Group 5 Team Programming Project: Catharine Tarquinio, Robert Gorman, Dominick Filar

A.1:

Cross-validation Fold	Decision Tree	Logistic Regression
Fold 1	0.6973	0.6527
Fold 2	0.7167	0.65
Fold 3	0.702	0.654
Fold 4	0.6907	0.648
Fold 5	0.7093	0.652
Fold 6	0.686	0.6127
Fold 7	0.692	0.6233
Fold 8	0.712	0.6127
Fold 9	0.702	0.634
Fold 10	0.7093	0.6333
Average Error %	0.2983	0.3627
Std. Dev. Error %	0.0096	0.0156

The decision tree is a more accurate predictor than logistic regression. Decision trees had an average accuracy of 0.70173, compared to logistic regression having an average accuracy of 0.63727.

A.4:

(p, Y) – True Positive: Correctly identifying loyal customers allows the company to retain customers at a lower cost than finding new customers.

(n, Y) – False Positive: Wrongly identifying non-customers as loyal customers leads to less profit and less loyal customers long term.

(p, N) – False Negative: Wrongly identifying loyal customers as non-customers results in lost revenue from the lifetime value of lost customers.

(n, N) – True Negative: Correctly predicting non-customers helps allocate the company's resources more efficiently to potential new paying customers.

Hypothetical Values

Retention Savings: $5 per customer.
Lost Revenue (churn): $10 per customer.
Retention Cost (High Value Customer): $2.50 per customer
Retention Cost (Low Value Customer): $1.50 per customer

True Positive (p, Y)
(Retention Savings) + (Saved Revenue) - (Retention Cost) = Net Retention Benefit
$5 + $10 - $2.50 = $12.50

False Positive (n, Y)
$0 - (Retention Cost) - (Unnecessary retention) = Optimized Retention Cost
$0 - $2.50 - $1.50 = -$4

False Negative (p, N)
$0 - (Lost Revenue) = -$10

True Negative (n, N)
$0

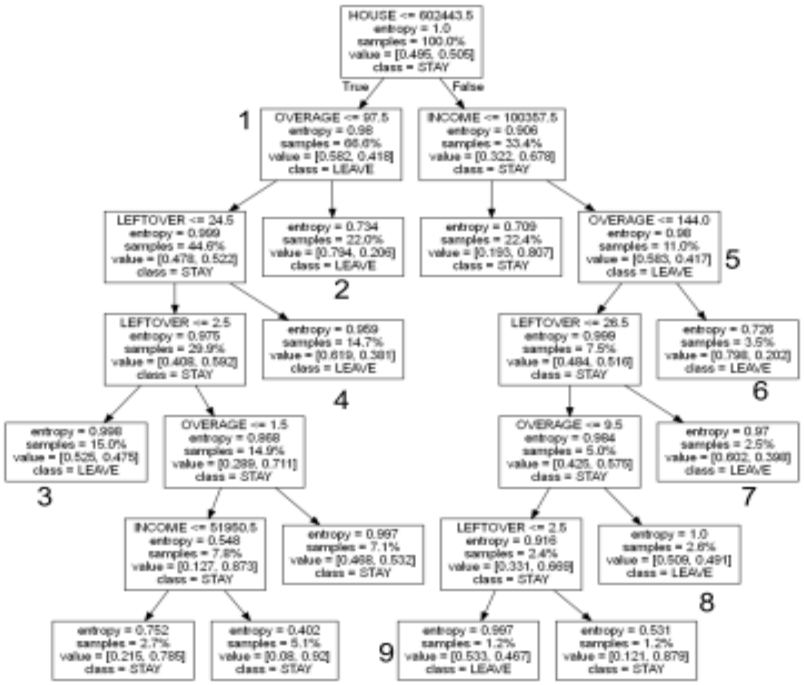
Confusion Matrix:

Predicted Class	**Actual Class**		
		p	**n**
	Y	$12.50 Gain	$4.00 Cost
	N	$10.00 Cost	$0.00 No Cost & No Gain

A True Positive (p,Y) results in a net benefit of $12.50 because we retain a loyal customer (and their customer lifetime value, or CLV) and avoid churn, whereas a False Positive (n, Y) results in a loss of -$4 due to unnecessary retention efforts spent on a non-customer. False Negatives (p, N) lead to a loss of -$10 due to lost revenue (CLV) from a loyal customer not being retained, while a True Negative (n, N) provides a benefit of $0 for correctly identifying a non-customer and not wasting resources.

A.5:

Churn Segment	Description	Samples (%)
1	41.8% of customers churn as a result of overages that they feel are not worth it in their plan.	66.6%
2	20.6% of customers churn for separate reasons compared to the other customers who churn due to overages	22.0%
3	47.5% of customers churn due to effects from having hardly any leftover in their plan	15.0%
4	38.1% of customers churn citing different reasons pertaining to leftover usage than the other nodes within the tree	14.7%
5	41.7% of customers churn with an increased overage value, as well as their housing income being greater than what the tree established	11.0%
6	20.2% of customers churn due to overage citing different reasons than the other node of the tree	3.5%
7	39.8% of customers churn due to other reasons pertaining to having less leftover in their plan	2.5%
8	49.1% of customers churn due to other reasons due to having less overage in their plan than anticipated	2.6%
9	46.7% of customers churn due to other reasons surrounding having less leftover in their plan than anticipated	1.2%



Our group believes that segment 6 is the best for identifying churn due to having 0.998 entropy, 47.5% churn rate, while also including 15% of the dataset. Although there are two segments with a higher % of the dataset (segments 1 & 2), we believe that this segment having the highest churn rate while having the highest entropy makes it the best choice to focus on. Having the churn rate split [0.525, 0.475] can allow for the strength of a predictive model to determine if a customer will stay or leave to be shown.

A.2 Appendix:

In [11]:
```
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)

kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state=0)

dt_accuracy_train = cross_val_score(decision_tree, X_train_scaled, y_train, cv = kfold, scoring = 'accu

accuracy = np.mean(dt_accuracy_train)
std = np.std(dt_accuracy_train)

dt_error = (1 - dt_accuracy_train)

dt_mean_error = np.mean(dt_error)
dt_std_error = np.std(dt_error)

print("10-Fold Accuracy:", dt_accuracy_train)
print("Mean Accuracy:", accuracy)
print("Standard Deviation:", std)

print("Average Error:", dt_mean_error)
print("Average Std. Dev. Error:", dt_std_error)
```

```
10-Fold Accuracy: [0.69733333 0.71666667 0.702      0.69066667 0.70933333 0.686
 0.692      0.712      0.702      0.70933333]
Mean Accuracy: 0.7017333333333333
Standard Deviation: 0.009634198346400067
Average Error: 0.2982666666666667
Average Std. Dev. Error: 0.009634198346400067
```

In [12]:
```
from IPython.display import Image
from sklearn.tree import export_graphviz

def visualize_tree(decision_tree, feature_names, class_names, directory="./images", name="tree",proport

    directory1 = directory[2:]
    os.system("mkdir %s" %(directory1))
    dot_name = "%s/%s.dot" % (directory, name)
    dot_file = export_graphviz(decision_tree, out_file=dot_name,feature_names=feature_names,class_names
                               proportion=proportion)

    image_name = "%s/%s.png" % (directory, name)
    print(dot_name)
    print(image_name)
    os.system("dot -Tpng %s  -o %s" % (dot_name, image_name))

    return Image(filename=image_name)

visualize_tree(decision_tree, predictor_cols, ["LEAVE", "STAY"])
```

```
./images/tree.dot
./images/tree.png
```

A.3 Appendix:

In [10]:
```
scale = StandardScaler()
X_train_scaled = scale.fit_transform(X_train)

kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 0)

lr_regression = LogisticRegression(max_iter = 100000, random_state = 0)
lr_accuracy_train = cross_val_score(lr_regression, X_train_scaled, y_train, cv = kfold, scoring='accura

accuracy = np.mean(lr_accuracy_train)
std = np.std(lr_accuracy_train)

lr_error = (1 - lr_accuracy_train)

lr_mean_error = np.mean(lr_error)
lr_std_error = np.std(lr_error)

print("10-Fold Accuracy:", lr_accuracy_train)
print("Mean Accuracy:", accuracy)
print("Mean Standard Deviation:", std)

print("Average Error:", lr_mean_error)
print("Average Std. Dev. Error:", lr_std_error)
```

```
10-Fold Accuracy: [0.65266667 0.65       0.654      0.648      0.652      0.61266667
 0.62333333 0.61266667 0.634      0.63333333]
Mean Accuracy: 0.6372666666666668
Mean Standard Deviation: 0.015627468693866508
Average Error: 0.36273333333333335
Average Std. Dev. Error: 0.015627468693866508
```

Part B-1:

The coefficients below are the results from the mode, the main takeaway is that the usage of a Simmons card which is labeled as “X2”, shows a positive correlation.

LR coefficients:
BETA0 (or constant term): -2.0067
BETA1 (coeff. For X1): 0.3299
BETA2 (coeff. For X2): 0.9179

Odds Ratios:
X1: 1.3908
X2: 2.5040

Part B-2:

The results from the model are shown below, two customers were compared; Jack, who spends $2000 annually (X1 = 2) and has a Simmons card (X2 = 1) & Jill, who spends $4000 annually (X1 = 4) but does not have a Simmons card (X2 = 0).

Probability of Response from Jack = 0.3944

Probability of Response from Jill = 0.3347

Jack is more likely to use the coupon, as indicated by his higher probability of response compared to Jill. This outcome shows the significant role of the Simmons loyalty card in influencing coupon usage, even outweighing higher annual spending.

Part B-3:

After looking at the logistic regression model results, a few different indicators were found for rolling out coupon predictions to a large customer database…

The model coefficients:

BETA0 (constant coeff): -2.0067

BETA1 (coeff. For X1): 0.3299

BETA2 (coeff. For X2): 0.9179

Odds Ratios:

X1: 1.3908 (Spending)

X2: 2.5040 (Card Ownership)

These both show that spending and card ownership positively correlate with coupon usage, with card ownership having a stronger correlation. We should consider that the model predicted probabilities of a response of 0.3944 for Jack and 0.3347 for Jill when trying to figure out a cutoff probability. We think that a possible cutoff of 0.35 might be a good number to start off at because it would classify Jack as likely to use the coupon and Jill as unlikely.

But in order to actually make the cutoff as beneficial as it should be, there are a few steps that we need to start with. This includes splitting our data into training and test sets. Then, along with trial and error in picking cutoff points, we should use confusion matrices and the cost-benefit matrix to see how accurate the predictions may be. In the end, the cutoff we choose should be found to balance the business goals we are trying to achieve and take into account coupon costs and benefits as well.

Appendix B:

B-1

In [5]:
```
predictor_cols = ["Spending(000)","Card"]
target_col = "Coupon-Usage-Indicator"

model = LogisticRegression()
model.fit(df[predictor_cols].values,df[target_col])
beta0 = model.intercept_[0]
beta1 = model.coef_[0][0]
beta2 = model.coef_[0][1]
print('LR coefficients:')
print('BETA0 (or constant term): {:.4f}'.format(beta0))
print('BETA1 (coeff. For X1 ): {:.4f}'.format(beta1))
print('BETA2 (coeff. For X2): {:.4f}'.format(beta2))
print('\n')
print('ODDS RATIOs:')
print('X1 : {:.4f}'.format(np.exp(beta1)))
print('X2 : {:.4f}'.format(np.exp(beta2)))
```

```
LR coefficients:
BETA0 (or constant term): -2.0067
BETA1 (coeff. For X1 ): 0.3299
BETA2 (coeff. For X2): 0.9179


ODDS RATIOs:
X1 : 1.3908
X2 : 2.5040
```

B-2

In [6]:
```
jack=[ [2.0, 1.0] ]
jill=[ [4.0, 0.0] ]
def predict_coupon_usage(X):
 pred_val = model.predict_proba(X)[:,1]
 return(pred_val)

print("Probability of Response from Jack = {:.4f}".format(predict_coupon_usage(jack)[0]))
print("Probability of Response from Jill = {:.4f}".format(predict_coupon_usage(jill)[0]))
```

```
Probability of Response from Jack = 0.3944
Probability of Response from Jill = 0.3347
```

Jack is more likely to respond.