_train, X_lab_con_test, y_lab_con_train, y_lab_con_test = train_test_split(X_lab_con, y_lab_con, test_si
ze=0.2)
```

In [99]:

```
nb_lab_con.fit(X_lab_con_train, y_lab_con_train);
```

In [100]:

```
print("Labour vs. Conservative Training Accuracy: %.4f" % nb_lab_con.score(X_lab_con_train, y_lab_con_train))
print("Labour vs. Conservative Validation Accuracy: %.4f" % nb_lab_con.score(X_lab_con_test, y_lab_con_test))
```

```
Labour vs. Conservative Training Accuracy: 0.8392
Labour vs. Conservative Validation Accuracy: 0.8608
```

In [101]:

```
nb_lab_lib = GaussianNB()
```

In [102]:

```
beps_lab_lib = beps[beps.vote != 1]
y_lab_lib = beps_lab_lib.vote
X_lab_lib = beps_lab_lib.drop('vote', axis='columns')
```

In [103]:

```
X_lab_lib_train, X_lab_lib_test, y_lab_lib_train, y_lab_lib_test = train_test_split(X_lab_lib, y_lab_lib, test_si
ze=0.2)
```

In [104]:

```
nb_lab_lib.fit(X_lab_lib_train, y_lab_lib_train);
```

In [105]:

```
print("Labour vs. Libral Democrat Training Accuracy: %.4f" % nb_lab_lib.score(X_lab_lib_train, y_lab_lib_train))
print("Labour vs. Libral Democrat Validation Accuracy: %.4f" % nb_lab_lib.score(X_lab_lib_test, y_lab_lib_test))
```

```
Labour vs. Libral Democrat Training Accuracy: 0.7294
Labour vs. Libral Democrat Validation Accuracy: 0.6667
```

In [106]:

```
nb_con_lib = GaussianNB()
```

In [107]:

```
beps_con_lib = beps[beps.vote != 0]
y_con_lib = beps_con_lib.vote
X_con_lib = beps_con_lib.drop('vote', axis='columns')
```

In [108]:

```
X_con_lib_train, X_con_lib_test, y_con_lib_train, y_con_lib_test = train_test_split(X_con_lib, y_con_lib, test_si
ze=0.2)
```

In [109]:

```
nb_con_lib.fit(X_con_lib_train, y_con_lib_train);
```

In [110]:

```
print("Conservative vs. Libral Democrat Training Accuracy: %.4f" % nb_con_lib.score(X_con_lib_train, y_con_lib_tr
ain))
print("Conservative vs. Libral Democrat Validation Accuracy: %.4f" % nb_con_lib.score(X_con_lib_test, y_con_lib_t
est))
```

```
Conservative vs. Libral Democrat Training Accuracy: 0.7764
Conservative vs. Libral Democrat Validation Accuracy: 0.8323
```

In [111]:

```
pred_lab_con = nb_lab_con.predict(X_test)
pred_lab_lib = nb_lab_lib.predict(X_test)
pred_con_lib = nb_con_lib.predict(X_test)
```

In [112]:

```
pred = []
for i in range(y_test.shape[0]):
    if pred_lab_con[i] == pred_lab_lib[i] or pred_lab_con[i] == pred_con_lib[i]:
        pred.append(pred_lab_con[i])
    elif pred_lab_lib[i] == pred_con_lib[i]:
        pred.append(pred_lab_lib[i])
    else:
        pred.append(pred_con_lib[i])
```

In [113]:

```
accuracy_score(pred, y_test)
```

Out[113]:

```
0.6721311475409836
```

Ensemble Model Analysis

In [114]:

```
# list best models
best_models = [
               SVC(**svc_best_params),
               MLPClassifier(**mlp_best_params),
               RandomForestClassifier(**rf_best_params),
               KNeighborsClassifier(**knn_best_params)
]

names = [
        'SVC',
        'MLP',
        'RF',
        'KNN'
]

ensemble_models = list(zip(names, best_models))
```

Stacking Approach

Stacked generalization consists in stacking the output of individual estimator and use a classifier to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

In [115]:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression

clf = StackingClassifier(estimators=ensemble_models, final_estimator=LogisticRegression())
```

In [116]:

```
ens_model = clf.fit(X_train, y_train)
```

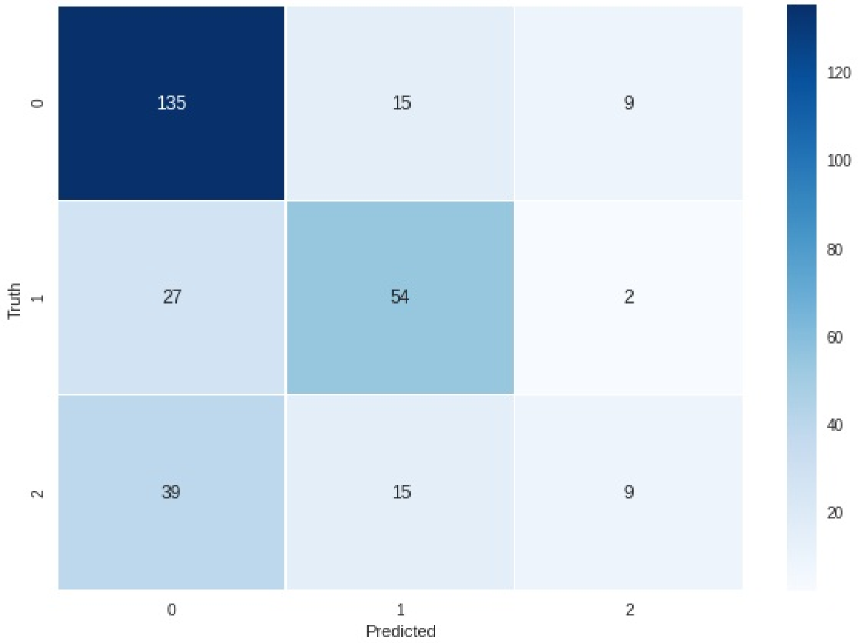
In [117]:

```
print(clf.score(X_test, y_test))
```

```
0.6491803278688525
```

In [118]:

```
# confusion matrix
predictions = ens_model.predict(X_test)
plot_confusion_matrix(y_test, predictions)
```




```
Classification Report
              precision    recall  f1-score   support

           0       0.67      0.85      0.75       159
           1       0.64      0.65      0.65        83
           2       0.45      0.14      0.22        63

    accuracy                           0.65       305
   macro avg       0.59      0.55      0.54       305
weighted avg       0.62      0.65      0.61       305
```

Bagging Approach

In [119]:

```
from sklearn.ensemble import BaggingClassifier

parameters = {
    'max_samples': [0.5, 0.6, 0.8, 1.0],
    'max_features': [0.5, 0.6, 0.8, 1.0]
}

bagging_models = []

for name, model in ensemble_models:
  m = GridSearchCV(estimator=BaggingClassifier(model), param_grid=parameters, cv=5, scoring=f1_scorer, n_jobs=-1,
verbose=2)
  m.fit(X_train, y_train)
  bagging_models.append(m)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:   15.7s finished

Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  5.4min finished

Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   51.8s
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  1.8min finished

Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:    2.8s finished
```

In [120]:

```
train_scores = []

for model in bagging_models:
  train_scores.append(m.best_score_)
print(train_scores)
test_scores = []
for model in bagging_models:
  sc = model.best_estimator_.score(X_test, y_test)
  test_scores.append(sc)
print(test_scores)
```

```
[0.6532786885245901, 0.6532786885245901, 0.6532786885245901, 0.6532786885245901]
[0.6918032786885245, 0.6557377049180327, 0.6655737704918033, 0.6360655737704918]
```

In [121]:

```
model_scores = list(zip(names, train_scores))
for name, score in model_scores:
  print("Train score for %s: %f" % (name,score))

model_scores = list(zip(names, test_scores))
for name, score in model_scores:
  print("Test score for %s: %f" % (name,score))
```

```
Train score for SVC: 0.653279
Train score for MLP: 0.653279
Train score for RF: 0.653279
Train score for KNN: 0.653279
Test score for SVC: 0.691803
Test score for MLP: 0.655738
Test score for RF: 0.665574
Test score for KNN: 0.636066
```

Boosting Approach

In [122]:

```
from sklearn.ensemble import AdaBoostClassifier

parameters = {
    'n_estimators': [50, 100, 150, 200]
}

grid = GridSearchCV(estimator=AdaBoostClassifier(), param_grid=parameters, cv=5, scoring=f1_scorer, n_jobs=-1, ve
rbose=2)
```

In [123]:

```
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed:    4.2s finished
```

Out[123]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0, n_estimators=50,
                                          random_state=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'n_estimators': [50, 100, 150, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(f1_score, average=micro), verbose=2)
```

In [124]:

```
grid.best_score_
```

Out[124]:

```
0.659016393442623
```

In [125]:

```
grid.best_estimator_.score(X_test, y_test)
```

Out[125]:

```
0.6721311475409836
```

Voting Approach

In [126]:

```
from sklearn.ensemble import VotingClassifier

clf = VotingClassifier(estimators=ensemble_models, voting='hard')
clf.fit(X_train, y_train)
print("Test scores: %f" % clf.score(X_test, y_test))
```

```
Test scores: 0.649180
```

AutoML

In [127]:

```
pipeline_optimizer = TPOTClassifier(generations=5, cv=5, random_state=42, verbosity=2)
```

In [128]:

```
pipeline_optimizer.fit(X_train, y_train)
```

```
Generation 1 - Current best internal CV score: 0.6811475409836066
Generation 2 - Current best internal CV score: 0.6811475409836066
Generation 3 - Current best internal CV score: 0.6852459016393443
Generation 4 - Current best internal CV score: 0.6852459016393443

Best pipeline: XGBClassifier(PCA(input_matrix, iterated_power=10, svd_solver=randomized), learning_r
ate=0.001, max_depth=9, min_child_weight=7, n_estimators=100, nthread=1, subsample=0.45)
```

Out[128]:

```
TPOTClassifier(config_dict=None, crossover_rate=0.1, cv=5,
               disable_update_check=False, early_stop=None, generations=5,
               log_file=<ipykernel.iostream.OutStream object at 0x7ff647c3d940>,
               max_eval_time_mins=5, max_time_mins=None, memory=None,
               mutation_rate=0.9, n_jobs=1, offspring_size=None,
               periodic_checkpoint_folder=None, population_size=100,
               random_state=42, scoring=None, subsample=1.0, template=None,
               use_dask=False, verbosity=2, warm_start=False)
```

In [129]:

```
print("AutoML Training Accuracy: %.4f" % pipeline_optimizer.score(X_train, y_train))
print("AutoML Validation Accuracy: %.4f" % pipeline_optimizer.score(X_test, y_test))
```

```
AutoML Training Accuracy: 0.7328
AutoML Validation Accuracy: 0.6459
```

AUC

In [130]:

```
auc_y = label_binarize(y_test, classes=[0, 1, 2])
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'yellow'])
```

In [131]:

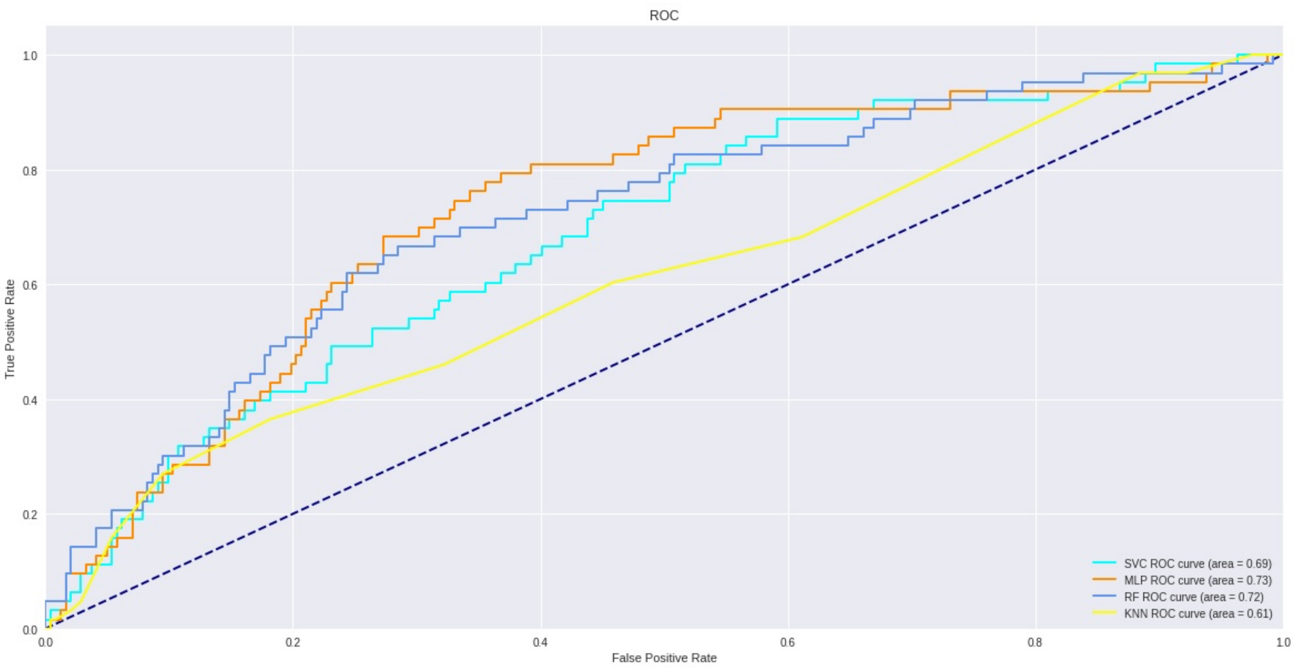
```
def auc_curve(model, y_test, color, name):
    clf = OneVsRestClassifier(model)
    clf.fit(X_train, y_train)
    if hasattr(clf, "decision_function"):
        y_score = clf.decision_function(X_test)
    else:
        y_score = clf.predict_proba(X_test)

    n_classes=3

    # Compute ROC curve for the model with the best paramater
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.plot(fpr[2], tpr[2], color=color,lw=lw, label='%s ROC curve (area = %0.2f)' % (name, roc_auc[2]))
```

In [132]:

```
auc_models = list(zip(colors, ensemble_models))
```

In [133]:

```
# Plot
plt.figure(figsize=(20, 10))
lw = 2
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

for color, (name, model) in auc_models:
  auc_curve(model, auc_y, color, name)


plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")

plt.show()
```



References

- https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/ (https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/)
- https://epistasislab.github.io/tpot/using/ (https://epistasislab.github.io/tpot/using/)
- https://scikit-learn.org (https://scikit-learn.org)
- https://github.com/corymaklin/svm/blob/master/svm.ipynb (https://github.com/corymaklin/svm/blob/master/svm.ipynb)
- https://scikit-learn.org/stable/auto_examples/datasets/plot_random_dataset.html (https://scikit-learn.org/stable/auto_examples/datasets/plot_random_dataset.html)
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html (https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html)
- https://stats.stackexchange.com/questions/437072/use-f1-score-in-gridsearchcv (https://stats.stackexchange.com/questions/437072/use-f1-score-in-gridsearchcv)
- https://matplotlib.org/3.1.1/tutorials/toolkits/mplot3d.html (https://matplotlib.org/3.1.1/tutorials/toolkits/mplot3d.html)
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)
- https://docs.w3cub.com/scikit_learn/auto_examples/model_selection/plot_validation_curve/ (https://docs.w3cub.com/scikit_learn/auto_examples/model_selection/plot_validation_curve/)
- https://matplotlib.org/3.1.1/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.html (https://matplotlib.org/3.1.1/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.html)
- https://pythonprogramming.net/matplotlib-3d-scatterplot-tutorial/ (https://pythonprogramming.net/matplotlib-3d-scatterplot-tutorial/)
- https://stackoverflow.com/questions/55375515/change-marker-color-in-3d-scatter-plot-based-on-condition (https://stackoverflow.com/questions/55375515/change-marker-color-in-3d-scatter-plot-based-on-condition)
- https://stackoverflow.com/questions/19451400/matplotlib-scatter-marker-size (https://stackoverflow.com/questions/19451400/matplotlib-scatter-marker-size)
- https://github.com/dataprofessor/code/blob/master/python/ROC_curve.ipynb (https://github.com/dataprofessor/code/blob/master/python/ROC_curve.ipynb)