# MACS30150 PS8 (Question 1)


### Dr. Richard Evans

### Submitted by Junho Choi


Let us import the necessary functions and packages.

In [21]:
```
import pandas as pd
import numpy as np
import graphviz
import os
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
from sklearn.ensemble import RandomForestRegressor as RanForReg
from sklearn.ensemble import RandomForestClassifier as RanForCla
from sklearn.linear_model import LogisticRegression as LR
from sklearn.svm import SVC
from sklearn.model_selection import LeaveOneOut, KFold
from sklearn.metrics import classification_report
```

Note that this below line of code was necessary for me to create the visualization for the tree graphs; others may have to adjust their PATH.

In [2]:
```
# os.environ["PATH"] += os.pathsep + 'D:/All/Documents/Graphviz/bin'
```

# Problem 1

## Problem 1-(a)

Let us first import the dataset from `biden.csv`.

In [22]:
```
biden = pd.read_csv('biden.csv')
```

Let us split the data into training ($70\%$) and test ($30\%$) data, as the question directs.

In [23]:
```
y = biden['biden']
colnames = list(biden.columns)
colnames.remove('biden')
X = biden[colnames]

yvals = y.values
Xvals = X.values

X_tr, X_te, y_tr, y_te = W
    train_test_split(Xvals, yvals, test_size=0.3, random_state=25)
```

Now, having set the training and test datasets, let us use the `DecisionTreeRegressor` to fit the training data. As directed by the question, we will use the conditions `max_depth=3` and `min_samples_leaf=5`.

In [24]:
```
dec_tree = DecisionTreeRegressor(max_depth=3, min_samples_leaf=5)
```

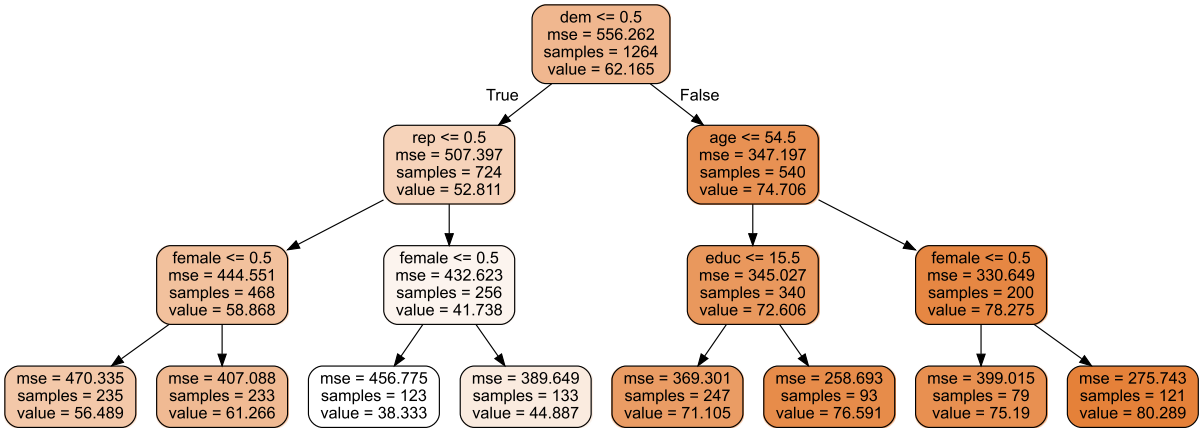
In [25]:
```
dec_tree_tr = dec_tree.fit(X_tr, y_tr)
y_pred = dec_tree_tr.predict(X_te)
```

Below code will plot the tree graph.

In [26]:
```
tree_graph = export_graphviz(
    dec_tree,
    out_file=None,
    feature_names=colnames,
    # class_names=iris.target_names,
    rounded=True,
    filled=True,
)

graph = graphviz.Source(tree_graph)
graph.render('tree_graph')
graph
```

Out[26]:




Let us now interpret the above result. Each "split" of the nodes are made based on minimization of the mean squared errors (MSE), and the "value" indicates the mean of the warmth for Biden (i.e. the `biden` variable). The first split is made based on the variable `dem` (i.e. whether you are affiliated with Democrats or not). Split to the left, indicating `True` for `dem <= 0.5` is for those who are not affiliated with Democrats; split to the right is. The second set of splits is made based on the variables `rep` (i.e. Republican affiliation) and `age` for the left and right sub-trees respectively. The third set of splits is made based on the variables `female`, `female`, `educ` (i.e. years of education), and `female`.

Using this information, we can find the end nodes (indicating subgroups) that are likely to be the most and least fervent supporters of Biden. In this dataset's training set, those who are affiliated with Democrats, are older than 54.5, and are female (the rightmost node) have the highest average `biden` value of 80.289; and therefore are the most likely to be supportive of Biden. On the other hand, those who are not affiliated with Democrats, who are affiliated with Republicans, and are not females (the third node from the left) have the lowest average `biden` value of 38.333; and therefore are the least likely to be supportive of Biden.

Finally, we can find the mean squared error of the test set based on the above model fit; it is found to be approximately $396.19$.

In [27]:
```
MSE_te = (((y_pred - y_te) ** 2).sum())/len(y_pred)
print(MSE_te)
```

```
396.193714632
```

## Problem 1-(b)

Let us conduct the hyperparameter tuning as direct by the question, using the below code chunks.

In [28]:
```
dec_tree_gen = DecisionTreeRegressor()

param_dist1 = {
    'max_depth': [3, 10],
    'min_samples_split': sp_randint(2, 20),
    'min_samples_leaf': sp_randint(2, 20)
}
```

In [29]:
```
random_search1 = ₩
    RandomizedSearchCV(dec_tree_gen, param_distributions=param_dist1,
                       n_iter=100, n_jobs=-1, cv=5, random_state=25,
                       scoring='neg_mean_squared_error')
```

In [30]:
```
random_search1.fit(Xvals, yvals)
print('RandBestEstimator1=', random_search1.best_estimator_)
print('RandBestParams1=', random_search1.best_params_)
print('RandBestScore1=', -random_search1.best_score_)
```

```
RandBestEstimator1= DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=17,
           min_samples_split=14, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
RandBestParams1= {'max_depth': 3, 'min_samples_leaf': 17, 'min_samples_split': 14}
RandBestScore1= 401.690360223
```

As seen from the above, the parameters that are calculated to be optimally tuned are `max_depth` being 3, `min_samples_leaf` being 17, and `min_samples_split` being 14. Finally, the MSE of the optimal results are calculated to be approximately $401.69$.

## Problem 1-(c)

As the random forest regression relies on randomness as well, I initialized it with `random_state=25` as well. Using the below set of codes, let us conduct the hyperparameter tuning for random forest regression.

In [31]:
```
rfr_gen = RanForReg(random_state=25)

param_dist2 = {
    'n_estimators': [10, 200],
    'max_depth': [3, 10],
    'min_samples_split': sp_randint(2, 20),
    'min_samples_leaf': sp_randint(2, 20),
    'max_features': sp_randint(1, 5)
}
```

In [32]:
```
random_search2 = ₩
    RandomizedSearchCV(rfr_gen, param_distributions=param_dist2,
                       n_iter=100, n_jobs=-1, cv=5, random_state=25,
                       scoring='neg_mean_squared_error')
```

In [33]:
```
random_search2.fit(Xvals, yvals)
print('RandBestEstimator2=', random_search2.best_estimator_)
print('RandBestParams2=', random_search2.best_params_)
print('RandBestScore2=', -random_search2.best_score_)
```

```
RandBestEstimator2= RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=3,
           max_features=2, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=17,
           min_samples_split=13, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=25,
           verbose=0, warm_start=False)
RandBestParams2= {'max_depth': 3, 'max_features': 2, 'min_samples_leaf': 17, 'min_samples_split': 13, 'n_es
timators': 10}
RandBestScore2= 397.068109012
```

Based on the above results, it is possible to find out of optimal tuning parameter values. They are as follows: `max_depth` being 3, `max_features` being 2, `min_samples_leaf` being 17, `min_samples_split` being 13, `n_estimators` being 10. The MSE of the optimal results can be calculated as being approximately $397.07$.

# MACS30150 PS8 (Question 2)

### Dr. Richard Evans

### Submitted by Junho Choi

Let us import the necessary functions and packages.

In [2]:
```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
from sklearn.ensemble import RandomForestRegressor as RanForReg
from sklearn.ensemble import RandomForestClassifier as RanForCla
from sklearn.linear_model import LogisticRegression as LR
from sklearn.svm import SVC
from sklearn.model_selection import LeaveOneOut, KFold
from sklearn.metrics import classification_report
```

## Problem 2

Before proceeding with the analyses in the sub-problems, let us read in the data. Before a previous problem set, we know that there exists missing data for displacement written as `?`. Therefore, we set the `na_values` as such and drop the rows with missing data.

In [3]:
```
auto = pd.read_csv('Auto.csv', na_values='?')
print(auto.shape)
```

```
(397, 9)
```

In [4]:
```
auto = auto.dropna(axis=0)
print(auto.shape) ## after dropping the NA values
```

```
(392, 9)
```

Let us create the variables `mpg_high`, `orgn1`, and `orgn2` as directed by the question.

In [5]:
```
med = auto['mpg'].median()
auto['mpg_high'] = 0
auto.loc[auto['mpg'] >= med, 'mpg_high'] = 1
```

In [6]:
```
auto['orgn1'] = 0
auto['orgn2'] = 0
auto.loc[auto['origin'] == 1, 'orgn1'] = 1
auto.loc[auto['origin'] == 2, 'orgn2'] = 1
```

The resulting dataset would look something like this.

In [7]:
```
auto.head(8)
```

Out[7]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | year | origin | name | mpg_high | orgn1 | orgn2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 18.0 | 8 | 307.0 | 130.0 | 3504 | 12.0 | 70 | 1 | chevrolet chevelle malibu | 0 | 1 | 0 |
| **1** | 15.0 | 8 | 350.0 | 165.0 | 3693 | 11.5 | 70 | 1 | buick skylark 320 | 0 | 1 | 0 |
| **2** | 18.0 | 8 | 318.0 | 150.0 | 3436 | 11.0 | 70 | 1 | plymouth satellite | 0 | 1 | 0 |
| **3** | 16.0 | 8 | 304.0 | 150.0 | 3433 | 12.0 | 70 | 1 | amc rebel sst | 0 | 1 | 0 |
| **4** | 17.0 | 8 | 302.0 | 140.0 | 3449 | 10.5 | 70 | 1 | ford torino | 0 | 1 | 0 |
| **5** | 15.0 | 8 | 429.0 | 198.0 | 4341 | 10.0 | 70 | 1 | ford galaxie 500 | 0 | 1 | 0 |
| **6** | 14.0 | 8 | 454.0 | 220.0 | 4354 | 9.0 | 70 | 1 | chevrolet impala | 0 | 1 | 0 |
| **7** | 14.0 | 8 | 440.0 | 215.0 | 4312 | 8.5 | 70 | 1 | plymouth fury iii | 0 | 1 | 0 |

Below code checks whether the two variables `orgn1` and `orgn2` have been successfully created.

In [8]:
```
## checking whether orgn1 and orgn2 variables have been created well
print(auto.groupby(['origin']).size())
print()
print(auto.orgn1.sum(), auto.orgn2.sum())
```

```
origin
1    245
2     68
3     79
dtype: int64

245 68
```

Let us set the y- and x-variables (i.e. the dependent variable and the regressors) to be used in the sub-problems.

In [9]:
```
y_auto = auto['mpg_high']
X_columns = ['cylinders', 'displacement', 'horsepower', 'weight',
             'acceleration', 'year', 'orgn1', 'orgn2']
X_auto = auto[X_columns]

X_auto_vals = X_auto.values
y_auto_vals = y_auto.values
```

## Problem 2-(a)

For this part, I use the (revised) function `cv_kf_mlogit` that I implemented from a previous problem set. It is as follows.

In [10]:
```
def cv_kf_mlogit(Xvals, yvals, k=4, rtn_vectors=True, random=25):
    ## creating the splits
    kf = KFold(n_splits=k, shuffle=True, random_state=random)
    kf.get_n_splits(Xvals)

    ## creating the vector of KFolds MSE
    MSE_vec_kf = np.zeros(k)
    k_ind = int(0)

    for train_idx, test_idx in kf.split(Xvals):
        ## splitting training and test sets
        X_tr_kf, X_test_kf = Xvals[train_idx], Xvals[test_idx]
        y_tr_kf, y_test_kf = yvals[train_idx], yvals[test_idx]

        ## making the prediction
        mlogit_kf = LR(random_state=random, solver='lbfgs',
                       multi_class='multinomial').fit(X_tr_kf, y_tr_kf)
        y_pred_kf = mlogit_kf.predict(X_test_kf)

        ## indicator function for categorical variables
        MSE_indicator_kf = (y_pred_kf != y_test_kf).mean()

        ## MSE vector for the specific k-fold
        MSE_vec_kf[k_ind] = MSE_indicator_kf
        print('Fold #{}:'.format(k_ind+1), 'MSE is {}'.format(MSE_indicator_kf))

        ## creating the vectors of actual and test yvals
        if k_ind == 0:
            y_act, y_pred = y_test_kf, y_pred_kf

        else:
            y_act = np.hstack((y_act, y_test_kf))
            y_pred = np.hstack((y_pred, y_pred_kf))

        k_ind += 1

    cv_kf = MSE_vec_kf.mean()
    print('--------------------------------')
    print('CV_KF is {}'.format(cv_kf))

    if rtn_vectors:
        return cv_kf, y_act, y_pred

    else:
        return cv_kf
```

Using the said function, the MSE of the K-folds (with $k = 4$) crossvalidation can be found as approximately $0.0969$.

In [11]:
```
cv_kf, y_act, y_pred = W
    cv_kf_mlogit(X_auto_vals, y_auto_vals, k=4,
                 rtn_vectors=True, random=25)
```

```
Fold #1: MSE is 0.14285714285714285
Fold #2: MSE is 0.09183673469387756
Fold #3: MSE is 0.07142857142857142
Fold #4: MSE is 0.08163265306122448
--------------------------------
CV_KF is 0.09693877551020408

C:WUsersWAdministratorWAnaconda3WlibWsite-packagesWsklearnWlinear_modelWlogistic.py:758: ConvergenceWarnin
g: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:WUsersWAdministratorWAnaconda3WlibWsite-packagesWsklearnWlinear_modelWlogistic.py:758: ConvergenceWarnin
g: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

In addition, the error rates (that is , $1 - \text{precision}$) for each group (i.e. groups `mpg_high=0` and `mpg_high=1` ) are as follows. For the low-mpg group (i.e. `mpg_high=1` ), the error rate is approximately $0.08$ ($= 1 - 0.92$). On the other hand, for the high-mpg group (i.e. `mpg_high=0` ), the error rate is approximately $0.11$ ($= 1 - 0.89$).

In [12]:
```
print(classification_report(y_act, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       196
           1       0.89      0.92      0.90       196

   micro avg       0.90      0.90      0.90       392
   macro avg       0.90      0.90      0.90       392
weighted avg       0.90      0.90      0.90       392
```

## Problem 2-(b)

Once again, because the random forest classifier also relies on randomness, let us set `random_state=25`. Using the below chunks of code, let us conduct hyperparameter tuning for random forest classifier.

In [13]:
```
rfc_gen = RanForCla(random_state=25)

param_dist3 = {
    'n_estimators': [10, 200],
    'max_depth': [3, 8],
    'min_samples_split': sp_randint(2, 20),
    'min_samples_leaf': sp_randint(2, 20),
    'max_features': sp_randint(1, 8)
}
```

In [14]:
```
random_search3 = ₩
    RandomizedSearchCV(rfc_gen, param_distributions=param_dist3,
                       n_iter=100, n_jobs=-1, cv=4, random_state=25,
                       scoring='neg_mean_squared_error')
```

In [18]:
```
random_search3.fit(X_auto_vals, y_auto_vals)
print('RandBestEstimator3=', random_search3.best_estimator_)
print('RandBestParams3=', random_search3.best_params_)
print('RandBestScore3=', -random_search3.best_score_)
```

```
RandBestEstimator3= RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=8, max_features=3, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=15, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
RandBestParams3= {'max_depth': 8, 'max_features': 3, 'min_samples_leaf': 15, 'min_samples_split': 2, 'n_est
imators': 10}
RandBestScore3= 0.08928571428571429
```

It can then be seen from above that the optimal tuning parameter values are as follows: `max_depth` of 8, `max_features` of 3, `min_samples_leaf` of 15, `min_samples_split` of 2, and finally `n_estimators` of 10. The MSE of optimal results is found to be approximately $0.0893$.

## Problem 2-(c)

Let us tune the hyperparameters for the support vector classifier ( `sklearn.svm.SVC` ) using the following chunks of code. As the question directs, I have initialized the kernel to be radial basis function ( `kernel=rbf` ).

In [22]:
```
svc = SVC(kernel='rbf')

param_dist4 = {
    'C': sp_uniform(loc=0.2, scale=4.0),
    'gamma': ['scale', 'auto'],
    'shrinking': [True, False]
}
```

In [23]:
```
random_search4 = W
   RandomizedSearchCV(svc, param_distributions=param_dist4,
                      n_iter=100, n_jobs=-1, cv=4, random_state=25,
                      scoring='neg_mean_squared_error')
```

In [21]:
```
random_search4.fit(X_auto_vals, y_auto_vals)
print('RandBestEstimator4=', random_search4.best_estimator_)
print('RandBestParams4=', random_search4.best_params_)
print('RandBestScore4=', -random_search4.best_score_)
```

```
RandBestEstimator4= SVC(C=1.8094629152568114, 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=False,
  tol=0.001, verbose=False)
RandBestParams4= {'C': 1.8094629152568114, 'gamma': 'scale', 'shrinking': False}
RandBestScore4= 0.11479591836734694
```

As seen from the above result, the optimized values for the parameters `C` (penalty parameter), `gamma` (kernel coefficient), and `shrinking` are as follows: (approximately) $1.8095$, `scale` , and `False` respectively. The MSE of the optimal results is approximately $0.1148$.

## Problem 2-(d)

From the above parts 2-(a) to 2-(c), the model in 2-(b) -- that is, random forest classifier with maximum depth of 8, maximum number of features of 3, minimum samples in leaf of 15, minimum samples split of 2, and number of estimators being 10 -- had the lowest value of MSE among the three models. Therefore, I would say the model in 2-(b) is the best predictor of `mpg_high` (based on MSE).