In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
from tqdm import tqdm
from sklearn.inspection import plot_partial_dependence
from sklearn.linear_model import ElasticNetCV
from sklearn.linear_model import ElasticNet
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
import numpy as np
import pandas as pd
```

Conceptual: Cost functions for classification trees

```
1. (15 points) Consider the Gini index, classification error, an
d cross-entropy in simple classification settings with two class
es. Of these three possible cost functions, which would be best
 to use when growing a decision tree? Which would be best to use
 when pruning a decision tree? Why?
```

When growing a decision tree, cross-entropy or the Gini index are potential evaluation measures; they are both sensitive to node purity and control for variance across classes (which helps prevent overfitting). Therefore, either of these would be better than using classification error as an evlauation measure, given that it is not sensitive enough. However, there is a tradeoff to consider, given that classification error rate might be better to use in situations where one prioritizes prediction accuracy in the final pruned tree.

Strobl, C., Boulesteix, A. L., & Augustin, T. (2007). Unbiased split selection for classification trees based on the Gini index. Computational Statistics & Data Analysis, 52(1), 483-501.

In [ ]:

```

```

Application: Predicting attitudes towards racist college professors In this problem set, you are going to predict attitudes towards racist college professors, using the GSS survey data. Specifically, each respondent was asked "Should a person who believes that Blacks are genetically inferior be allowed to teach in a college or university?" Given the kerfuffle over Richard J. Herrnstein and Charles Murray's The Bell Curve and the ostracization of Nobel laureate James Watson over his controversial views on race and intelligence, this analysis will provide further insight into the public debate over this issue.

gss_*.csv contains a selection of features from the 2012 GSS. The outcome of interest colrac is a binary variable coded as either ALLOWED or NOT ALLOWED, where 1 = the racist professor should be allowed to teach, and 0 = the racist professor should not be allowed to teach. Documentation for the other predictors (if the variable is not clearly coded) can be viewed here. Some data pre-processing has been done in advance for you to ease your model fitting: (1) Missing values have been imputed; (2) Categorical variables with low-frequency classes had those classes collapsed into an "other" category; (3) Nominal variables with more than two classes have been converted to dummy variables; and (4) Remaining categorical variables have been converted to integer values, stripping their original labels

Your mission is to bring trees into the context of other classification approaches, thereby constructing a series of models to accurately predict an individual's attitude towards permitting racist professors to teach. The learning objectives of this exercise are:

```
1. Implement a battery of learners (including trees)
2. Tune hyperparameters
3. Substantively evaluate models
```

In [2]:
```
gss_tr = pd.read_csv("gss_train.csv")
```

In [3]:
```
gss_te = pd.read_csv("gss_test.csv")
```

In [4]:
```
x_tr = gss_tr.drop(['colrac'], axis=1)
y_tr = gss_tr['colrac']
X_te = gss_te.drop(['colrac'], axis=1)
Y_te = gss_te['colrac']
```

In [46]:
```
x_tr.head() #making sure colrac was dropped (what we're regressing on)
```

Out[46]:

	age	attend	authoritarianism	black	born	childs	colath	colcom	colmil	colhomo	...	partyi
0	21	0	4	0	0	0	1	0	1	1	...	
1	42	0	4	0	0	2	0	1	0	0	...	
2	70	1	1	1	0	3	0	1	0	0	...	
3	35	3	2	0	0	2	0	0	1	0	...	
4	24	3	6	0	1	3	1	0	0	0	...	

5 rows × 55 columns

In [48]:
```
X_te.head() #making sure colrac was dropped (what we're regressing on)
```

Out[48]:

	age	attend	authoritarianism	black	born	childs	colath	colcom	colmil	colhomo	...	partyi
0	22	2	1	0	0	0	0	1	1	0	...	
1	49	0	4	0	0	2	0	1	0	0	...	
2	50	0	3	0	0	2	1	0	1	0	...	
3	55	4	3	0	1	6	1	1	1	0	...	
4	22	3	4	0	0	0	1	0	1	1	...	

5 rows × 55 columns

```
2. Estimate the models (35 points; 5 points/model) Estimate the
 following models, predicting colrac using the training set (the
 training .csv) with 10-fold CV:
```

- Logistic regression
- Naive Bayes
- Elastic net regression
- Decision tree (CART)
- Bagging
- Random forest
- Boosting

Did out of order, but included all

Tune the relevant hyperparameters for each model as necessary. Only use the tuned model with the best performance for the remaining exercises. Be sure to leave sufficient time for hyperparameter tuning. Grid searches can be computationally taxing and take quite a while, especially for tree-aggregation methods.

In [29]:

```
#Logistic regression

logistic_reg = LogisticRegression()

#Accuracy
#calculating cv score, taking mean
logistic_reg_acc = np.mean(cross_val_score(logistic_reg, x_tr, y_tr, sc
oring = 'accuracy', cv=10))

#AUC/ROC
#calculating cv score, taking mean
```

```
logistic_roc_auc = np.mean(cross_val_score(logistic_reg, x_tr, y_tr, sc
oring = 'roc_auc', cv=10))
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
```

```
odel/logistic.py:432: FutureWarning: Default solver will be changed to

'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m
odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
/Users/Sruti/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_m

odel/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [30]:
```
print(logistic_reg_acc)
print(logistic_roc_auc)
```

```
0.7926804423928105
0.8703107556427476
```

In [22]:
```
#Naive Bayes
naive_bayes = GaussianNB()

#AUC/ROC
#calculating cv score, taking mean

roc_ac_nb = np.mean(cross_val_score(naive_bayes, x_tr, y_tr, scoring =
'roc_auc', cv=10))

#Accuracy
acc_nb = np.mean(cross_val_score(naive_bayes, x_tr, y_tr, scoring = 'ac
curacy', cv=10))
#calculating cv score, taking mean


#10-fold for both, using built-in sklearn methods
```

In [25]:
```
print(acc_nb) #NB accuracy output, lower than reg
print(roc_ac_nb) #NB roc/auc output, lower than reg
```

```
0.7344474902240962
0.8080500250922787
```

In [26]:
```
#Random Forest
```

In [7]:
```
random_forest = RandomForestClassifier()
```

```
param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50],
'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90],
'max_features': [5, 10, 20, 30, 40, 50] }

#max_depth: The maximum depth of the tree
#n_estimators: The number of trees in the forest.
#max_features: The number of features to consider when looking for the
 best split
#verbose: Controls the verbosity when fitting and predicting
#n_jobs: # of jobs to run in parallel, -1 means using all processors

random_forest_grid = GridSearchCV(estimator = random_forest, param_grid
 = param_grid,
cv = 10, n_jobs = -1, verbose = 2)

random_forest_grid.fit(x_tr, y_tr)
```

```
Fitting 10 folds for each of 324 candidates, totalling 3240 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    5.5s
[Parallel(n_jobs=-1)]: Done 230 tasks      | elapsed:   15.8s
[Parallel(n_jobs=-1)]: Done 636 tasks      | elapsed:   41.4s
[Parallel(n_jobs=-1)]: Done 1075 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 1581 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 2249 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 2859 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 3240 out of 3240 | elapsed:  3.9min finishe
d
```

Out[7]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_wei
ght=None,
                                              criterion='gini', max_dep
th=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.
0
```

```
0,
                                                            min_impurity_split=None,

                                                            min_samples_leaf=1,
                                                            min_samples_split=2,
                                                            min_weight_fraction_leaf=
0.0,
                                                            n_estimators='warn', n_jo
bs=None,
                                                            oob_score=False,
                                                            random_state=None, verbos
e=0,
                                                            warm_start=False),
                 iid='warn', n_jobs=-1,
                 param_grid={'max_depth': [10, 20, 30, 40, 50, 60, 70, 80,
90],
                             'max_features': [5, 10, 20, 30, 40, 50],
                             'n_estimators': [5, 10, 20, 30, 40, 50]},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=Fa
lse,
                 scoring=None, verbose=2)
```

In [8]:

```
random_forest_grid.best_params_  #using these 'best' in next, final ste
p
```

Out[8]:

```
{'max_depth': 10, 'max_features': 10, 'n_estimators': 40}
```

In [9]:

```
random_forest_final = RandomForestClassifier(n_estimators=40, max_depth
=10, max_features=10)


#Accuracy
##calculating cv score, taking mean
rf_accuracy = np.mean(cross_val_score(random_forest_final, x_tr, y_tr,
scoring = 'accuracy', cv=10))


#AUC/ROC
#calculating cv score, taking mean
```

```
rf_roc_auc = np.mean(cross_val_score(random_forest_final, x_tr, y_tr, s
coring = 'roc_auc', cv=10))
```

In [10]:
```
print(rf_accuracy) #random forest accuracy; better than nb
print(rf_roc_auc) #random forest auc/roc; better than nb
```

```
0.7919633371215347
0.8774826682371953
```

In [5]:
```
#Decision tree
decision_tree = DecisionTreeClassifier()

#Accuracy
##calculating cv score, taking mean
decision_tree_acc = np.mean(cross_val_score(decision_tree, x_tr, y_tr,
scoring = 'accuracy', cv=10))


#AUC/ROC
#calculating cv score, taking mean
dt_roc_auc = np.mean(cross_val_score(decision_tree, x_tr, y_tr, scoring
 = 'roc_auc', cv=10))
```

In [6]:
```
print(decision_tree_acc) #decision tree accuracy; roughly the same as n
b
print(dt_roc_auc) #decision tree roc/auc; not as good as nb or random f
orest
```

```
0.7303381555778488
0.7184698869205912
```

In [11]:
```
#Elastic net regression

#Tuned alpha and l1, ratio = 0.5 and alpha = 0.004

tune = ElasticNetCV(cv=10)
tune.fit(x_tr, y_tr)
alpha = tune.alpha_
```

```
l1 = tune.l1_ratio
print(alpha, l1)
```

```
0.0038452641680228584 0.5
```

In [12]:
```
#using the alpha and l1 obtained above
elastic_net = ElasticNet(alpha=0.0038452641680228584, l1_ratio=0.5)

#regression, mse
elastic_net_acc = np.mean(cross_val_score(elastic_net, x_tr, y_tr, scor
ing='neg_mean_squared_error', cv=10))

#AUC/ROC still
en_roc_auc= np.mean(cross_val_score(elastic_net, x_tr, y_tr, scoring =
'roc_auc', cv=10))
```

In [14]:
```
#elastic_net_acc = 0.1471453221731916, using mse, elastic net accuracy
#en_roc_auc = 0.8740225489138439, elastic net roc/auc; almost as high a
s rf
```

In [16]:
```
#Bagging

#staying consistent with n_estimators from random forest
param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50]}
```

In [17]:
```
bagging = BaggingClassifier()

bagging_grid = GridSearchCV(estimator = bagging, param_grid = param_gri
d,
cv = 10, n_jobs = -1, verbose = 2)

#max_depth: The maximum depth of the tree
#n_estimators: The number of trees in the forest.
#max_features: The number of features to consider when looking for the
 best split
#verbose: Controls the verbosity when fitting and predicting
#n_jobs: # of jobs to run in parallel, -1 means using all processors
```

```
bagging_grid.fit(x_tr, y_tr)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    7.6s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:   12.9s finished
```

Out[17]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=BaggingClassifier(base_estimator=None, bootstrap
=True,
                                         bootstrap_features=False,
                                         max_features=1.0, max_samples=
1.0,
                                         n_estimators=10, n_jobs=None,
                                         oob_score=False, random_state=
None,
                                         verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'n_estimators': [5, 10, 20, 30, 40, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fa
lse,
             scoring=None, verbose=2)
```

In [18]:

```
bagging_grid.best_params_  #using this in next, final step
```

Out[18]:

```
{'n_estimators': 40}
```

In [19]:

```
bagging_fit = BaggingClassifier(n_estimators=40)

#Accuracy
##calculating cv score, taking mean
bagging_acc = np.mean(cross_val_score(bagging_fit, x_tr, y_tr, scoring
= 'accuracy', cv=10))

#AUC/ROC
##calculating cv score, taking mean
```

```
b_roc_auc = np.mean(cross_val_score(bagging_fit, x_tr, y_tr, scoring =
'roc_auc', cv=10))
```

In [20]:
```
print(bagging_acc) #bagging accuracy, a little lower than rf
print(b_roc_auc) #bagging roc/auc; almost as high as rf, higher than el
astic net
```

```
0.7763816754708414
0.8749782762509121
```

In [21]:
```
#Gradient Boosting

#keeping n_estimators consistent with rf and bagging

param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50], 'learning_rate':
 [0.1, 0.4, 0.7, 1], 'max_features': [None, 'sqrt']}
```

In [22]:
```
grad_boost = GradientBoostingClassifier()

grad_boost_grid = GridSearchCV(estimator = grad_boost, param_grid = par
am_grid,
cv = 10, n_jobs = -1, verbose = 2)

#max_depth: The maximum depth of the tree
#n_estimators: The number of trees in the forest.
#max_features: The number of features to consider when looking for the
 best split
#verbose: Controls the verbosity when fitting and predicting
#n_jobs: # of jobs to run in parallel, -1 means using all processors

grad_boost_grid.fit(x_tr, y_tr)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done 108 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed:   16.7s finished
```

```
Out[22]: GridSearchCV(cv=10, error_score='raise-deprecating',
                      estimator=GradientBoostingClassifier(criterion='friedman_m
         se',
                                                           init=None, learning_r
         ate=0.1,
                                                           loss='deviance', max_
         depth=3,
                                                           max_features=None,
                                                           max_leaf_nodes=None,
                                                           min_impurity_decrease
         =0.0,
                                                           min_impurity_split=No
         ne,
                                                           min_samples_leaf=1,
                                                           min_samples_split=2,
                                                           min_weight_fraction_l
         eaf=0.0,
                                                           n_estimators=100,
                                                           n_iter_no_change=Non
         e,
                                                           presort='auto',
                                                           random_state=None,
                                                           subsample=1.0, tol=0.
         0001,
                                                           validation_fraction=
         0.1,
                                                           verbose=0, warm_start
         =False),
                      iid='warn', n_jobs=-1,
                      param_grid={'learning_rate': [0.1, 0.4, 0.7, 1],
                                  'max_features': [None, 'sqrt'],
                                  'n_estimators': [5, 10, 20, 30, 40, 50]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=Fa
         lse,
                      scoring=None, verbose=2)
```

```
In [23]: grad_boost_grid.best_params_ #using these in next, final step
```

```
Out[23]: {'learning_rate': 0.1, 'max_features': None, 'n_estimators': 50}
```

In [25]:
```
grad_boost_fit = GradientBoostingClassifier(learning_rate=0.1, max_feat
ures='sqrt', n_estimators=50)

#Accuracy
##calculating cv score, taking mean
grad_acc = np.mean(cross_val_score(grad_boost_fit, x_tr, y_tr, scoring
= 'accuracy', cv=10))

#AUC/ROC
##calculating cv score, taking mean
grad_roc_auc= np.mean(cross_val_score(grad_boost_fit, x_tr, y_tr, scori
ng = 'roc_auc', cv=10))
```

In [26]:
```
print(grad_acc) #gradient boost accuracy - higher than rf
print(grad_roc_auc) #gradient boost roc/auc - slightly lower than rf
```

```
0.8021627606239304
0.874622962550528
```

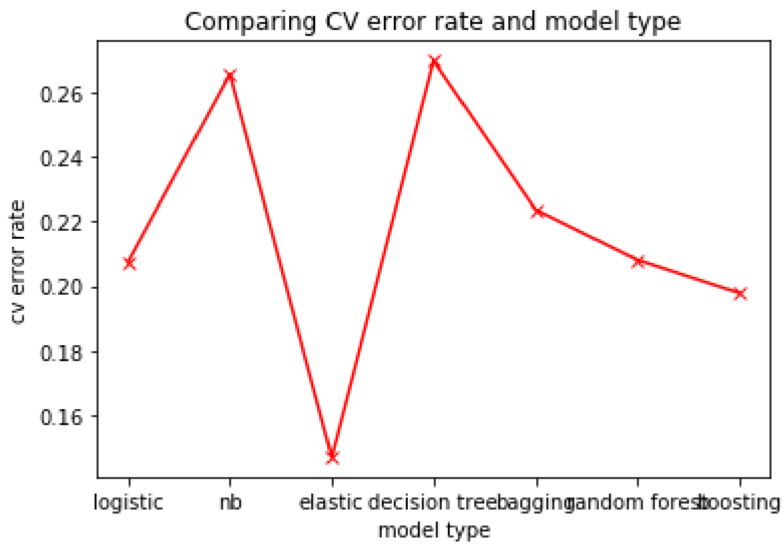
3. Evaluate the models (20 points) Compare and present each model's (training) performance based on

- Cross-validated error rate
- ROC/AUC

In [ ]:
```
#Accuracy

cver = list(1 - np.array([0.79268, 0.73445, 0.853, 0.73034, 0.77642, 0.
79196, 0.80222]))
#listing all the error scores we've obtained, 1-acc for each model outp
ut
#elastic net = 1-mse output
```

In [36]:
```
model = ['logistic', 'nb', 'elastic', 'decision tree', 'bagging', 'rand
```

```
om forest', 'boosting']
#all the models I ran

#graphing
plt.plot(model, cver, marker='x', color='red')

plt.xlabel('model type')
plt.ylabel('cv error rate')
plt.title('Comparing CV error rate and model type');
```



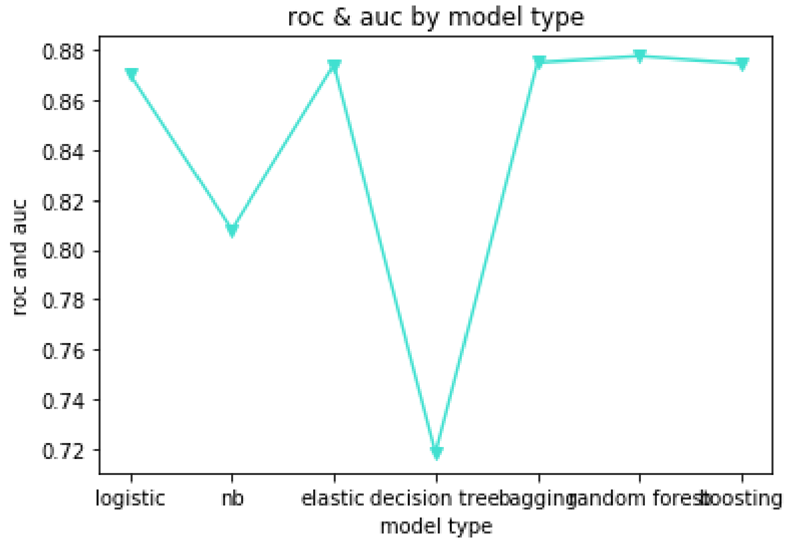
In [ ]:

```
#AUC/ROC

#all the auc/roc i've obtained
roc_auc = [0.8703, 0.8081, 0.8740, 0.7185, 0.8750, 0.8775, 0.8746]

#all the models I ran
model = ['logistic', 'nb', 'elastic', 'decision tree', 'bagging', 'rand
om forest', 'boosting']
```

In [32]:

```
#graphing
plt.plot(model, roc_auc, marker='v', color='turquoise')
```

```
plt.xlabel('model type')
plt.ylabel('roc and auc')
plt.title('roc & auc by model type');
```



4. (15 points) Which is the best model? Defend your choice.

Based on both error plots: boosting has the lowest cv error rate (described in terms of acc) and the third highest roc/auc (though, very close to the two highest: random forest and bagging, which is why I'd still consider it the best). Logistic regression also performs very well (very low cv error rate, very high roc/auc). Elastic net had very low mse, but extremely low roc/auc.

5. Evaluate the best model (15 points) Evaluate the final, best model's (selected in 4) performance on the test set (the test . csv) by calculating and presenting the classification error rate and AUC. Compared to the fit evaluated on the training set in q

uestions 3-4, does the "best" model generalize well? Why or why not? How do you know?

In [67]:
```
grad_boost_fit.fit(x_tr, y_tr) #using the same fit function from earlie
r
```

Out[67]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.2, loss='deviance', max_dept
h=3,
                           max_features='sqrt', max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_spli
t=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=4
0,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.000
1,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [71]:
```
#fitting to test data and calculating corresponding acc and auc/roc sco
res:

test_error_gb= 1 - accuracy_score(Y_te, grad_boost_fit.predict(X_te))


gb_roc_auc= roc_auc_score(Y_te, grad_boost_fit.predict(X_te))
```

In [72]:
```
print(test_error_gb) #(test) gradient boost accuracy
print(gb_roc_auc) #test auc roc gradient boost
```

```
0.21501014198782964
0.7776481297583582
```

The "best" model, gradient boost, doesn't generalize as well - the test roc/auc (0.778) is lower than the training roc/auc (.8768322053533322), which suggests that the model overfit to the training data.

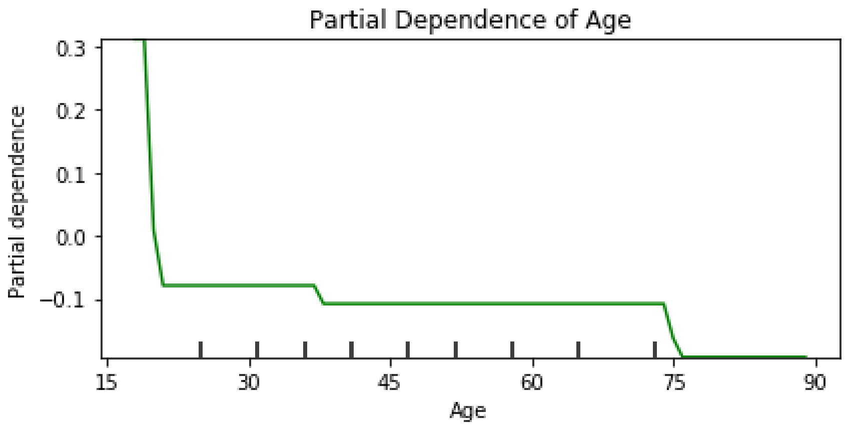
```
6. Bonus: PDPs/ICE (Up to 5 extra points) Present and substantiv
ely interpret the "best" model (selected in question 4) using PD
Ps/ICE curves over the range of: tolerance and age. Note, interp
retation must be more than simple presentation of plots/curves.
 You must sufficiently describe the changes in probability estim
ates over the range of these two features. You may earn up to 5
 extra points, where partial credit is possible if the solution
 is insufficient along some dimension (e.g., technically/code, i
nterpretation, visual presentation, etc.).
```

In [ ]:

```
age = [0] #0th index
```

In [79]:

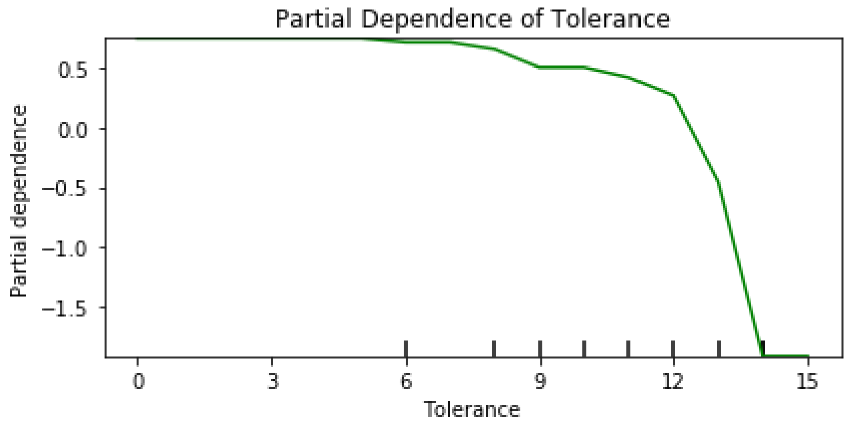
```
plot_partial_dependence(grad_boost_fit, x_tr, age)
plt.xlabel('Age')
plt.title('Partial Dependence of Age');
```



Age doesn't seem to be affecting colrac very much (no clear positive or negative correlation from plot), partial dependence is pretty much the same across age, save from roughly 15-18 where it starkly decreases.

In [ ]:
```
features = [32] #32nd index (used iloc to determine)
```

In [84]:
```
plot_partial_dependence(grad_boost_fit, x_tr, features)
plt.xlabel('Tolerance')
plt.title('Partial Dependence of Tolerance');
```



Looks like there's a strong negative correlation here (as tolerance increases, partial dependence decreases -> values given for colrac decrease - in other words, higher tolerance generally links with the opinion that racist profs should not be allowed to teach.

In [ ]: