DATA 558 - Statistical Machine Learning

Spring 2023

Arjun Sharma

Conceptual Questions

Problem 1

Describe advantages and disadvantages of tree-based methods. Describe also the motivation behind each of the methods: bagging, random forests, and boosting. Compare and contrast these methods; make sure you describe a feature that is special/unique about each of them.

Solution

Advantages of tree-based models:

1. Handling Non-linearity: Trees can capture non-linear relationships between predictors and the target variable without explicitly requiring feature engineering or transformations. They can handle complex decision boundaries and interactions between features.

2. Robust to Outliers and null values: Tree-based methods are less affected by outliers in the data compared to algorithms like linear regression, Support Vector Machines, K-Nearest Neighbors, etc., They can also handle null values in the data.

3. Feature Importance: Tree-based methods can provide measures of feature importance, indicating which predictors have the most influence on the target variable, which is helpful during feature selection.

4. Non-parametric: Tree-based models make very few assumptions about the underlying data distribution which allows them to capture complicated relationships without imposing strict assumptions.

5. Robust Against Irrelevant Features: Tree-based methods can handle irrelevant features, as they identify the more informative features during the splitting process.

Disadvantages of tree-based models:

1. Overfitting: Trees have a tendency to overfit the training data, especially when the tree depth is not limited. This can lead to poor generalization performance on unseen data.

2. Sensitivity: Tree-based methods are sensitive to small changes in the training data, which can result in different trees and potentially different predictions.

3. Bias towards Dominant Features: Decision trees can be biased towards features with higher cardinality. This can lead to the dominance of certain features over others.

Motivation Behind Bagging, Random Forests, Boosting

Bagging: Bagging aims to reduce the variance and improve the stability of a model by generating multiple subsets of the training data and training multiple models on these subsets. Each model is trained independently, and their predictions are combined through aggregation to obtain the final prediction.

Random Forests: Random Forests aim to reduce the variance, sensitivity, and bias in individual decision trees by creating multiple decision trees that are uncorrelated, in order to improve generalization as a group.

Boosting: Boosting aims to sequentially build a strong ensemble by focusing on the misclassified samples from the previous models. The motivation is to improve the model's predictive performance by iteratively adjusting the weights or importance of training samples to give more emphasis to the difficult samples.

Comparison:

Bagging vs Boosting

1. Bagging uses bootstrapping to create subsets of the training data, while boosting adjusts the sample weights or residuals.

2. Bagging trains models independently, while boosting sequentially builds models.

3. Bagging reduces variance and overfitting, while boosting reduces bias and focuses on challenging samples.

4. Boosting typically yields higher predictive accuracy by focusing on challenging instances, but it may be more sensitive to noisy data.

Random Forest vs Boosting

1. Random forest uses bootstrapping to create subsets of the training data, while boosting adjusts the sample weights or residuals.
2. Random forest trains models independently, while boosting sequentially builds models.
3. Random forest reduces variance and overfitting, while boosting reduces bias and focuses on challenging samples.
4. Random forest introduces randomness in feature selection, whereas boosting adjusts the weights/importance of samples.
5. Boosting typically yields higher predictive accuracy by focusing on challenging instances, but it may be more sensitive to noisyd data.

Bagging vs Random Forest

1. Random forest is less prone to overfitting and often have better generalization performance than bagging.
2. Random Forest is an ensemble of only decision trees, whereas bagging by itself can be an ensemble of various different kinds of models not restricted to decision trees.

Unique Features:

Bagging:

Bagging aggregates the predictions of individual models through averaging or majority voting for regression and classification tasks respectively to obtain the final prediction.

Random Forests:

The unique feature of random forests is the random subset of features considered at each split in the decision tree. Random feature selection helps decorrelate the trees and encourages them to explore different aspects of the data, leading to a more diverse ensemble.

Boosting:

Boosting involved adaptive adjustment of sample weights or importance during the iterative process of giving rise to the strong learner as a result of the sequence of weak learners. It assigns higher weights to misclassified samples or the residuals of the previous models, emphasizing difficult samples over the sequence of weak learners.

Problem 2

Please determine whether following practices are good or not. Please explain why in either case.

(a)

We have a prediction task as hand and we turn to using decision trees to learn a prediction model. We remember that bagging is a useful way to improve their performance. So we use bagging to learn a prediction model based on an average of B trees. We show the final prediction model to our boss and they comment that the prediction model seems to be quite biased. So we increase B in our bagging procedure.

Solution:

This is bad practise. Increasing the number of trees in bagging can help reduce variance and improve the stability of the prediction model. However, it may not directly address the issue of bias.

It would be more effective to investigate the sources of bias in the model and address them. The person working on this bagging technique should look into other concerns like feature engineering, data quality, etc.,

They could also look at adjusting other hyperparameters, trying newer models or collecting even more data.

(b):

We are running a prediction model over a large number of predictors. For our purposes, computational cost of learning the prediction model is a huge priority, although of course we still do want a prediction model that is accurate. We remember that boosting algorithms perform very well and use them for our task.

Solution:

This is bad practise. Boosting Algorithms are one of the most computationally expensive Machine Learning models, and hence may not be contextually appropriate when we are trying to conserve our computational resources.

Instead, there should be more emphasis on feature selection and engineering when we have a large number of features.

Boosting is hard to interpret, and hence we would encounter more difficulty in trying to ascertain the more important features out of the large number of features at our disposal. Hence, using simpler models would be recommended as well.

(c)

We apply the same boosting algorithm (with the same weak learners) on multiple datasets generating from the same mechanism and notice that the output is highly variable. To decrease the variability, we use more expressive (more complex) learners.

Solution:

This is bad practise. Using more complex learners to decrease variability in the output of a boosting algorithm on multiple datasets generated from the same mechanism may lead to overfitting and worsen the performance of the boosting algorithm.

It can lead to overfitting and higher variance, making the model even more susceptible to noise, exacerbating the tendency of boosting models to be sensitive to noisy data.

Furthermore, this would make it even more computationally expensive to use more complex learners.

(d)

We apply a boosting algorithm with fixed B (number of iterations in the algorithm) and notice that the prediction model is biased. So we increase B and learn a new prediction model.

Solution:

This is bad practise. Increasing B can help with some of the reduce in variance and can improve the stability of the prediction model. However, it may not directly address the issue of bias.

It would be more effective to investigate the sources of bias in the model and address them. It would make more sense to look into other concerns like feature engineering, data quality, etc., that are more fundamental in addressing bias.

They could also look at adjusting other hyperparameters, trying newer models or collecting even more data.

(e):

We apply the random forest algorithm and notice that the output is highly biased. To mitigate the bias, we increase the number of predictors that are considered at every split when learning the decision trees.

Solution:

This is bad practise. Increasing the number of predictors considered at every split when learning decision trees in a random forest algorithm to mitigate bias can lead to more issues without directly addressing the issue of bias.

Increasing the number of predictors at each split could increase the similarity in the sub-trees of the model. This can lead to similar results in the trees within the ensemble. Also, it would be more computationally expensive as each of the decision trees constituting the random forest consider more predictors

It would be more effective to investigate the sources of bias in the model and address them. It would make more sense to look into other concerns like feature engineering, data quality, etc., that are more fundamental in addressing bias.

They could also look at adjusting other hyperparameters, trying newer models or collecting even more data.

(f):

We have a prediction task at hand and we know based on domain expertise that a good prediction model would not be axis aligned with the predictors. So instead of using random forests off the shelf, we first perform PCA to get a set of transformed features and then apply random forests.

Solution:

If we are not too concerned about the interpretability, this can be considered good practise.

Considering we know that a good model would not be axis aligned with the predictors, it would make sense to perform PCA to identify the principal components. This would lead to decorrelated features, and a smaller number of features.

If the number of features is high, using PCA could decrease the number of features dramatically. This is especially useful considering that Random Forests don't perform well with higher dimensional data. In addition, it would decrease the computational costs with creating a random forest model.

Question 1

In this question, you need to first generate 100 observations related to house price described in (a) and then use tree-based method for classification.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
np.random.seed(1)
```

In [3]:
```
# Data generation
n = 100

size = np.random.normal(loc=1500, scale=300, size=n)
bedrooms = np.random.choice([1, 2, 3, 4, 5], size=n)
age = np.random.normal(loc=20, scale=5, size=n)
renovation = np.random.binomial(1, 0.3, size=n)
noise_level = np.random.choice([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], size=n)


price = size * 100 + bedrooms * 5000 - age * 200 + renovation * 10000 + noise_level * 500

y = np.random.binomial(1, 1 / (1 + np.exp(166750 - price)), size=n)


df = pd.DataFrame({'size': size, 'bedrooms': bedrooms, 'age': age, 'renovation': renovation,
                     'noise_level': noise_level, 'y': y})
```

```
C:\Users\arjun\AppData\Local\Temp\ipykernel_11152\3495582695.py:13: RuntimeWarning: overflow encountered in exp
  y = np.random.binomial(1, 1 / (1 + np.exp(166750 - price)), size=n)
```

In [4]:
```
df.head()
```

Out[4]:

	size	bedrooms	age	renovation	noise_level	y
0	1987.303609	2	15.832214	0	6	1
1	1316.473076	3	18.103206	0	5	0
2	1341.548474	5	17.281888	0	4	0
3	1178.109413	4	25.418088	0	1	0
4	1759.622289	1	20.609030	1	7	1

Decision Tree

(b) Decision trees with cost complexity pruning. Please use cross validation to determine how much to prune. Please report the confusion matrix on both the training data and test data.

In [5]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import confusion_matrix
```

In [6]:
```
X = df.drop('y', axis=1)
y = df['y']
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [8]:
```
clf = DecisionTreeClassifier()

# Perform cross-validation to determine the optimal pruning parameter
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas[:-1]  # Exclude the maximum alpha
scores = []
for alpha in ccp_alphas:
    clf.set_params(ccp_alpha=alpha)
    cv_score = cross_val_score(clf, X_train, y_train, cv=5)
    scores.append(cv_score.mean())
```

In [9]:
```
# Find the optimal pruning parameter
optimal_alpha = ccp_alphas[np.argmax(scores)]
```

In [10]:
```
# Train the decision tree classifier with the optimal pruning parameter
clf.set_params(ccp_alpha=optimal_alpha)
clf.fit(X_train, y_train)
```

Out[10]:
```
DecisionTreeClassifier(ccp_alpha=0.019375000000000017)
```

In [11]:
```
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)
```

In [12]:
```
# Compute the confusion matrix for training and test data
train_confusion_matrix = confusion_matrix(y_train, y_train_pred)
test_confusion_matrix = confusion_matrix(y_test, y_test_pred)
```

In [13]:
```
TP_train = train_confusion_matrix[0][0]
FP_train = train_confusion_matrix[0][1]
FN_train = train_confusion_matrix[1][0]
TN_train = train_confusion_matrix[1][1]
```

In [14]:
```
print("Confusion Matrix for Training Data: \n", train_confusion_matrix)
```

```
Confusion Matrix for Training Data:
 [[37  3]
 [ 0 40]]
```

In [15]:
```
print("Confusion Matrix fpr Test Data: \n", test_confusion_matrix)
```

```
Confusion Matrix fpr Test Data:
 [[ 9  1]
 [ 0 10]]
```

In [16]:
```
print(clf.get_params())
```

```
{'ccp_alpha': 0.019375000000000017, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_lea
f_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'r
andom_state': None, 'splitter': 'best'}
```

RandomForest Classifier

(c) Random forests. You can use cross-validation to choose hyper-parameters. Please select the range of hyper-parameters by your own. Please report the confusion matrix on both the training data and test data.

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [18]:
```
### Hyperparameter tuning
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(start = 5, stop = 30, num = 6)]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]
```

In [19]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [20]:
```
# Creating a random grid

random_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'max_depth': max_depth
}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'min_samples
_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10], 'max_depth': [5, 10, 15, 20, 25, 30]}
```

In [21]:
```
rf = RandomForestClassifier()
```

In [22]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
                               scoring = 'accuracy', n_iter = 20, cv = 5, verbose = 2, n_jobs = 1)
```

In [23]:
```
rf_random.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=10, min_samples_split=100, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=1100; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=1100; total time=   0.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=300; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=300; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=300; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=300; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=300; total time=   0.1s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time=   0.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=400; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=400; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=400; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=400; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=400; total time=   0.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=900; total time=   0.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=900; total time=   0.5s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=100; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.2s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   0.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=15, n_estimators=1200; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=15, n_estimators=1200; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=15, n_estimators=1200; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=15, n_estimators=1200; total time=   0.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=2, min_samples_split=15, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=1100; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=1100; total time=   0.6s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=100; total time=   0.0s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=900; total time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=900; total time=   0.5s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=900; total time=   0.5s
```

```
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=900; total time=   0.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=900; total time=   0.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=900; total time=   0.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=900; total time=   0.5s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=100, n_estimators=700; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=100, n_estimators=700; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=100, n_estimators=700; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=100, n_estimators=700; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=2, min_samples_split=100, n_estimators=700; total time=   0.4s
```

Out[23]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   scoring='accuracy', verbose=2)
```

In [24]:

```
rf_y_train_pred = rf_random.predict(X_train)
```

In [25]:

```
rf_y_test_pred = rf_random.predict(X_test)
```

In [26]:

```
rf_train_confusion_matrix = confusion_matrix(y_train, rf_y_train_pred)
rf_test_confusion_matrix = confusion_matrix(y_test, rf_y_test_pred)

print("Confusion Matrix (Training Data): \n", rf_train_confusion_matrix)

print("Confusion Matrix (Test Data): \n", rf_test_confusion_matrix)
```

```
Confusion Matrix (Training Data): 
 [[40  0]
 [ 0 40]]
Confusion Matrix (Test Data): 
 [[10  0]
 [ 0 10]]
```

In [27]:

```
rf_random.get_params()
```

Out[27]:

```
{'cv': 5,
 'error_score': nan,
 'estimator__bootstrap': True,
 'estimator__ccp_alpha': 0.0,
 'estimator__class_weight': None,
 'estimator__criterion': 'gini',
 'estimator__max_depth': None,
 'estimator__max_features': 'auto',
 'estimator__max_leaf_nodes': None,
 'estimator__max_samples': None,
 'estimator__min_impurity_decrease': 0.0,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 2,
 'estimator__min_weight_fraction_leaf': 0.0,
 'estimator__n_estimators': 100,
 'estimator__n_jobs': None,
 'estimator__oob_score': False,
 'estimator__random_state': None,
 'estimator__verbose': 0,
 'estimator__warm_start': False,
 'estimator': RandomForestClassifier(),
 'n_iter': 20,
 'n_jobs': 1,
 'param_distributions': {'n_estimators': [100,
   200,
   300,
   400,
   500,
   600,
   700,
   800,
   900,
   1000,
   1100,
   1200],
  'max_features': ['auto', 'sqrt'],
  'min_samples_split': [2, 5, 10, 15, 100],
  'min_samples_leaf': [1, 2, 5, 10],
  'max_depth': [5, 10, 15, 20, 25, 30]},
 'pre_dispatch': '2*n_jobs',
 'random_state': None,
 'refit': True,
 'return_train_score': False,
 'scoring': 'accuracy',
 'verbose': 2}
```

AdaBoost Classifier

(d) Boosting. You can use cross-validation to choose hyper-parameters. Please select the range of hyper-parameters by your own. Please report the confusion matrix on both the training data and test data.

In [28]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [29]:
```
ada_clf = AdaBoostClassifier()
```

In [30]:
```
# Adaboost Hyperparameter Tuning
n_estimators_ada = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
learning_rate_ada = [x for x in 10.**np.arange(-5, 1)]
```

In [31]:
```
param_grid = {
    'n_estimators': n_estimators_ada,
    'learning_rate': learning_rate_ada,
}

print(param_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'learning_rate': [1e-05, 0.0001, 0.001, 0.01,
0.1, 1.0]}
```

In [32]:
```
ada_random = RandomizedSearchCV(estimator = ada_clf, param_distributions = param_grid,
                                scoring = 'accuracy', n_iter = 20, cv = 5, verbose = 2, n_jobs = 1)
```

In [33]:
```
ada_random.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END ................learning_rate=1.0, n_estimators=900; total time=   0.5s
[CV] END ................learning_rate=1.0, n_estimators=900; total time=   0.5s
[CV] END ................learning_rate=1.0, n_estimators=900; total time=   0.5s
[CV] END ................learning_rate=1.0, n_estimators=900; total time=   0.5s
[CV] END ................learning_rate=1.0, n_estimators=900; total time=   0.5s
[CV] END ............learning_rate=0.0001, n_estimators=1000; total time=   0.7s
[CV] END ............learning_rate=0.0001, n_estimators=1000; total time=   0.6s
[CV] END ............learning_rate=0.0001, n_estimators=1000; total time=   0.6s
[CV] END ............learning_rate=0.0001, n_estimators=1000; total time=   0.6s
[CV] END ............learning_rate=0.0001, n_estimators=1000; total time=   0.6s
[CV] END ...............learning_rate=0.1, n_estimators=1000; total time=   0.6s
[CV] END ...............learning_rate=0.1, n_estimators=1000; total time=   0.6s
[CV] END ...............learning_rate=0.1, n_estimators=1000; total time=   0.6s
[CV] END ...............learning_rate=0.1, n_estimators=1000; total time=   0.6s
[CV] END ...............learning_rate=0.1, n_estimators=1000; total time=   0.6s
[CV] END ................learning_rate=0.1, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=0.1, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=0.1, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=0.1, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=0.1, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.001, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.001, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.001, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.001, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.001, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=600; total time=   0.3s
[CV] END ...............learning_rate=1.0, n_estimators=1100; total time=   0.6s
[CV] END ...............learning_rate=1.0, n_estimators=1100; total time=   0.6s
[CV] END ...............learning_rate=1.0, n_estimators=1100; total time=   0.6s
[CV] END ...............learning_rate=1.0, n_estimators=1100; total time=   0.6s
[CV] END ...............learning_rate=1.0, n_estimators=1100; total time=   0.6s
[CV] END .............learning_rate=0.0001, n_estimators=700; total time=   0.4s
[CV] END .............learning_rate=0.0001, n_estimators=700; total time=   0.4s
[CV] END .............learning_rate=0.0001, n_estimators=700; total time=   0.4s
[CV] END .............learning_rate=0.0001, n_estimators=700; total time=   0.4s
[CV] END .............learning_rate=0.0001, n_estimators=700; total time=   0.4s
[CV] END ................learning_rate=1.0, n_estimators=300; total time=   0.1s
[CV] END ................learning_rate=1.0, n_estimators=300; total time=   0.1s
[CV] END ................learning_rate=1.0, n_estimators=300; total time=   0.1s
[CV] END ................learning_rate=1.0, n_estimators=300; total time=   0.1s
[CV] END ................learning_rate=1.0, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.0001, n_estimators=400; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=400; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=400; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=400; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=400; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.0001, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.0001, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.0001, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.0001, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=0.001, n_estimators=1100; total time=   0.6s
[CV] END .............learning_rate=0.001, n_estimators=1100; total time=   0.6s
[CV] END .............learning_rate=0.001, n_estimators=1100; total time=   0.6s
[CV] END .............learning_rate=0.001, n_estimators=1100; total time=   0.6s
[CV] END .............learning_rate=0.001, n_estimators=1100; total time=   0.6s
[CV] END ...............learning_rate=0.01, n_estimators=300; total time=   0.1s
[CV] END ...............learning_rate=0.01, n_estimators=300; total time=   0.1s
[CV] END ...............learning_rate=0.01, n_estimators=300; total time=   0.1s
[CV] END ...............learning_rate=0.01, n_estimators=300; total time=   0.1s
[CV] END ...............learning_rate=0.01, n_estimators=300; total time=   0.1s
[CV] END .............learning_rate=1e-05, n_estimators=1200; total time=   0.7s
[CV] END .............learning_rate=1e-05, n_estimators=1200; total time=   0.7s
[CV] END .............learning_rate=1e-05, n_estimators=1200; total time=   0.7s
[CV] END .............learning_rate=1e-05, n_estimators=1200; total time=   0.7s
[CV] END .............learning_rate=1e-05, n_estimators=1200; total time=   0.7s
[CV] END ................learning_rate=1.0, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=1.0, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=1.0, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=1.0, n_estimators=600; total time=   0.3s
[CV] END ................learning_rate=1.0, n_estimators=600; total time=   0.3s
[CV] END ..............learning_rate=0.001, n_estimators=900; total time=   0.5s
[CV] END ..............learning_rate=0.001, n_estimators=900; total time=   0.5s
[CV] END ..............learning_rate=0.001, n_estimators=900; total time=   0.5s
[CV] END ..............learning_rate=0.001, n_estimators=900; total time=   0.5s
[CV] END ..............learning_rate=0.001, n_estimators=900; total time=   0.5s
[CV] END .............learning_rate=0.0001, n_estimators=500; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=500; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=500; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=500; total time=   0.2s
[CV] END .............learning_rate=0.0001, n_estimators=500; total time=   0.2s
[CV] END ...............learning_rate=0.1, n_estimators=1200; total time=   0.7s
[CV] END ...............learning_rate=0.1, n_estimators=1200; total time=   0.7s
[CV] END ...............learning_rate=0.1, n_estimators=1200; total time=   0.7s
[CV] END ...............learning_rate=0.1, n_estimators=1200; total time=   0.7s
[CV] END ...............learning_rate=0.1, n_estimators=1200; total time=   0.7s
[CV] END ................learning_rate=0.1, n_estimators=700; total time=   0.4s
```

```
[CV] END ...............learning_rate=0.1, n_estimators=700; total time=   0.4s
[CV] END ...............learning_rate=0.1, n_estimators=700; total time=   0.4s
[CV] END ...............learning_rate=0.1, n_estimators=700; total time=   0.4s
[CV] END ...............learning_rate=0.1, n_estimators=700; total time=   0.4s
[CV] END ..............learning_rate=0.1, n_estimators=1100; total time=   0.6s
[CV] END ..............learning_rate=0.1, n_estimators=1100; total time=   0.6s
[CV] END ..............learning_rate=0.1, n_estimators=1100; total time=   0.6s
[CV] END ..............learning_rate=0.1, n_estimators=1100; total time=   0.6s
[CV] END ..............learning_rate=0.1, n_estimators=1100; total time=   0.6s
```

Out[33]:

```
RandomizedSearchCV(cv=5, estimator=AdaBoostClassifier(), n_iter=20, n_jobs=1,
                   param_distributions={'learning_rate': [1e-05, 0.0001, 0.001,
                                                          0.01, 0.1, 1.0],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   scoring='accuracy', verbose=2)
```

In [34]:

```
ada_y_train_pred = ada_random.predict(X_train)
ada_y_test_pred = ada_random.predict(X_test)

train_confusion_matrix_ada = confusion_matrix(y_train, ada_y_train_pred)
test_confusion_matrix_ada = confusion_matrix(y_test, ada_y_test_pred)
```

In [35]:

```
print("Confusion Matrix (Training Data):\n", train_confusion_matrix)
```

```
Confusion Matrix (Training Data):
 [[37  3]
 [ 0 40]]
```

In [36]:

```
print("Confusion Matrix (Test Data):\n", test_confusion_matrix)
```

```
Confusion Matrix (Test Data):
 [[ 9  1]
 [ 0 10]]
```

In [37]:

```
ada_random.get_params()
```

Out[37]:

```
{'cv': 5,
 'error_score': nan,
 'estimator__algorithm': 'SAMME.R',
 'estimator__base_estimator': None,
 'estimator__learning_rate': 1.0,
 'estimator__n_estimators': 50,
 'estimator__random_state': None,
 'estimator': AdaBoostClassifier(),
 'n_iter': 20,
 'n_jobs': 1,
 'param_distributions': {'n_estimators': [100,
   200,
   300,
   400,
   500,
   600,
   700,
   800,
   900,
   1000,
   1100,
   1200],
  'learning_rate': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1.0]},
 'pre_dispatch': '2*n_jobs',
 'random_state': None,
 'refit': True,
 'return_train_score': False,
 'scoring': 'accuracy',
 'verbose': 2}
```

Question 2

This problem will make use of the Carseats dataset in the ISLR and ISLP packages. In comparison to the lab, where we treated the Sales variable as a binary response with two levels, this problem will be focused on using tree-based methods to predict the Sales variable as a quantitative response, using all other variables in the dataset as predictors. For this problem, you are free to use built-in packages in either R or Python of your choice.

In [38]:

```
df = pd.read_csv('C:/Users/arjun/Downloads/Carseats.csv')
```

In [39]:

```
df.head()
```

Out[39]:

	Unnamed: 0	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	1	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
1	2	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
2	3	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
3	4	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4	5	4.15	141	64	3	340	128	Bad	38	13	Yes	No

In [40]:
```
df.drop('Unnamed: 0', axis = 1, inplace = True)
```

In [41]:
```
df.describe()
```

Out[41]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Education
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	7.496325	124.975000	68.657500	6.635000	264.840000	115.795000	53.322500	13.900000
std	2.824115	15.334512	27.986037	6.650364	147.376436	23.676664	16.200297	2.620528
min	0.000000	77.000000	21.000000	0.000000	10.000000	24.000000	25.000000	10.000000
25%	5.390000	115.000000	42.750000	0.000000	139.000000	100.000000	39.750000	12.000000
50%	7.490000	125.000000	69.000000	5.000000	272.000000	117.000000	54.500000	14.000000
75%	9.320000	135.000000	91.000000	12.000000	398.500000	131.000000	66.000000	16.000000
max	16.270000	175.000000	120.000000	29.000000	509.000000	191.000000	80.000000	18.000000

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64  
 2   Income       400 non-null    int64  
 3   Advertising  400 non-null    int64  
 4   Population   400 non-null    int64  
 5   Price        400 non-null    int64  
 6   ShelveLoc    400 non-null    object 
 7   Age          400 non-null    int64  
 8   Education    400 non-null    int64  
 9   Urban        400 non-null    object 
 10  US           400 non-null    object 
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [43]:
```
df['ShelveLoc'].unique()
```

Out[43]:
```
array(['Bad', 'Good', 'Medium'], dtype=object)
```

In [44]:
```
df['Urban'].unique()
```

Out[44]:
```
array(['Yes', 'No'], dtype=object)
```

In [45]:
```
df['US'].unique()
```

Out[45]:
```
array(['Yes', 'No'], dtype=object)
```

In [46]:
```
import numpy as np
```

In [47]:
```
df['ShelveLoc'] = np.where(df['ShelveLoc'] == 'Bad', 1, df['ShelveLoc'])
df['ShelveLoc'] = np.where(df['ShelveLoc'] == 'Medium', 2, df['ShelveLoc'])
df['ShelveLoc'] = np.where(df['ShelveLoc'] == 'Good', 3, df['ShelveLoc'])
```

In [48]:
```
df['Urban'] = np.where(df['Urban'] == 'Yes', 1, 0)
df['US'] = np.where(df['US'] == 'Yes', 1, 0)
```

In [49]:
```
df = df.apply(pd.to_numeric)
```

In [50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64  
 2   Income       400 non-null    int64  
 3   Advertising  400 non-null    int64  
 4   Population   400 non-null    int64  
 5   Price        400 non-null    int64  
 6   ShelveLoc    400 non-null    int64  
 7   Age          400 non-null    int64  
 8   Education    400 non-null    int64  
 9   Urban        400 non-null    int32  
 10  US           400 non-null    int32  
dtypes: float64(1), int32(2), int64(8)
memory usage: 31.4 KB
```

In [51]:
```
df['US'].value_counts()
```

Out[51]:
```
1    258
0    142
Name: US, dtype: int64
```

In [52]:
```
df['ShelveLoc'].value_counts()
```

Out[52]:
```
2    219
1     96
3     85
Name: ShelveLoc, dtype: int64
```

In [53]:
```
df['Urban'].value_counts()
```

Out[53]:
```
1    282
0    118
Name: Urban, dtype: int64
```

In [54]:
```
X = df.drop('Sales', axis=1)
y = df['Sales']
```

Decision Tree (Full and Pruned Trees)

(a) Split the data into 70% training and 30% test observations. Fit a regression tree to predict the Sales variable as a quantitative response, using all other variables in the dataset as predictors, and perform cross-validation to determine the optimal level of complexity. (Note: You will not be setting a max depth argument here since it will be learned during cross- validation). Report the test MSE, as well as number of terminal nodes, for 1) the full tree and 2) pruned tree of optimal CV complexity.

In [55]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

In [56]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [57]:
```
# Initialize the full tree
full_tree = DecisionTreeRegressor()
full_tree.fit(X_train, y_train)
```

Out[57]:
```
DecisionTreeRegressor()
```

In [58]:
```
y_pred_full_tree = full_tree.predict(X_test)
test_mse_full_tree = mean_squared_error(y_test, y_pred_full_tree)
print("Test MSE of Full Tree:", test_mse_full_tree)
```

```
Test MSE of Full Tree: 5.138118333333334
```

In [59]:
```
# Get the number of terminal nodes for the full tree
num_terminal_nodes_full_tree = full_tree.get_n_leaves()
print("Number of Terminal Nodes of Full Tree:", num_terminal_nodes_full_tree)
```

```
Number of Terminal Nodes of Full Tree: 275
```

In [60]:
```
print("Parameters of the full tree: ", full_tree.get_params())
```

```
Parameters of the full tree:  {'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth': None, 'max_features': None, 'max_lea
f_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'r
andom_state': None, 'splitter': 'best'}
```

In [61]:
```
param_grid = {'max_depth': [int(x) for x in np.linspace(1, 16)]}

# Create the decision tree regressor
tree = DecisionTreeRegressor()
```

In [62]:
```
pruned_tree = GridSearchCV(estimator = tree, param_grid = param_grid, cv=5, scoring='neg_mean_squared_error')
pruned_tree.fit(X_train, y_train)
```

Out[62]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4,
                                       5, 5, 5, 6, 6, 6, 7, 7, 8, 8, 8, 8, 9,
                                       9, 9, ...]},
             scoring='neg_mean_squared_error')
```

In [63]:
```
best_depth = pruned_tree.best_params_['max_depth']
best_estimator = pruned_tree.best_estimator_
```

In [64]:
```
print('Best Tree Depth:', best_depth)
print('Best Estimator:', best_estimator)
```

```
Best Tree Depth: 4
Best Estimator: DecisionTreeRegressor(max_depth=4)
```

In [65]:
```
y_pred_pruned_tree = pruned_tree.predict(X_test)
test_mse_pruned_tree = mean_squared_error(y_test, y_pred_pruned_tree)
print("Test MSE of Pruned Tree:", test_mse_pruned_tree)
```

```
Test MSE of Pruned Tree: 5.694151778300362
```

In [66]:
```
num_terminal_nodes_pruned_tree = best_estimator.get_n_leaves()
print("Number of Terminal Nodes of Pruned Tree:", num_terminal_nodes_pruned_tree)
```

```
Number of Terminal Nodes of Pruned Tree: 16
```

In [67]:
```
print("Parameters of the pruned tree: ", best_estimator.get_params())
```

```
Parameters of the pruned tree:  {'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth': 4, 'max_features': None, 'max_leaf
_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'ra
ndom_state': None, 'splitter': 'best'}
```

Bagging

(b) Use bagging to predict the Sales variable as a quantitative response, using all other variables in the dataset as predictors. Report the resulting test MSE as well as the relative importance of each predictor.

In [68]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
```

In [69]:
```
### Hyperparameter tuning
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(start = 5, stop = 30, num = 6)]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]
```

In [70]:
```
# Creating a random grid

param_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'max_depth': max_depth
}

print(param_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'min_samples
_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10], 'max_depth': [5, 10, 15, 20, 25, 30]}
```

In [71]:
```
rf = RandomForestRegressor()
```

In [72]:
```
rf_grid = RandomizedSearchCV(estimator = rf, param_distributions = param_grid,
                             scoring = 'neg_mean_squared_error', n_iter = 20, cv = 5, verbose = 2, n_jobs = 1)
```

In [73]:
```
rf_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=900; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=900; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=900; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=900; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=900; total time=   0.6s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=1000; total time=   0.7s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=1000; total time=   0.7s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=1000; total time=   0.7s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=1000; total time=   0.7s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=1000; total time=   0.8s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=100; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=100; total time=   0.0s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=1000; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=1000; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=1000; total time=   0.7s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=1000; total time=   0.6s
[CV] END max_depth=25, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=1000; total time=   0.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.1s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.1s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5, min_samples_split=2, n_estimators=200; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=400; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=400; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=400; total time=   0.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=400; total time=   0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=2, n_estimators=400; total time=   0.1s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1100; total time=   0.7s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=600; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=600; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=600; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=600; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=10, n_estimators=600; total time=   0.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=1100; total time=   0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=100, n_estimators=1100; total time=   0.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   0.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=1000; total time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=1000; total time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=1000; total time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=1000; total time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=10, min_samples_split=5, n_estimators=1000; total time=   0.5s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   0.4s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   0.3s
[CV] END max_depth=5, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=600; total time=   0.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=500; total time=   0.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=500; total time=   0.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=500; total time=   0.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=500; total time=   0.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=500; total time=   0.3s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=100, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=100, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=100, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=100, n_estimators=100; total time=   0.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=1, min_samples_split=100, n_estimators=100; total time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=500; total time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=500; total time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=500; total time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=500; total time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=2, n_estimators=500; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=600; total time=   0.3s
```

```
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=600; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=600; total time=   0.3s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=600; total time=   0.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5, min_samples_split=10, n_estimators=1200; total time=   0.6s
```

Out[73]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=20, n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   scoring='neg_mean_squared_error', verbose=2)
```

In [74]:

```
rf_y_train_pred = rf_grid.predict(X_train)
rf_y_test_pred = rf_grid.predict(X_test)
```

In [75]:

```
best_rf = best_estimator = rf_grid.best_estimator_
```

In [76]:

```
train_mse_rf_grid = mean_squared_error(y_train, rf_y_train_pred)
test_mse_rf_grid = mean_squared_error(y_test, rf_y_test_pred)
```

In [77]:

```
print("Mean Squared Error: ", test_mse_rf_grid)
```

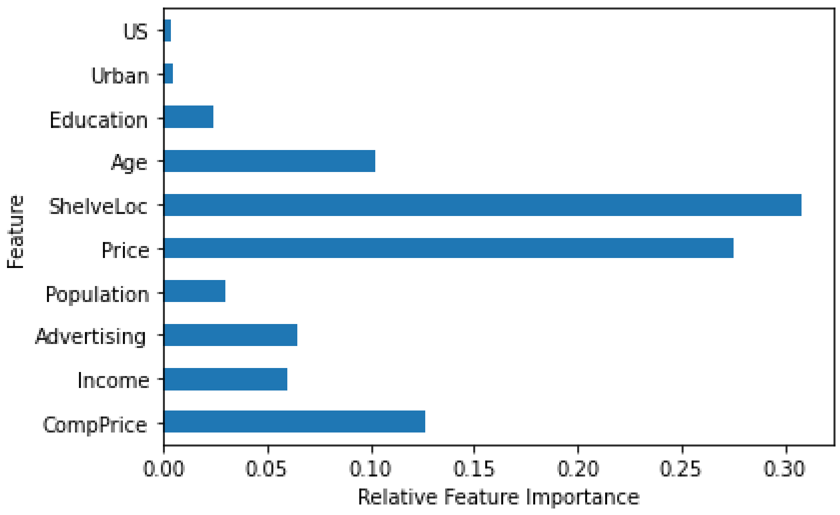
```
Mean Squared Error:  2.7967355640358105
```

In [78]:

```
import matplotlib.pyplot as plt
```

In [79]:

```
# Feature Importances for relative importance
feat_imp = pd.Series(best_rf.feature_importances_, index = X.columns)
feat_imp.plot(kind = 'barh')
plt.xlabel('Relative Feature Importance')
plt.ylabel('Feature')
plt.show()
```



In [80]:

```
print(best_rf.feature_importances_)
```

```
[0.12626199 0.06031201 0.06444804 0.0299271  0.2757365  0.30812857
 0.10218633 0.02400587 0.00499806 0.00399553]
```

Random Forest with 1 to p features:

(c) Let p be the number of predictors, i.e., one fewer than the total number of variables/columns in the Carseats dataset. For m = 1, . . . , p, fit a random forest model to predict the Sales variable as a quantitative response, using all other variables in the dataset as predictors. Plot the resulting test MSE (on the y-axis) as a function of m (on the x-axis, ranging from 1 to p). Using m = $\sqrt{p}$ (rounded to the nearest integer if necessary), report the test MSE and relative importance of each predictor

In [81]:

```
p = X.shape[1]

m = int(np.sqrt(p))

m_values = []
mse_values = []
importance_values = []
```

In [82]:

```
for num_predictors in range(1, p+1):
    # Select num_predictors predictors
    selected_predictors = X.columns[:num_predictors]
    X_subset = X_train[selected_predictors]
    X_test_sub = X_test[selected_predictors]
    # Create the random forest regressor
    rf = RandomForestRegressor()
```

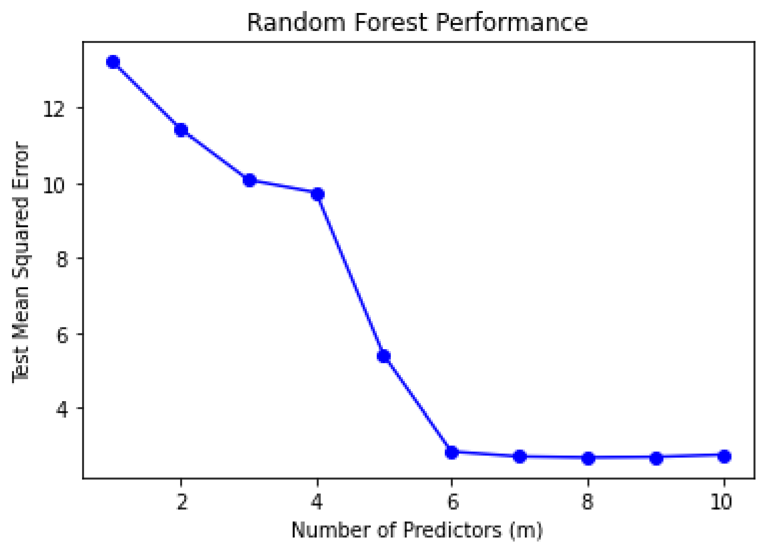
```
    rf.fit(X_subset, y_train)

    y_pred = rf.predict(X_subset)
    y_test_pred = rf.predict(X_test_sub)
    mse = mean_squared_error(y_test_pred, y_test)

    importance = rf.feature_importances_

    m_values.append(num_predictors)
    mse_values.append(mse)
    importance_values.append(importance)
```

In [83]:
```
plt.plot(m_values, mse_values, color = 'blue', marker='o')
plt.xlabel('Number of Predictors (m)')
plt.ylabel('Test Mean Squared Error')
plt.title('Random Forest Performance')
plt.show()
```



In [84]:
```
m_index = int(np.sqrt(p)) - 1

print("Test MSE for m = sqrt(p):", mse_values[m_index])
print("Relative importance of each predictor for m = sqrt(p):")
for i, predictor in enumerate(X.columns[:m]):
    print(predictor, ":", importance_values[m_index][i])
```

```
Test MSE for m = sqrt(p): 10.088313145333329
Relative importance of each predictor for m = sqrt(p):
CompPrice : 0.39515085446937454
Income : 0.387585921066421
Advertising : 0.21726322446420454
```

LASSO on The Whole Dataset

(d) Using the full dataset, i.e., not just the 70% split for training observations, fit a LASSO regression model to predict the Sales variable as a quantitative response, using all other variables in the dataset as predictors. Plot the coefficient path for whichever range of λ values is the default choice for the function you used. Discuss whether this plot makes sense in conjunction with the variable importance rankings from your bagging and random forest procedures.

In [85]:
```
from sklearn.linear_model import Lasso
```

In [124…
```
alphas = np.logspace(-4, 0, 50)
```

In [139…
```
# Fit the model on the training data
coefs = []
for i in range(len(alphas)):
    lasso = Lasso(alpha = alphas[i])
    l1_model = lasso.fit(X, y)
    coefs.append(l1_model.coef_)
```

In [140…
```
coef_df = pd.DataFrame(coefs, columns = X.columns)
```

In [141…
```
coef_df.head()
```

Out[141]:

	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	0.092552	0.016152	0.120321	0.000291	-0.095247	2.411321	-0.046860	-0.020928	0.140692	-0.128427
1	0.092552	0.016152	0.120311	0.000291	-0.095247	2.411269	-0.046860	-0.020923	0.140585	-0.128236
2	0.092552	0.016152	0.120300	0.000291	-0.095247	2.411206	-0.046859	-0.020918	0.140456	-0.128006
3	0.092552	0.016151	0.120286	0.000291	-0.095247	2.411130	-0.046859	-0.020912	0.140301	-0.127729
4	0.092552	0.016151	0.120269	0.000291	-0.095246	2.411038	-0.046859	-0.020905	0.140113	-0.127393

In [142…
```
coef_df['lambda_value'] = alphas
```

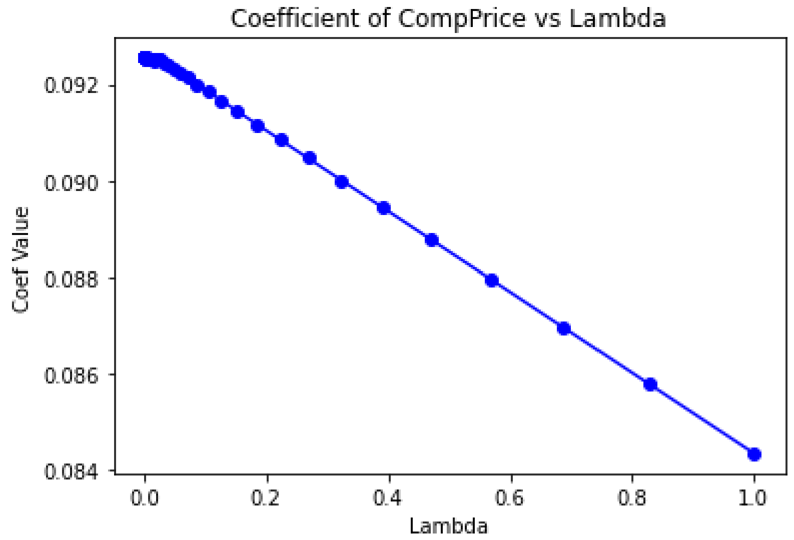
In [143…
```
coef_df.shape
```

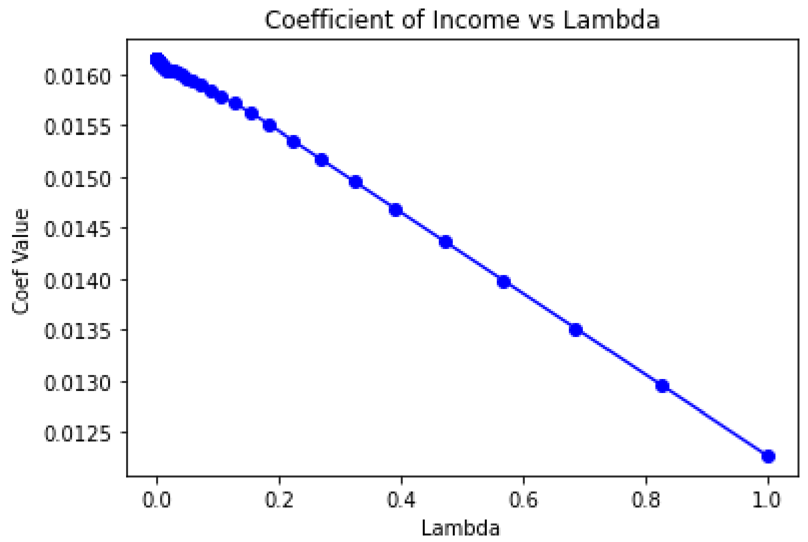
Out[143]: (50, 11)

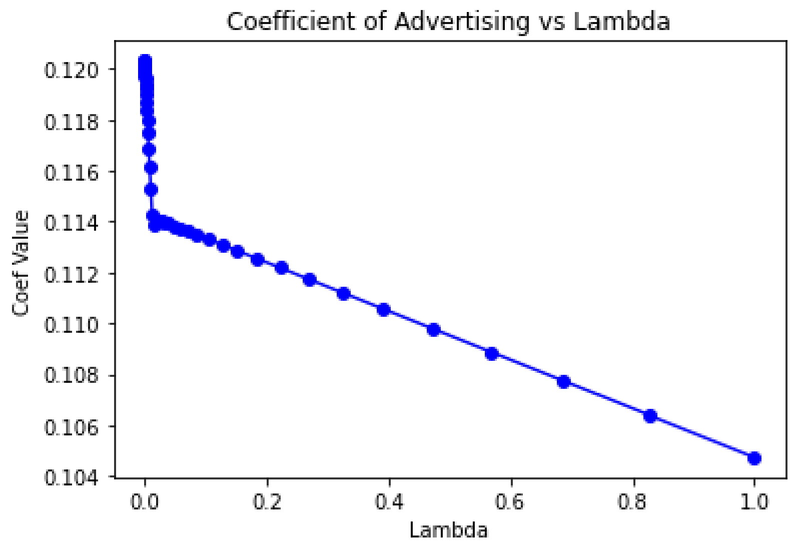
In [144…
```
coef_df.info()
```

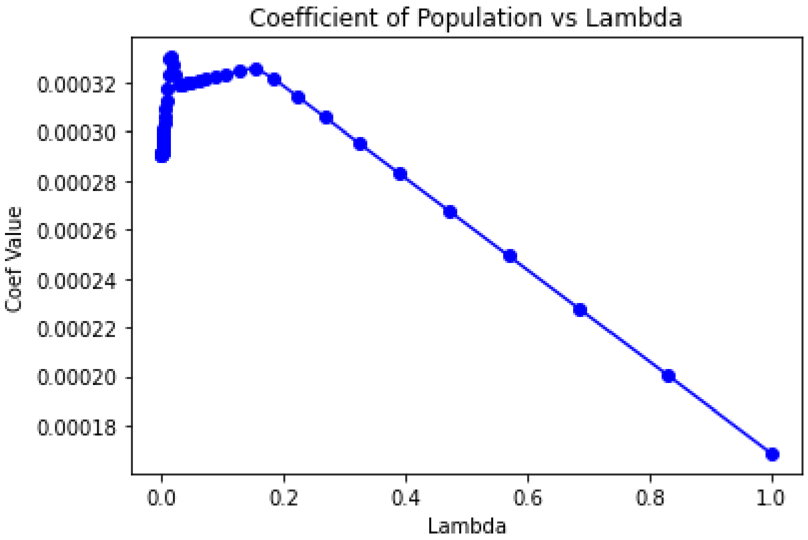
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   CompPrice     50 non-null     float64
 1   Income        50 non-null     float64
 2   Advertising   50 non-null     float64
 3   Population    50 non-null     float64
 4   Price         50 non-null     float64
 5   ShelveLoc     50 non-null     float64
 6   Age           50 non-null     float64
 7   Education     50 non-null     float64
 8   Urban         50 non-null     float64
 9   US            50 non-null     float64
 10  lambda_value  50 non-null     float64
dtypes: float64(11)
memory usage: 4.4 KB
```

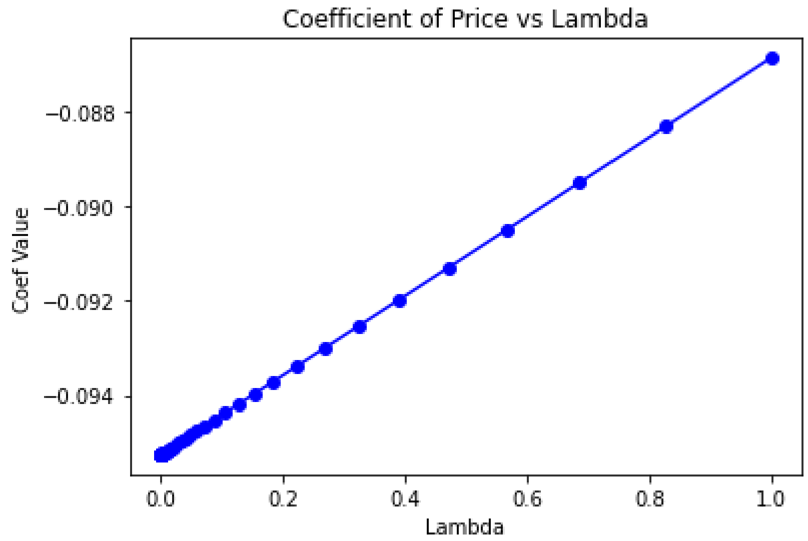
In [145…
```
for i in X.columns:
    col_vals = coef_df[i]
    plt.plot(alphas,col_vals,color = 'blue', marker = 'o')
    plt.figsize = (10,10)
    plt.title(f'Coefficient of {i} vs Lambda')
    plt.xlabel('Lambda')
    plt.ylabel('Coef Value')
    plt.show()
```

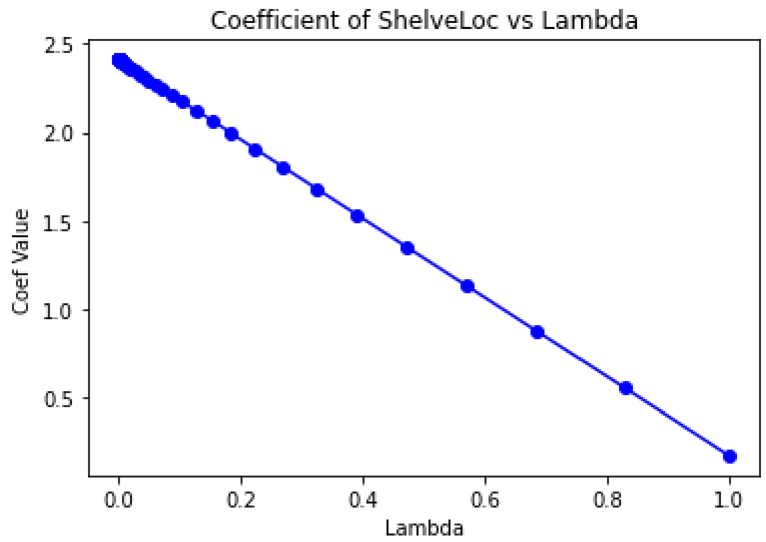


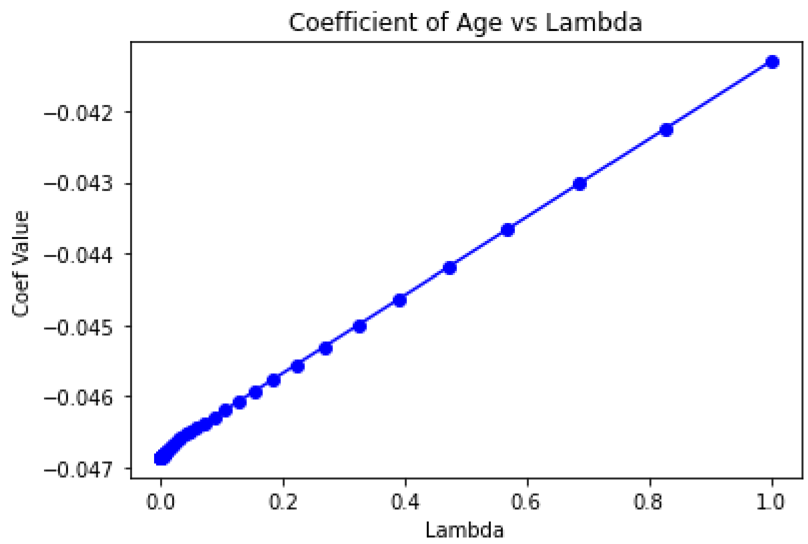


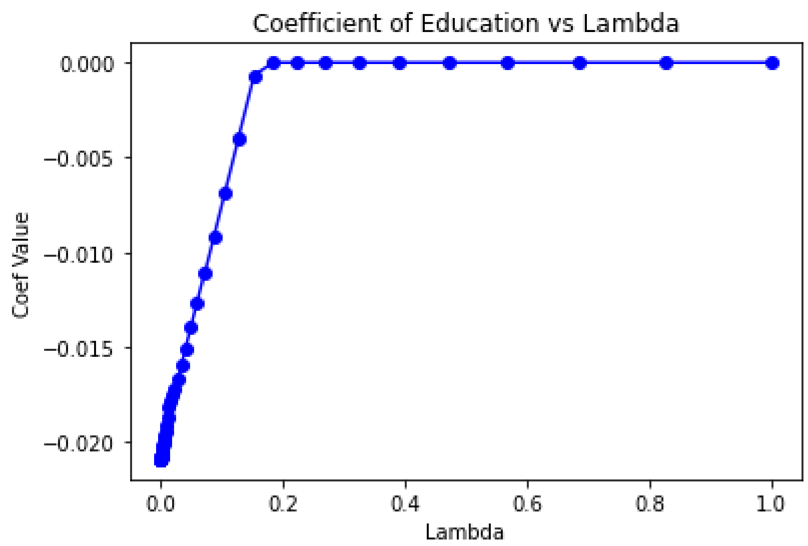


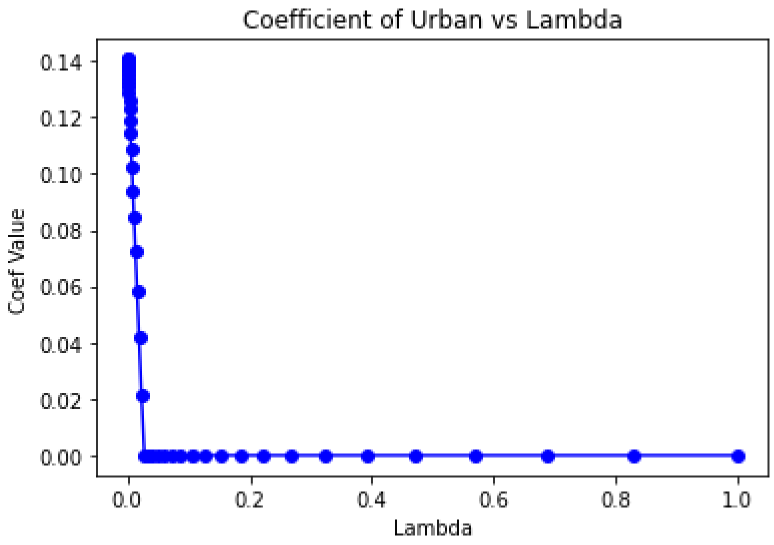

Coefficient of Population vs Lambda
Coef Value
0.00032
0.00030
0.00028
0.00026
0.00024
0.00022
0.00020
0.00018
0.0
0.2
0.4
0.6
0.8
1.0
Lambda

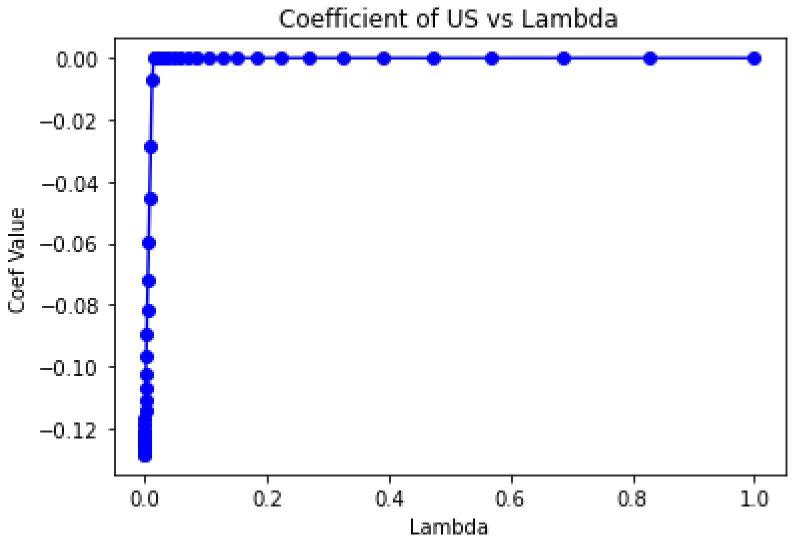

Coefficient of Price vs Lambda
Coef Value
−0.088
−0.090
−0.092
−0.094
0.0
0.2
0.4
0.6
0.8
1.0
Lambda


Coefficient of ShelveLoc vs Lambda
Coef Value
2.5
2.0
1.5
1.0
0.5
0.0
0.2
0.4
0.6
0.8
1.0
Lambda


Coefficient of Age vs Lambda
Coef Value
−0.042
−0.043
−0.044
−0.045
−0.046
−0.047
0.0
0.2
0.4
0.6
0.8
1.0
Lambda


Coefficient of Education vs Lambda
Coef Value
0.000
−0.005
−0.010
−0.015
−0.020
0.0
0.2
0.4
0.6
0.8
1.0
Lambda





The results of the LASSO model are mostly in agreement with the other models. With the exception of advertising, all other features are 1 or 2 positions away from that of the other model in terms of their positioning in the importance hierarchy.

The features in decreasing order of absolute values of coefficients (LASSO):

1. ShelveLoc
2. Advertising
3. Price
4. ComPrice
5. Age
6. Income
7. Population
8. Education
9. Urban
10. US

(8, 9, and 10 are interchangeable since they all have zero coefficients)

Features in decreasing order of relative importance (Random Forest):

1. ShelveLoc
2. Price
3. ComPrice
4. Age
5. Advertising
6. Income
7. Population
8. Education
9. Urban
10. US

The above result is for a custom range of lambdas to understand the coefficient trends on a smaller range.

The overall coefficient path can be visualized below

In [156…

```
from sklearn.preprocessing import StandardScaler
```

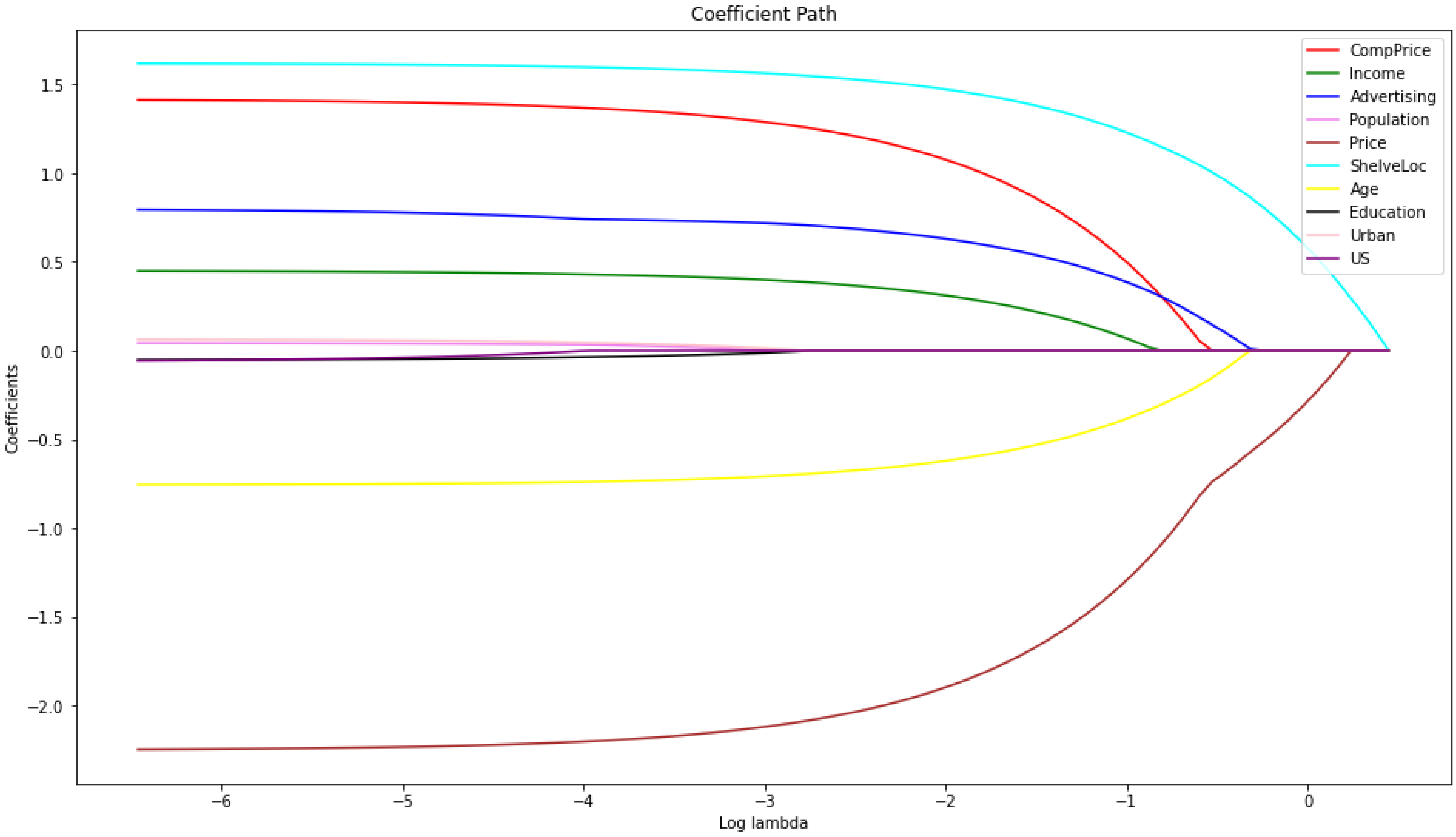
In [162…

```
sc = StandardScaler() # Scaling done to introduce uniformity in the coefficient path
X_sc = sc.fit_transform(X)
lasso_cv = LassoCV(cv=5)
lasso_cv.fit(X_sc, y)
alphas = lasso.alphas_
coefficient_path = lasso.path(X_sc, y)[1]
```

In [163…

```
plt.figure(figsize=(16, 9))
```

```
colors = ['red', 'green', 'blue', 'violet', 'brown', 'cyan', 'yellow', 'black', 'pink', 'purple']
for i,color in enumerate(colors):
    plt.plot(np.log(alphas), coefficient_path[i], color)
plt.xlabel("Log lambda")
plt.ylabel("Coefficients")
plt.title("Coefficient Path")
plt.legend(X.columns, loc="upper right")
plt.grid(False)
plt.show()
```



In the overall coefficient path, here is the decreasing order of importance:

1. ShelveLoc
2. Price
3. Advertising
4. Age
5. CompPrice
6. Income
7. Population
8. Urban
9. Education
10. US

Which, again, is mostly consistent with previous results