Use the datasets cars.csv. Based on the data, predict if a store has exceeded the target level of selling car seats for children. The dependent variable takes on value 1 if they have exceeded the target. The predictors can be found in Table 1.

1. Find the optimal depth of the tree using the accuracy score based on cross-validation between depth 1-25.

To start of with we standardized the data set by using the following formula:

```
sdf = (df - df.mean()) / df.std()
```

We then picked out Sales as our dependent variables and Comprice, Income, Advertising, Population, Price, Age, Education, Urban, US, as our independent variables from the dataset.

To find the optimal depth for the regression tree, we then used cross-validation with a depth range between 1 and 25. Cross-validation allows us to estimate the model's accuracy at each depth, helping us identify the depth that minimizes the error and provides the best fit for our dataset.

We then ran a training decision tree regressor for each depth and evaluated the model using 10-fold cross-validation. The cross_val_score will be used as the metric for measuring model performance since it's a classification problem.

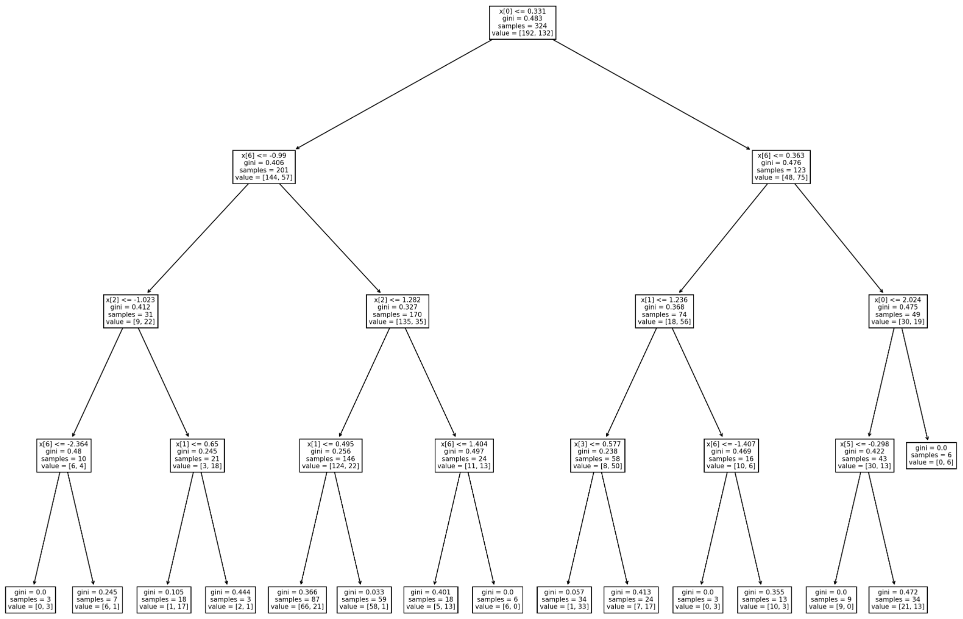
This was done utilizing the following code:

Code:

```
X= cars[cars.columns.difference(['Sales'])]
X= (X-X.mean())/X.std()
y = cars['Sales']

# Estimate the model
regcl = DecisionTreeClassifier(max_depth=3, random_state=0)
regcl = regcl.fit(X, y)

# Create a vector of max depth
degree = np.linspace(1,25,25)
```

```
degree1=degree.astype(int)
# Create help matrix:
Acc=[]

# Write a loop
for d in degree1:
    regcl = DecisionTreeClassifier(criterion='entropy',max_depth=d, random_state=0)
    regcl = regcl.fit(X, y)
    scoresd = cross_val_score(regcl, X, y,cv=10)
    sc= scoresd.mean()
    Acc.append(sc)

# Determine the optimal maximum depth
optimal_depth = degree1[Acc.index(max(Acc))]
print(f"The optimal maximum depth is: {optimal_depth}")

output:  The optimal maximum depth is: 4
```

2. Visualize the optimal tree and interpret it.



x[0] <= 0.331 gini = 0.483 samples = 324 value = [192, 132]
x[6] <= -0.99 gini = 0.406 samples = 201 value = [144, 57]
x[6] <= 0.363 gini = 0.476 samples = 123 value = [48, 75]
x[2] <= -1.023 gini = 0.412 samples = 31 value = [9, 22]
x[2] <= 1.282 gini = 0.327 samples = 170 value = [135, 35]
x[1] <= 1.236 gini = 0.368 samples = 74 value = [18, 56]
x[0] <= 2.024 gini = 0.475 samples = 49 value = [30, 19]
x[6] <= -2.364 gini = 0.48 samples = 10 value = [6, 4]
x[1] <= 0.65 gini = 0.245 samples = 21 value = [3, 18]
x[1] <= 0.495 gini = 0.256 samples = 146 value = [124, 22]
x[6] <= 1.404 gini = 0.497 samples = 24 value = [11, 13]
x[3] <= 0.577 gini = 0.238 samples = 58 value = [8, 50]
x[6] <= -1.407 gini = 0.469 samples = 16 value = [10, 6]
x[5] <= -0.298 gini = 0.422 samples = 43 value = [30, 13]
gini = 0.0 samples = 6 value = [0, 6]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.245 samples = 7 value = [6, 1]
gini = 0.105 samples = 18 value = [1, 17]
gini = 0.444 samples = 3 value = [2, 1]
gini = 0.366 samples = 87 value = [66, 21]
gini = 0.033 samples = 59 value = [58, 1]
gini = 0.401 samples = 18 value = [5, 13]
gini = 0.0 samples = 6 value = [6, 0]
gini = 0.057 samples = 34 value = [1, 33]
gini = 0.413 samples = 24 value = [7, 17]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.355 samples = 13 value = [10, 3]
gini = 0.0 samples = 9 value = [9, 0]
gini = 0.472 samples = 34 value = [21, 13]

```
# Visualize optimal classification tree
fig, ax = plt.subplots(figsize=(20, 15))
regcl = tree. DecisionTreeClassifier(max_depth= 4, random_state=0)
regcl = regcl.fit(X, y)
tree.plot_tree(regcl);
```

The visualization shows us how each decision is made. By providing us with information about what variable it uses, and which value is set as a threshold, as well as the squared error value.

The squared error represents the variance in the target variable (Sales) at each node before further splitting. Lower squared error values indicate that the model is making more accurate predictions at that specific decision point.

The maximum depth of 4 in the context of this dataset indicates that the decision tree model will make decisions based on up to four levels of splits in the feature space. Simply put, the model will go through 4 stages of decision making and that the 4th decision will determine the predicted value of sales(yhat mathematically). A maximum depth of 4 means that the tree can make decisions based on up to four splits. This depth allows the model to capture significant interactions between features while maintaining a level of simplicity that helps prevent overfitting. Hence, larger tree depths for this dataset will lead to overfitting.

Each node represents a decision based on a feature, and the split aims to minimize the squared error by dividing the samples into more homogenous groups. For example, if Advertising is less than or equal to 0.33, the samples are further split based on Price, leading to a more accurate prediction of Sales.

The values at the leaf nodes show the predicted sales value when a particular path is followed. Each path down the tree corresponds to specific conditions (e.g., low advertising and low price). These paths help us understand the relationship between the features and the target variable, making it easier to interpret how different factors like Advertising, Price, or Age impact Sales.

Overall, the decision tree helps us visually understand the segmentation of the dataset, showing how different ranges of each feature contribute to predictions. This makes it a powerful tool for identifying the most significant factors and understanding the rules that drive thesales, thus aiding in better decision-making and strategy formulation.

3. Estimate a random forest method based on the optimal depth found in question 1 and calculate the accuracy score using 10 fold cross-validation.

Using the optimal tree depth of 4 determined from the previous analysis, we implemented cross-validation using the cross_val_score which is generally used for classification problems for each of the 10 folds. This approach allowed us to assess the model’s performance across different subsets of the data. Finally, we averaged the scores from these 10 folds to obtain an overall estimate of the model's predictive accuracy as shown below:

Code:

```
rfcl = RandomForestClassifier(max_depth =4, random_state = 0)
rfcl=rfcl.fit(X,y)
# Cross-validation for rfcl
scoresrfcl = cross_val_score(rfcl, X, y,cv=10)
scoresrfcl.mean()

Accuracy Score is: 0.721875
```

4. Estimate support vector machines with polynomials of degree 2,3 and 4. Also, radial support vector machines. Calculate the accuracy score using 10 folded cross-validation of each method.

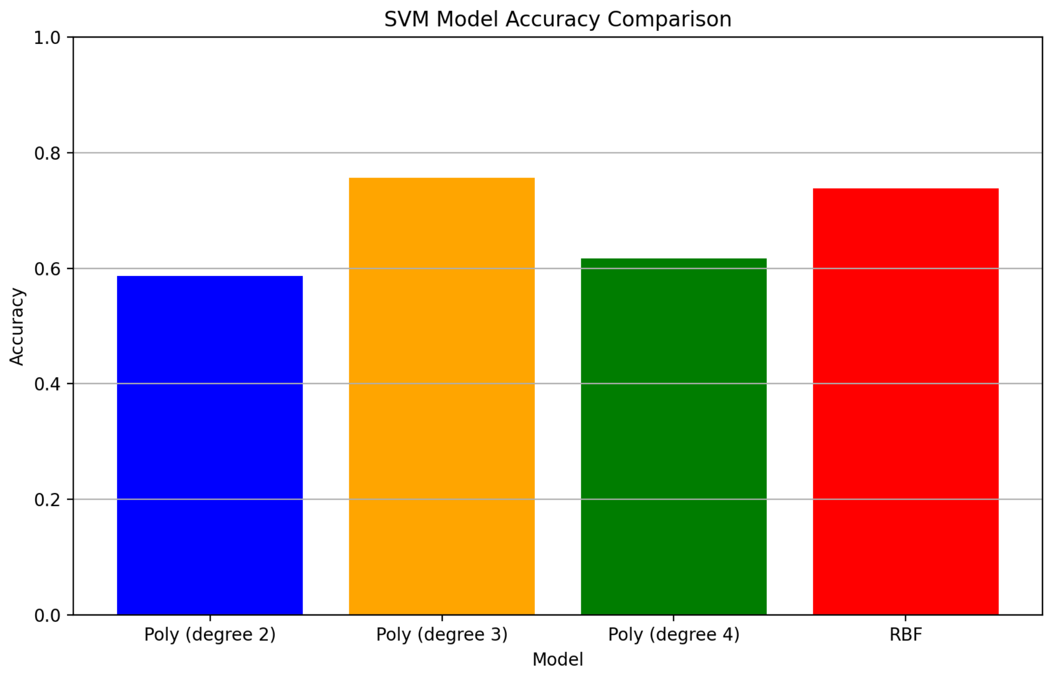
To explore the effectiveness of support vector machines (SVM) for predicting car sales, we estimated models using polynomial kernels of varying degrees (2, 3, and 4), as well as an SVM using a radial basis function (RBF) kernel. The performance of each SVM model was evaluated using the Mean Squared Error (MSE) obtained from 10-fold cross-validation.

This was done utilizing the following code:

```
# Define the different SVM
poly2_svc = svm.SVC(kernel='poly', degree=2)
poly3_svc = svm.SVC(kernel='poly', degree=3)
poly4_svc = svm.SVC(kernel='poly', degree=4)
radial_svc = svm.SVC(kernel='rbf')

# Estimate the models
svmp2= poly2_svc.fit(X,y)
svmp3= poly3_svc.fit(X,y)
svmp4= poly4_svc.fit(X,y)
svmr=radial_svc.fit(X,y)

# Cross-validate
scoresp2 = cross_val_score(svmp2, X, y,cv=10).mean()
scoresp3 = cross_val_score(svmp3, X, y,cv=10).mean()
scoresp4 = cross_val_score(svmp4, X, y,cv=10).mean()
scoresr = cross_val_score(svmr, X, y,cv=10).mean()
# prnt accuracy scores
print(scoresp2)
print(scoresp3)
print(scoresp4)
print(scoresr)

Outputs :
Polynomial_degree2 accuracy score: 0.5863636363636363
Polynomial_degree3 accuracy score: 0.756534090909091
Polynomial_degree4 accuracy score: 0.6173295454545454
Radial SVM accuracy score : 0.7377840909090909
```

5. Based on cross-validation, which method is optimal

Below is a bar chart to visualize the performance of the various models better**.** These scores indicate the average accuracy of each model across 10-fold cross-validation. The polynomial kernel of degree 3 as evident in the visual achieved the highest accuracy, suggesting it may be the most effective model for this cars dataset.



6. Import carstest.csv. Estimate the accuracy score for the validation set and calculate the test score.

```
## Random forest model without split
X = carstest[carstest.columns.difference(['Sales'])]
X = (X-X.mean())/X.std()
```

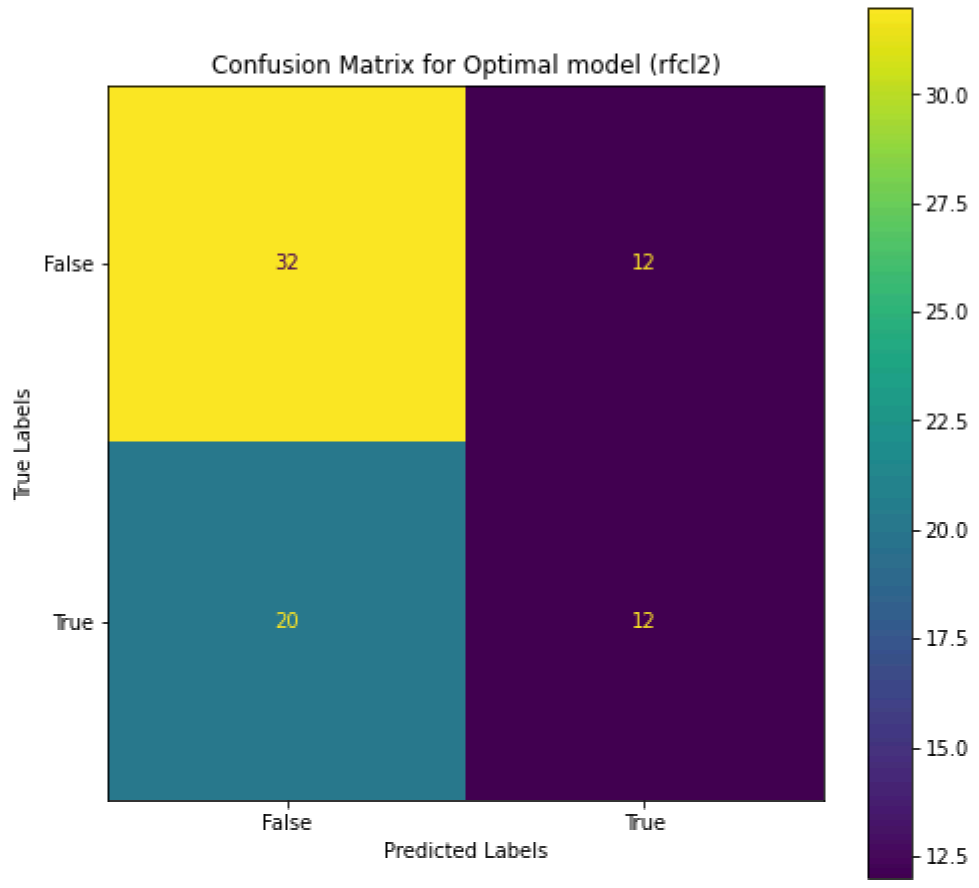
```
y = carstest['Sales']

# Develop and fit our model
rfcl1 = RandomForestClassifier(max_depth =4, random_state = 0)
rfcl1=rfcl1.fit(X,y)

# Cross-validate the rfcl1
scoresrfcl1 = cross_val_score(rfcl1, X, y,cv=10)
print('\n',f'rfcl1 accuracy score is: {scoresrfcl1.mean():.4f}')

rfcl1 accuracy score is: 0.5821

## Random forest model on split dataset
X = carstest[carstest.columns.difference(['Sales'])]
X = (X-X.mean())/X.std()
y = carstest['Sales']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

# Develop and fit our model
rfcl2 = RandomForestClassifier(max_depth =4, random_state = 0)
rfcl2=rfcl2.fit(X_train,y_train)

# Cross-validate the rfcl2
scoresrfcl2 = cross_val_score(rfcl2, X_train, y_train,cv=10)
print('\n',f'rfcl2 accuracy score is: {scoresrfcl2.mean():.4f}')

# Lets calculate the test score
y_pred = rfcl2.predict(X_test)

# Calculate the Means square error
np.mean((y_test - y_pred)**2)
print('\n',f'the test score is: {np.mean((y_test - y_pred)**2):.4f}')

rfcl2 accuracy score is: 0.7000

 the test score is: 0.3750
```

7. Show the confusion matrix for the optimal method. Does it manage to predict high sales and not high sales with equal precision?

From the above the optimal model is the model is the model trained on split dataset (rfcl2)

```
# Confusion matrix for optimal model
```

```
y_pred = cross_val_predict(rfcl2, X, y, cv=10)
conf_mat = confusion_matrix(y, y_pred)
# Step 2
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
conf_mat, display_labels = [False, True])
fig, ax = plt.subplots(figsize=(8, 8))
cm_display.plot(ax=ax)
# Add title and axis labels
plt.title("Confusion Matrix for Optimal model (rfcl2)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



To determine if the model is able to predict high sales and not high sales with equal precision, we need to analyze the confusion matrix. The confusion matrix provides information about the true positive, true negative, false positive, and false negative predictions made by the model.

From the confusion matrix plot, we can see the following:

True Positive (TP): The number of instances where the model correctly predicted high sales.

True Negative (TN): The number of instances where the model correctly predicted not high sales. False Positive (FP): The number of instances where the model incorrectly predicted high sales. False Negative (FN): The number of instances where the model incorrectly predicted not high sales.

To assess the model's performance in predicting high sales and not high sales,we will use the following metrics

1. Precision for high sales: Precision = TP / (TP + FP) This metric tells us how many of the instances predicted as high sales were actually high sales.
2. Precision for not high sales: Precision = TN / (TN + FN) This metric tells us how many of the instances predicted as not high sales were actually not high sales.

Applying our metrics to our confusion matrix we can calculate the precision for each class as follows:

Precision for high sales: Precision = TP / (TP + FP) Precision = 12 / (12 + 20) = 0.375 or 37.5%
Precision for not high sales: Precision = TN / (TN + FN) Precision = 32 / (32 + 12) = 0.727 or 72.7%

Based on these values, the model appears to have a higher precision in predicting not high sales (72.7%) compared to predicting high sales (37.5%). This suggests that the model is better at identifying instances of not high sales, but it is not performing as well in predicting high sales. In addition, the model seems to have a higher rate of false positives for the high sales class, which is contributing to the lower precision.

8. Is the optimal method based on cross-validation also optimal for the test set? Comment the results

```
# Cross-validate the rfcl on the test set
test_scores = cross_val_score(rfcl2, X_test, y_test, cv=10)
print('\n', f'rfcl2 test set accuracy score is: {test_scores.mean():.4f}')
```

```
rfcl2 test set accuracy score is: 0.7000
```

When we compare the performance on the training set to that of the test set, it can be observed that the accuracy of the model on the training and test sets is 0.7 which suggests that the model is performing well and is optimal for the test set. A lower score on the test set cross validation would have been an indication that the model performs is overfitting to our training set and as such performing poorly.