Mladen Kolar

Machine Learning
Winter 2023

Midterms

Due on Sunday, Feb 19

Solution - Aman Krishna

Email - amank@uchicago.edu

In [1]:

```
# First we import the necessary libraries
import pandas as pd
pd.set_option("display.precision", 4)
import numpy as np
from matplotlib import pyplot as plt
from prettytable import PrettyTable
from statsmodels.formula.api import ols
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
pd.options.display.float_format = '{:.4f}'.format
pd.options.mode.chained_assignment = None  # default='warn'
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

1 Question [30 points]

The file eBayAuctions.csv contains information on 1972 auctions transacted on eBay.com during May–June 2004. The goal is to use these data to build a model that will distinguish competitive auctions from noncompetitive ones. A competitive auction is defined as an auction with at least two bids placed on the item being auctioned. The data include variables that describe the item (auction category), the seller (his or her eBay rating), and the auction terms that the seller selected (auction duration, opening price, currency, day of week of auction close). In addition, we have the price at which the auction closed. The goal is to predict whether or not an auction of interest will be competitive.

Pull the relevant data

In [2]:

```
data = pd.read_csv("eBayAuctions.csv")
data.head(5)
```

Out[2]:

	Category	currency	sellerRating	Duration	endDay	ClosePrice	OpenPrice	Competitive?
0	Music/Movie/Game	US	3249	5	Mon	0.0100	0.0100	0
1	Music/Movie/Game	US	3249	5	Mon	0.0100	0.0100	0
2	Music/Movie/Game	US	3249	5	Mon	0.0100	0.0100	0
3	Music/Movie/Game	US	3249	5	Mon	0.0100	0.0100	0
4	Music/Movie/Game	US	3249	5	Mon	0.0100	0.0100	0

Split the data into training and test sets and 1hot encode the categorical variables

In [3]:

```
#create dummy variables for the categorical variables ['Category', 'currency', 'endDay']
from sklearn.model_selection import train_test_split
data = pd.get_dummies(data, columns=['Category', 'currency', 'endDay'], drop_first=True)
features = data.columns.drop(['Competitive?','ClosePrice'])
target = 'Competitive?'
X_train, X_val, y_train, y_val = train_test_split(data[features], data[target], test_size=0.4, random_state=410)
```

1.1 Discuss if you can use all the variables to predict at the start of an auction whether it will be competitive.

To determine whether we can use all the given variables to predict whether an auction will be competitive at the start, we need to consider the relevance and relationship of each variable to the target variable (i.e., "Competitive?").

- Category: The type of product being sold could potentially have an impact on competitiveness, as some categories may attract more bidders than others.
- Currency: The currency used to price the item may impact the competitiveness, as different currencies can affect the price and thus bidding behavior.
- SellerRating: The rating of the seller could have an impact on the number of bidders and the competitiveness of the auction, as buyers may be more likely to bid on items from more reputable sellers.
- Duration: The length of the auction could impact competitiveness, as longer auctions may attract more bidders and lead to more competition.
- EndDay: The day of the week on which the auction ends could also impact competitiveness, as some days may attract more bidders than others.
- **ClosePrice: The final price of the auction will directly reflect the level of competitiveness. However, it is not a variable that can be used to predict competitiveness at the start of the auction.**
- OpenPrice: The starting price of the auction could have an impact on competitiveness, as a low starting price could attract more bidders and lead to more competition.

  Since we cannot use ClosePrice to predict competitiveness, we will drop it from the dataset.

1.2 Build a classification tree, a boosted tree, a bagged tree, and a random forest model (with mtry = 4). Choose the tuning parameters for these models by optimizing the performance on the validation set. Report accuracies of these four models as well as their confusion matrices on the validation set.

In [4]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import  GridSearchCV
import xgboost as xgb
from sklearn.model_selection import KFold

def parameter_search(X_train, y_train, X_val, y_val,model):
    #Create combinations of parameters randomly and calculate the accuracy score
    #for each combination
    np.random.seed(420)
    max_depth = np.random.randint(1, 50, 1)
    min_samples_leaf = np.random.randint(1, 25, 1)
    min_samples_split = np.random.randint(1, 25, 1)
    n_estimators = np.random.randint(50, 900, 50)
    learning_rate = np.random.uniform(0.1, 2.0, 0.1)
    base_estimator__max_depth = np.random.randint(1, 25, 1)
    max_features = np.random.randint(0.1, 10, 0.1)



def build_models(X_train, y_train, X_val, y_val):

    # Tune parameters for classification tree
    tree = DecisionTreeClassifier(random_state=420)
    tree_param_grid = {'max_depth': [34,45], 'min_samples_leaf': [9,10], 'min_samples_split': [11,12]}
    tree_grid_search = GridSearchCV(tree, tree_param_grid, cv=5, n_jobs=-1)
    tree_grid_search.fit(X_train, y_train)
    tree = tree_grid_search.best_estimator_
    print("Best parameters for classification tree: ", tree_grid_search.best_params_)

    # Tune parameters for boosted tree
    tree_params = {'max_depth': tree.max_depth}
    boost = AdaBoostClassifier(base_estimator=tree, random_state=420)
    boost_param_grid = {'n_estimators': [210,220,230,240], 'learning_rate': [1.35,1.4,1.45,1.5], 'base_estimator__max_depth': [
    boost_grid_search = GridSearchCV(boost, boost_param_grid, cv=5, n_jobs=-1)
    boost_grid_search.fit(X_train, y_train)
    boost = boost_grid_search.best_estimator_
    print("Best parameters for boosted tree: ", boost_grid_search.best_params_)

    # Tune parameters for bagged tree
    bag = BaggingClassifier(base_estimator=tree, random_state=420)
    bag_param_grid = {'n_estimators': [160,170,180], 'max_samples': [0.5,0.7, 1], 'max_features': [0.3, 0.5,0.7]}
    bag_grid_search = GridSearchCV(bag, bag_param_grid, cv=5, n_jobs=-1)
    bag_grid_search.fit(X_train, y_train)
    bag = bag_grid_search.best_estimator_
    print("Best parameters for bagged tree: ", bag_grid_search.best_params_)

    # Tune parameters for XGBClassifier
    xgb_model = xgb.XGBClassifier(objective ='reg:squarederror', colsample_bytree = 0.3, alpha = 10)
    kf = KFold(n_splits=5)
    param_grid = {'max_depth': [3,4,5],'n_estimators': [100,200,300],'learning_rate': [0.1,0.2,0.3]}
    xgb_grid_search = GridSearchCV(xgb_model, param_grid, cv=kf, verbose=0)
    xgb_grid_search.fit(X_train, y_train)
    xgb_model = xgb_grid_search.best_estimator_
    print("Best hyperparameters for XGBoost: ", xgb_grid_search.best_params_)

    # Tune parameters for random forest
    rf = RandomForestClassifier(random_state=420)
    rf_param_grid = {'n_estimators': [200,220, 250,300], 'max_features': [4],'max_depth': [10,11,12,13,14]}
    rf_grid_search = GridSearchCV(rf, rf_param_grid, cv=5, n_jobs=-1)
    rf_grid_search.fit(X_train, y_train)
    rf = rf_grid_search.best_estimator_
    print("Best hyperparameters for rf tree: ", rf_grid_search.best_params_)

    # Fit models with optimized parameters
    tree.fit(X_train, y_train)
    boost.fit(X_train, y_train)
    bag.fit(X_train, y_train)
    xgb_model.fit(X_train, y_train)
    rf.fit(X_train, y_train)

    # Make predictions on validation set
    tree_pred = tree.predict(X_val)
    boost_pred = boost.predict(X_val)
    bag_pred = bag.predict(X_val)
    xgb_pred = xgb_model.predict(X_val)
    xgb_pred = np.where(xgb_pred > 0.5, 1, 0)
    rf_pred = rf.predict(X_val)

    # Calculate model accuracies and confusion matrices
    tree_acc = accuracy_score(y_val, tree_pred)
    boost_acc = accuracy_score(y_val, boost_pred)
    bag_acc = accuracy_score(y_val, bag_pred)
    xgb_acc = accuracy_score(y_val, xgb_pred)
    rf_acc = accuracy_score(y_val, rf_pred)

    tree_cm = confusion_matrix(y_val, tree_pred)
    boost_cm = confusion_matrix(y_val, boost_pred)
    bag_cm = confusion_matrix(y_val, bag_pred)
    xgb_cm = confusion_matrix(y_val, xgb_pred)
    rf_cm = confusion_matrix(y_val, rf_pred)

    # Return results using PrettyTable
    results = PrettyTable()
    results.field_names = ["Model","Accuracy Test Set", "Confusion Matrix"]
    results.add_row(["Classification Tree", f'{round(tree_acc*100,4)}%', tree_cm])
    results.add_row(["Boosted Tree", f'{round(boost_acc*100,4)}%', boost_cm])
    results.add_row(["Bagged Tree", f'{round(bag_acc*100,4)}%', bag_cm])
    results.add_row(["XGBoost", f'{round(xgb_acc*100,4)}%', xgb_cm])
    results.add_row(["Random Forest", f'{round(rf_acc*100,4)}%', rf_cm])
    return results, tree, boost, bag,xgb_model, rf
```

In [5]:

```
results,tree, boost, bag,xgb_model, rf = build_models(X_train, y_train, X_val, y_val)
print(results)
```

```
Best parameters for classification tree:  {'max_depth': 34, 'min_samples_leaf': 10, 'min_samples_split': 11}
Best parameters for boosted tree:  {'base_estimator__max_depth': 14, 'learning_rate': 1.4, 'n_estimators': 230}
Best parameters for bagged tree:  {'max_features': 0.3, 'max_samples': 0.7, 'n_estimators': 170}
Best hyperparameters for XGBoost:  {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 100}
Best hyperparameters for rf tree:  {'max_depth': 10, 'max_features': 4, 'n_estimators': 250}
+---------------------+-------------------+------------------+
|        Model        | Accuracy Test Set | Confusion Matrix |
+---------------------+-------------------+------------------+
| Classification Tree |      72.4968%     |    [[251 110]    |
|                     |                   |    [107 321]]    |
|     Boosted Tree    |      72.7503%     |    [[252 109]    |
|                     |                   |    [106 322]]    |
|     Bagged Tree     |      70.9759%     |    [[218 143]    |
|                     |                   |    [ 86 342]]    |
|       XGBoost       |      72.8771%     |    [[248 113]    |
|                     |                   |    [101 327]]    |
|    Random Forest    |      73.6375%     |    [[249 112]    |
|                     |                   |    [ 96 332]]    |
+---------------------+-------------------+------------------+
```

1.3 Create lift curves for these four models. What is the lift for the top 10% of observations in the validation set?

In [6]:

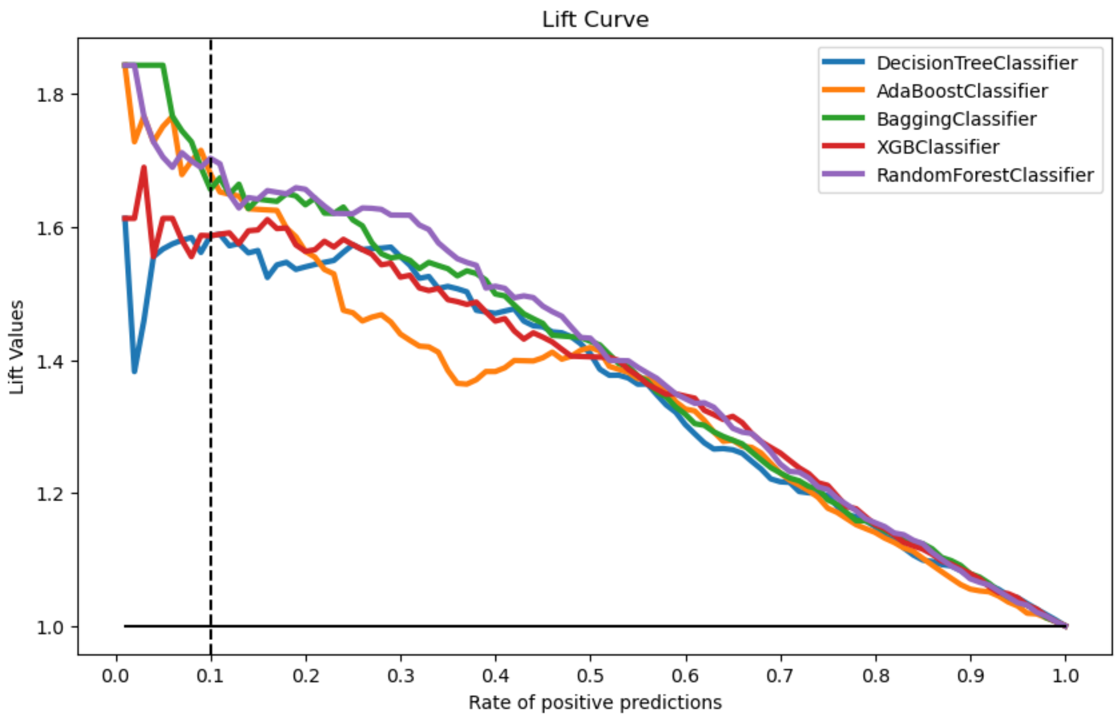
```
import scikitplot as skplt

# def get_lift_curves(models, X_train,X_val,y_train,y_val):
#     for model_name, model in models.items():
#         # Fit model on training data
#         model.fit(X_train, y_train)

#         # Get predicted probabilities for test data
#         y_proba = model.predict_proba(X_val)

#         # Generate lift curve
#         skplt.metrics.plot_lift_curve(y_val, y_proba, title=f'{model_name} Lift Curve')

#     return y_proba
```

In [7]:

```
def plot_Lift_curve(models, X_val, y_val, step=0.01):

    # Create a figure for the lift curve of 4 models
    fig, ax = plt.subplots(figsize=(10, 6))
    lift_10_percent = {}
    for model in models:
        if model.__class__.__name__ == 'XGBRegressor':
            continue
        y_pred = model.predict_proba(X_val)[:, 1]


        aux_lift = pd.DataFrame()

        aux_lift['real'] = y_val
        aux_lift['predicted'] = y_pred
        aux_lift.sort_values('predicted', ascending=False, inplace=True)

        x_val = np.arange(step, 1 + step, step)

        ratio_ones = aux_lift['real'].sum() / len(aux_lift)

        y_v = []


        for x in x_val:
            num_data = int(np.ceil(x * len(aux_lift)))
            data_here = aux_lift.iloc[:num_data, :]
            ratio_ones_here = data_here['real'].sum() / len(data_here)
            y_v.append(ratio_ones_here / ratio_ones)

        ax.plot(x_val, y_v, linewidth=3, markersize=5, label=model.__class__.__name__)
        ax.set_xticks(np.arange(0, 1.1, 0.1))
        #save the lift curve for 10% of the data in a dictionary with key as model.__class__.__name__
        lift_10_percent[model.__class__.__name__] = round(y_v[10],4)

    ax.plot(x_val, np.ones(len(x_val)), 'k-')
    ax.axvline(x=0.1, color='k', linestyle='--')
    ax.set_xlabel('Rate of positive predictions')
    ax.set_ylabel('Lift Values')
    plt.title('Lift Curve')
    plt.legend()
    plt.show()
    return lift_10_percent
```

In [8]:

```
lift_10 = plot_Lift_curve([tree, boost, bag,xgb_model, rf], X_val, y_val)
```



In [9]:

```
lift_10_table = PrettyTable()
lift_10_table.field_names = ["Model", "Lift at 10%"]
for key, value in lift_10.items():
    lift_10_table.add_row([key, value])
print(lift_10_table)
```

```
+------------------------+-------------+
|         Model          | Lift at 10% |
+------------------------+-------------+
| DecisionTreeClassifier |    1.5892   |
|   AdaBoostClassifier   |    1.6528   |
|   BaggingClassifier    |    1.6739   |
|     XGBClassifier      |    1.5892   |
| RandomForestClassifier |    1.6951   |
+------------------------+-------------+
```

The top 10% of the lift curve signifies how much better the model is at identifying positive class compared to a random guess, by indicating the ratio of the proportion of the positive class in that 10% compared to the overall proportion of the positive class in the dataset.

The Random Forest model has the highest lift for the top 10% of the validation set, with a lift of approximately 1.7. The Decision Tree model has the lowest lift for the top 10% of the validation set, with a lift of approximately 1.6.

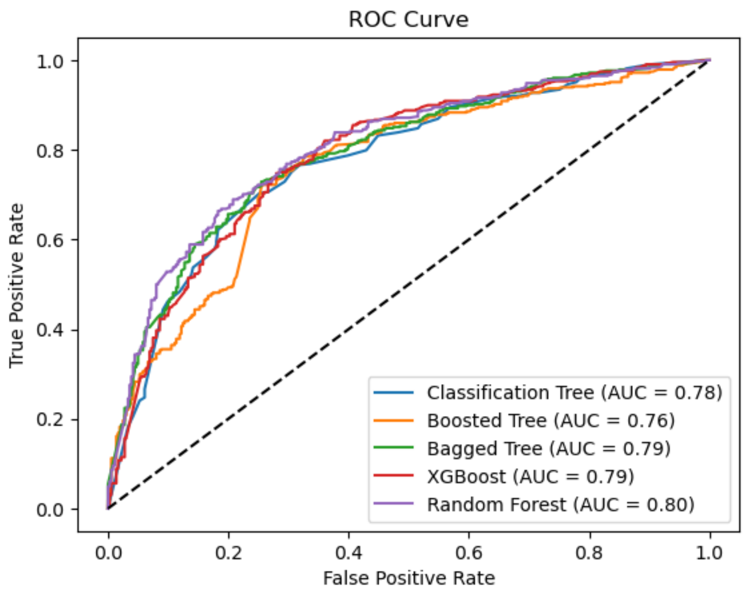
This is as expected since we got the highest accuracy for the Random Forest model, and the lowest accuracy for the Decision Tree model in our previous question.

In [10]:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def plot_roc(models, X_train, X_val, y_train, y_val):
    # Plot ROC curve for each model
    for model_name, model in models.items():
        # Fit model on training data
        model.fit(X_train, y_train)

        # Get predicted probabilities for validation data
        y_proba = model.predict_proba(X_val)[:, 1]

        # Compute ROC curve and AUC score
        fpr, tpr, thresholds = roc_curve(y_val, y_proba)
        roc_auc = auc(fpr, tpr)

        # Plot ROC curve
        plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

    # Plot the random classifier line
    plt.plot([0, 1], [0, 1], color='black', linestyle='--')

    # Set plot properties
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

In [11]:

```
models = {
    'Classification Tree': tree,
    'Boosted Tree': boost,
    'Bagged Tree': bag,
    'XGBoost': xgb_model,
    'Random Forest': rf
}
plot_roc(models, X_train, X_val, y_train, y_val)
```



1.4 Briefly describe how boosted tree, bagged tree and random forest models are conceptually similar and how are they conceptually different.

Boosted Tree, Bagged Tree, and Random Forest models are similar in that they all are group (ensemble) methods that combine or aggregate multiple decision trees to make a more accurate prediction than using a single decision tree. Bagging and boosting are known as ensemble meta-algorithms.

Boosted Tree, Bagged Tree, and Random Forest models are different in the way they construct these decision trees, and how they combine them.

Boosted Tree models construct decision trees sequentially, where each new tree is created to improve on the mistakes of the previous tree. Boosting transforms weak decision trees (called weak learners) into strong learners. The final prediction is a weighted sum of all the decision trees.

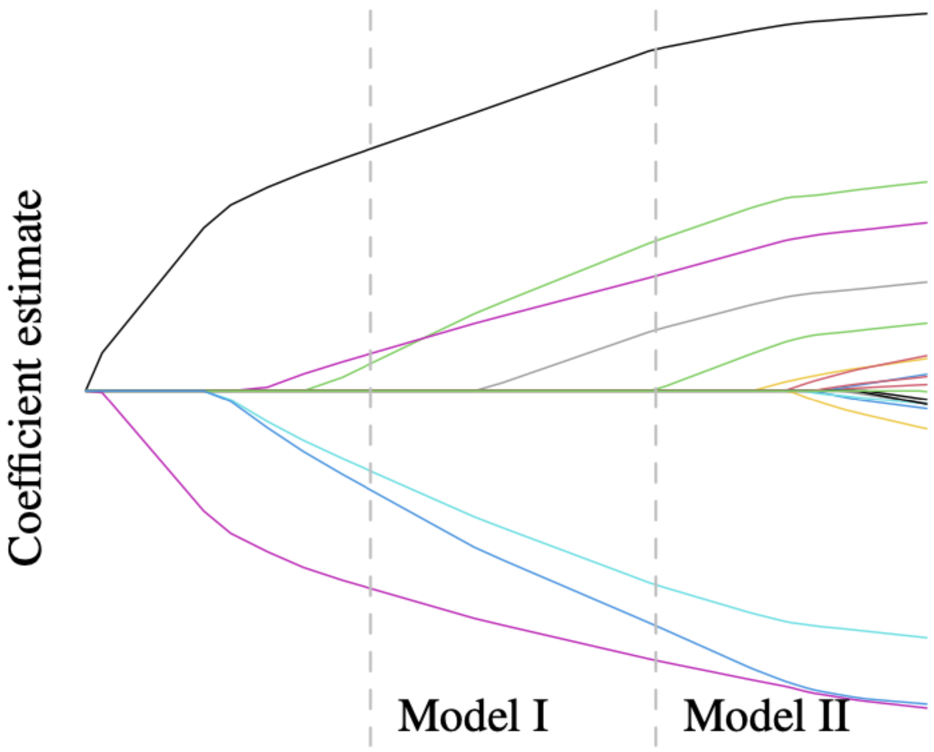
Bagged Tree models construct decision trees independently, where each tree is trained on a bootstrap sample of the data. Bootstrap aggregation (i.e., bagging) is a general technique that combines bootstraping and any regression/classification algorithm to construct an ensemble.

Random Forest models construct decision trees in the same way as Bagged Trees, but at each node of the decision tree only a random subset of features is considered. Random forest is an extension of bagging that also randomly selects subsets of features used in each data sample. The final prediction is an average of all the decision trees.

2 Question [32 points]

2.1 The following plot is the solution path for a Lasso estimate. The two dashed lines correspond to two values of the

tuning parameter λ, and hence two models.



a. Model I uses a larger λ than Model II.

> **TRUE** Model I uses a larger λ than Model II, as for Lasso regression increasing the value of λ increases the amount of shrinkage applied to the coefficients. This may result in some coefficients being shrunk to zero, which would result in a smaller number of non-zero coefficients in Model I than in Model II.

b. Model I has a higher variance than Model II.

> **FALSE** Model I is a simpler model than Model II, as it has fewer non-zero coefficients. The Lasso regression method shrinks the coefficient estimates towards zero by imposing a penalty on the sum of the absolute values of the coefficients. This is because, as the value of lambda increases, the penalty becomes more severe, resulting in a more constrained model with fewer non-zero coefficients. This process helps to prevent overfitting and reduces the variance of the model.

c. Model I has a larger in-sample RSS than Model II.

> **TRUE** When the value of λ increases, the Lasso penalty becomes more severe, resulting in a sparser model with fewer non-zero coefficients. This leads to a decrease in model complexity, which is characterized by the shrinkage of the coefficient estimates towards zero. As a result, the model becomes more interpretable, with fewer variables affecting the outcome.
>
> However, as λ grows, the in-sample loss increases, leading to a higher residual sum of squares (RSS). This trade-off between model accuracy and model complexity is known as the bias-variance trade-off. As λ increases, the bias of the model increases, leading to a higher RSS.

d. Model I has a larger test error than Model II.

> **FALSE** It is not possible to determine whether Model I has a larger test error than Model II, as we do not have access to the test error for either model. We can only compare the in-sample RSS for each model, which is what we did in the previous question.

2.2 Suppose that we would like to fit a linear model to predict MLB players' salaries based on their basic information and statistics for the previous season. The following table shows the residual sum of squares for all the one- predictor models.

Model	RSS
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{AtBat}$	4.50×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Hits}$	4.30×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{HmRun}$	4.70×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Runs}$	4.39×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{RBI}$	4.25×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Walks}$	4.28×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Years}$	4.47×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{League}$	5.33×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Division}$	5.13×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{PutOuts}$	4.85×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Assists}$	5.32×10^7
$\widehat{\texttt{Salary}} = \beta_0 + \beta_1 \times \texttt{Errors}$	5.33×10^7

Based on the information given by this table, it is certain that

a. The two-variable model identified by forward stepwise selection includes RBI.

> **TRUE** The forward stepwise selection method starts with a null model, and then adds variables one at a time, based on the lowest RSS. The two-variable model identified by forward stepwise selection includes RBI, as it has the lowest RSS of all the one-variable models.

b. The two-variable model identified by forward stepwise selection includes Walks.

> **FALSE** We already know that the two-variable model identified by forward stepwise selection includes RBI. Once we choose the first variable (RBI) we are unaware of the new RSS values for the two variable regressions for the remaining variables with RBI and one fixed variable, so we cannot determine which variable would be the next to be added to the model.

c. The two-variable model identified by best subset selection includes RBI.

> **FALSE** The 1-variable model identified by best subset is the same as identified by forward stepwise selection. But the 2-variable model identified by best subset selection does not necessarily include RBI.

d. The two-variable model identified by backward stepwise selection includes RBI.

> **FALSE** The backward stepwise selection method starts with a full model, and then removes variables one at a time, based on the lowest RSS. Since, it greedily throws away variables because their addition is not useful at some given point, we might throw away the variable that would have been useful in the future. This is why the two-variable model identified by backward stepwise selection may not include RBI.

2.3 Do the following action increases the bias of the model?

a. Increase k, the number of nearest neighbors, in a kNN classifier.

> **TRUE** As k increases in the KNN model we get a simpler model which has a higher bias. This is because the model is less flexible and has a lower variance.

b. Add interaction terms to a linear regression model.

FALSE Adding interaction terms to a linear regression model increases the complexity of the model, which decreases the bias of the model.

c. Make more splits when growing a decision tree.

FALSE Making more splits when growing a decision tree generally increases the complexity of the model, which decreases the bias of the model.

d. Increase B, the number of trees in a random forest model.

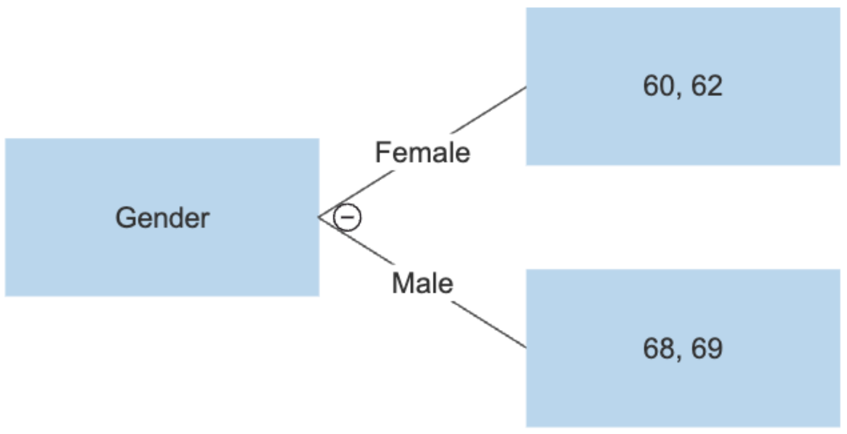
FALSE As B increases in the random forest model we get a more complex model which has a lower bias.

2.4 Suppose that we fit a decision tree to predict a person's height based on gender and age using the following dataset.

Gender	Age	Height (in)
M	20	69
F	21	62
F	15	60
M	19	68

a. The first split is on age.

FALSE The first split with gender as the predictor variable has a lower RSS than the split with age as the predictor variable.



b. The second split is on age.

TRUE The second split will be at age. This is simply as the there are no more splits possible in gender as it is a binary variable.

c. With two splits, the RSS is smaller than 1.

TRUE With two splits, the RSS is 1.25/2 (as caculate below). Thus it is smaller than 1.

d. With three splits, the RSS is 0.

TRUE With three splits, the RSS is 0, as we have perfectly predicted the height for all the observations.

In [12]:

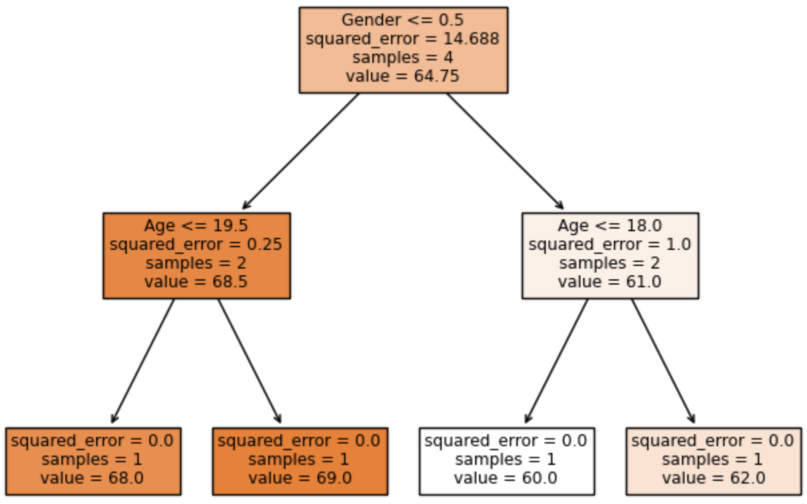
```
from sklearn.tree import DecisionTreeRegressor
import pandas as pd

# Define the training data
data = {'Gender': ['M', 'F', 'F', 'M'],
        'Age': [20, 21, 15, 19],
        'Height': [69, 62, 60, 68]}
df = pd.DataFrame(data)

# Split the data into features (X) and labels (y)
X = df[['Gender', 'Age']]
y = df['Height']

# Convert the gender feature to numerical values
X['Gender'] = X['Gender'].map({'M': 0, 'F': 1})

# Train the decision tree model
model = DecisionTreeRegressor()
model.fit(X, y)
```

Out[12]:

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

In [13]:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(8, 6))
plot_tree(model, feature_names=['Gender', 'Age'], filled=True)
plt.show()
```



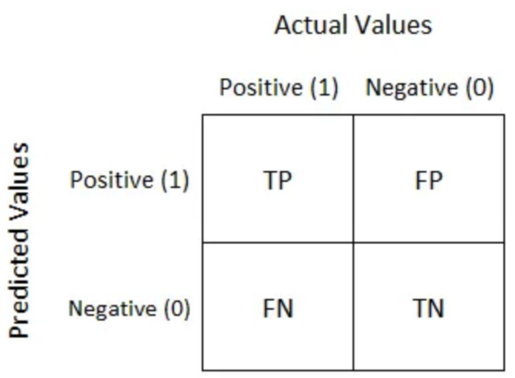
3 Question [30 points]

Your company has hired a consulting firm to help with the fraud problem. You are present in the meeting where the consulting firm presents the results of their pilot study, showing that the model has a very low error rate (percent incorrectly classified instances). They argue to your boss that based on this great performance, she should hire them to build the system. You need to explain to your boss the notions of false positive and false negative errors, and how the system should be evaluated. You may assume that the only relevant decision is (binary): if the system predicts fraud, block the account; if the system predicts no fraud, do nothing.
Explain the following notions:

a) describe the confusion matrix to your boss;

The confusion matrix is a table with four cells that shows the number of-

- true positives (The system correctly predict fraud when there was fraud),
- false positives (System incorrectly detects fraud when there is no real fraud),
- true negatives (System correctly does not detect fraud and there is none),
- false negatives (System incorrectly does not detect fraud while there is actual fraud)



b) describe how you will fill out the confusion matrix for the consultant's model;

The confusion matrix for the consultant's model is as follows:

- true positives (The system correctly predict fraud when there was fraud),
- false positives (System incorrectly detects fraud when there is no real fraud),
- true negatives (System correctly does not detect fraud and there is none),
- false negatives (System incorrectly does not detect fraud while there is actual fraud)

In [14]:

```python
x = PrettyTable()
x.field_names = [" ", "Predicted Fraud", "Predicted No Fraud"]
x.add_row(["Actual Fraud", "True Positive", "False Negative"])
x.add_row(["Actual No Fraud", "False Positive", "True Negative"])
print(x)
```

```
+-----------------+-----------------+--------------------+
|                 | Predicted Fraud | Predicted No Fraud |
+-----------------+-----------------+--------------------+
|   Actual Fraud  |  True Positive  |   False Negative   |
| Actual No Fraud |  False Positive |   True Negative    |
+-----------------+-----------------+--------------------+
```

c) describe the cost/benefit matrix for this problem;

The cost/benefit matrix will be different for each of the four cells in the confusion matrix.
The True Positive cell will have a **benefit** due to **prevention of loss**, as the system correctly predicts fraud and the account is blocked.
The False Positive cell will have a **cost** due to **unecessary blocking of account**, as the system incorrectly predicts fraud and the account is blocked.
The True Negative cell will have a **benefit** due to **no unecessary blocking**, as the system correctly predicts no fraud and the account is not blocked.
The False Negative cell will have a **cost** due to **loss of money**, as the system incorrectly predicts no fraud and the account is not blocked.

In [15]:

```python
x1 = PrettyTable()
x1.field_names = ["Cost/Benefit", "Predicted Fraud", "Predicted No Fraud"]
x1.add_row(["Actual Fraud", "+", "-"])
x1.add_row(["Actual No Fraud", "-", "+"])
print(x1)
```

```
+-----------------+-----------------+--------------------+
|   Cost/Benefit  | Predicted Fraud | Predicted No Fraud |
+-----------------+-----------------+--------------------+
|   Actual Fraud  |        +        |         -          |
| Actual No Fraud |        -        |         +          |
+-----------------+-----------------+--------------------+
```

d) explain briefly why the confusion matrix and the cost/benefit matrix are important for this problem(1-2sentences);

The confusion matrix and the cost/benefit matrix are important for this problem as they help us understand the performance of the model. The confusion matrix helps us understand the number of true positives, false positives, true negatives and false negatives. The cost/benefit matrix helps us understand the cost and benefit of each of the four cells in the confusion matrix.

e) show the proper evaluation function (equation) for the consultant's model;

The proper evaluation function for the consultant's model is the **accuracy**.
We can use the following measures:-

1. $Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}$
2. $Benefit = TP * \$/account + TN * \$/account$
   $Cost = FP * \$/account + FN * \$/account$
3. $Precision = \frac{TP}{TP+FP}$
   $Recall = \frac{TP}{TP+FN}$
   $F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$

f) how do the confusion and cost matrices come into play in this function.

The confusion matrix and the cost/benefit matrix come into play in this function as they help us understand the number of true positives, false positives, true negatives and false negatives. The cost/benefit matrix helps us understand the cost and benefit of each of the four cells in the confusion matrix. The accuracy function is a function of the number of true positives, false positives, true negatives and false negatives.

4 Question [40 points]

You will use data in marketing.csv for this question. This dataset contains 64,000 customers who last purchased within twelve months. The customers were involved in an e-mail test: 1/3 were randomly chosen to receive an e-mail campaign featuring a discount offer; 1/3 were randomly chosen to receive an e-mail campaign featuring a buy one get one free offer; 1/3 were randomly chosen to not receive an e-mail campaign. During a period of two weeks following the e-mail campaign, results were tracked. The first 8 columns provide individual-level data and conversion column is the label that we will try to predict:

- recency: months since last purchase
- history: $value of the historical purchases
- used_discount: indicates if the customer used a discount before
- used_bogo: indicates if the customer used a buy one get one before
- zip_code: class of the zip code as Suburban/Urban/Rural
- is_referral: indicates if the customer was acquired from referral channel
- channel: channels that the customer is using, Phone/Web/Multichannel
- offer: the offers sent to the customers, Discount/But One Get One/No Offer

The data were collected through an experiment that allows us to find growth opportunities. Splitting the customers who we are going to send the offer into test (groups receiving one of the two offers) and control groups helps us to calculate incremental gains of offers

a) Does giving an offer increase conversion? If yes, what kind of offer performs best? Discount or Buy One Get One?

In [16]:

```
data = pd.read_csv('marketing.csv')
data.head()
```

Out[16]:

	recency	history	used_discount	used_bogo	zip_code	is_referral	channel	offer	conversion
0	10	142.4400	1	0	Surburban	0	Phone	Buy One Get One	0
1	6	329.0800	1	1	Rural	1	Web	No Offer	0
2	7	180.6500	0	1	Surburban	1	Web	Buy One Get One	0
3	9	675.8300	1	0	Rural	1	Web	Discount	0
4	2	45.3400	1	0	Urban	0	Web	Buy One Get One	0

Missing value & Type check

In [17]:

```
data.isnull().sum()
```

Out[17]:

```
recency          0
history          0
used_discount    0
used_bogo        0
zip_code         0
is_referral      0
channel          0
offer            0
conversion       0
dtype: int64
```

In [18]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64000 entries, 0 to 63999
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   recency        64000 non-null  int64  
 1   history        64000 non-null  float64
 2   used_discount  64000 non-null  int64  
 3   used_bogo      64000 non-null  int64  
 4   zip_code       64000 non-null  object 
 5   is_referral    64000 non-null  int64  
 6   channel        64000 non-null  object 
 7   offer          64000 non-null  object 
 8   conversion     64000 non-null  int64  
dtypes: float64(1), int64(5), object(3)
memory usage: 4.4+ MB
```

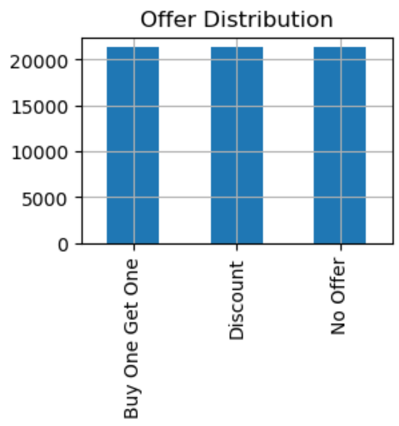
Now we will split the data based on the offer type and check the conversion rate for each group. We have been given that the data is split into three groups:

In [19]:

```
data.offer.value_counts().plot(kind='bar',grid=True,title='Offer Distribution', figsize=(3,2))
```

Out[19]:

```
<AxesSubplot:title={'center':'Offer Distribution'}>
```

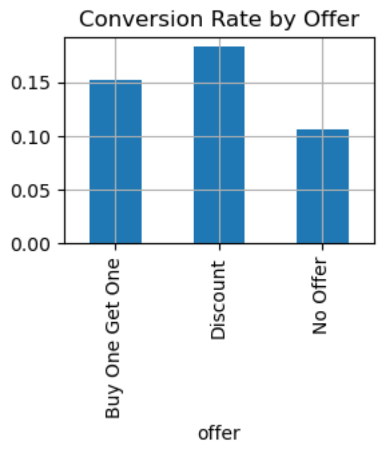


In [20]:

```
# Find the mean of the conversion rate for each offer
data.groupby('offer').conversion.mean().plot(kind='bar',grid=True,title='Conversion Rate by Offer', figsize=(3,2))
```

Out[20]:

```
<AxesSubplot:title={'center':'Conversion Rate by Offer'}, xlabel='offer'>
```



1. We see that giving any kind of offer (Discount or Buy One Get One) increases the conversion rate.
2. We see that the Discount offer performs better than the Buy One Get One offer over this particular sample.

For the rest of the question, we will focus on the customers who received the Discount offer or did not receive any offer.

In [21]:

```
#Drop the rows with Buy One Get One offer
data = data[data.offer != 'Buy One Get One']
```

In [22]:

```
data.shape
```

Out[22]:

```
(42613, 9)
```

b) Partition the data into 60% train and 40% validation set.

In [23]:

```
#Partitioning the data into 60% train and 40% validation set.
target = 'conversion'
treatment = 'offer'
data_dummies = pd.get_dummies(data)
data_dummies.drop('offer_No Offer', axis=1, inplace=True)
X_training, X_test,y_training,  y_test = train_test_split(data_dummies.drop(target, axis=1), data_dummies[target],test_size=0.
```

c) Build a model for P (conversion = 1 | offer, x), where x denotes the individual-level characteristics. Discuss how you chose the parameters used to build the model. Which one is it? Why did you choose it?

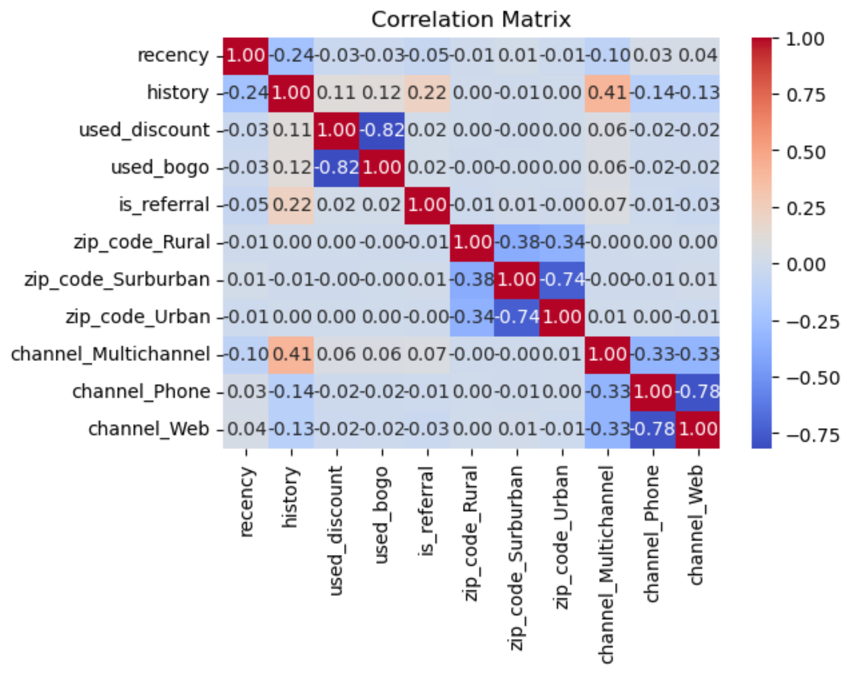
In [24]:

```
def plot_correlation_matrix(data, target, treatment):
    features = [col for col in data.columns if col not in [target, treatment]]
    dfX = data[features]
    dfXn = pd.get_dummies(dfX)
    plt.figure(figsize = (6,4))
    #sns.set(font_scale=0.75)
    sns.heatmap(dfXn.corr().round(3), annot=True, cmap="coolwarm", fmt=".2f")
    plt.title("Correlation Matrix")
    plt.show()
```

We start with a correlation plot of the input variable(s) "x" other than "offer"

In [25]:

```
plot_correlation_matrix(data, 'conversion', 'offer')
```



	recency	history	used_discount	used_bogo	is_referral	zip_code_Rural	zip_code_Surburban	zip_code_Urban	channel_Multichannel	channel_Phone	channel_Web
recency	1.00	-0.24	-0.03	-0.03	-0.05	-0.01	0.01	-0.01	-0.10	0.03	0.04
history	-0.24	1.00	0.11	0.12	0.22	0.00	-0.01	0.00	0.41	-0.14	-0.13
used_discount	-0.03	0.11	1.00	-0.82	0.02	0.00	-0.00	0.00	0.06	-0.02	-0.02
used_bogo	-0.03	0.12	-0.82	1.00	0.02	-0.00	-0.00	0.00	0.06	-0.02	-0.02
is_referral	-0.05	0.22	0.02	0.02	1.00	-0.01	0.01	-0.00	0.07	-0.01	-0.03
zip_code_Rural	-0.01	0.00	0.00	-0.00	-0.01	1.00	-0.38	-0.34	-0.00	0.00	0.00
zip_code_Surburban	0.01	-0.01	-0.00	-0.00	0.01	-0.38	1.00	-0.74	-0.00	-0.01	0.01
zip_code_Urban	-0.01	0.00	0.00	0.00	-0.00	-0.34	-0.74	1.00	0.01	0.00	-0.01
channel_Multichannel	-0.10	0.41	0.06	0.06	0.07	-0.00	-0.00	0.01	1.00	-0.33	-0.33
channel_Phone	0.03	-0.14	-0.02	-0.02	-0.01	0.00	-0.01	0.00	-0.33	1.00	-0.78
channel_Web	0.04	-0.13	-0.02	-0.02	-0.03	0.00	0.01	-0.01	-0.33	-0.78	1.00

As expected the used_discount and used_bogo are negatively correlated with each other, suggesting that customers dont use both the offers at the same time.

We cannot assume that customers who used the buy one get one offer will not use the discount offer, and hence will likely keep them in our dataset.

We will use Boruta and Forward Stepwise Regression to select the input variables for our model.

In [26]:

```
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFECV
from boruta import BorutaPy
from xgboost import XGBClassifier

def boruta_feature_selector(X_training, y_training, model=RandomForestClassifier(n_estimators=200, random_state=410)):
    # Boruta feature selection
    boruta = BorutaPy(model, n_estimators='auto', verbose=0, random_state=410)
    boruta.fit(X_training.values, y_training.values)

    # Print the importance of each X variable according to Boruta
    print("=========================================\nX Variable Importance according to Boruta:")
    table2 = PrettyTable()
    table2.field_names = ["X Variable", "Boruta Support", "Boruta Ranking"]
    #sort the features according to their importance
    for i in np.argsort(boruta.ranking_):
        table2.add_row([X_training.columns[i], boruta.support_[i], boruta.ranking_[i]])

    print(table2)

    # Print the selected features
    selected_boruta = X_training.columns[boruta.support_]
    print("============================================\nSelected Features by Boruta:", selected_boruta)
```

In [27]:

```
boruta_feature_selector(X_training, y_training, model=RandomForestClassifier(max_depth=5, random_state=410))
```

```
=========================================
X Variable Importance according to Boruta:
+----------------------+----------------+----------------+
|      X Variable      | Boruta Support | Boruta Ranking |
+----------------------+----------------+----------------+
|       recency        |      True      |       1        |
|       history        |      True      |       1        |
|     is_referral      |      True      |       1        |
|    offer_Discount    |      True      |       1        |
|    zip_code_Rural    |     False      |       2        |
|    used_discount     |     False      |       3        |
|    channel_Phone     |     False      |       4        |
|      used_bogo       |     False      |       5        |
|     channel_Web      |     False      |       6        |
| channel_Multichannel |     False      |       7        |
|  zip_code_Surburban  |     False      |       8        |
|    zip_code_Urban    |     False      |       8        |
+----------------------+----------------+----------------+
============================================
Selected Features by Boruta: Index(['recency', 'history', 'is_referral', 'offer_Discount'], dtype='object')
```

In [28]:

```
boruta_feature_selector(X_training, y_training, model=XGBClassifier(n_estimators=200, random_state=410,eval_metric='error'))
```

```
=========================================
X Variable Importance according to Boruta:
+----------------------+----------------+----------------+
|      X Variable      | Boruta Support | Boruta Ranking |
+----------------------+----------------+----------------+
|       history        |      True      |       1        |
|    used_discount     |      True      |       1        |
|      used_bogo       |      True      |       1        |
|     is_referral      |      True      |       1        |
|    offer_Discount    |      True      |       1        |
|       recency        |     False      |       2        |
|    zip_code_Rural    |     False      |       3        |
|    channel_Phone     |     False      |       4        |
|  zip_code_Surburban  |     False      |       5        |
|    zip_code_Urban    |     False      |       6        |
|     channel_Web      |     False      |       6        |
| channel_Multichannel |     False      |       8        |
+----------------------+----------------+----------------+
============================================
Selected Features by Boruta: Index(['history', 'used_discount', 'used_bogo', 'is_referral',
       'offer_Discount'],
      dtype='object')
```

We use the Boruta wrapper with the Random Forest Classifier and XGBoost Classifier to rank input variable importance.

History and Offer Discount are the top two variables that are selected by both the classifiers.

In [29]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

def train_logistic_regression(X_train, y_train, X_test, y_test):
    # Perform grid search to find the best hyperparameters for logistic regression
    param_grid = {'C': [0.1, 1, 2, 3], 'penalty': ['l1', 'l2']}
    lr = LogisticRegression(max_iter=10000)
    grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='roc_auc')
    grid_search.fit(X_train, y_train)
    print("Best parameters for logistic regression:", grid_search.best_params_)

    # Train a logistic regression model on the training data using the best hyperparameters
    lr = LogisticRegression(max_iter=10000, C=grid_search.best_params_['C'], penalty=grid_search.best_params_['penalty'], class
    lr.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = lr.predict(X_test)

    # Evaluate the model's performance using the AUC-ROC score, accuracy score and confusion matrix
    auc_roc = roc_auc_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    table = PrettyTable()
    table.field_names = [" ", "auc_roc", "accuracy", "conf_matrix"]
    table.add_row(["Logistic Regression", round(auc_roc,2), round(accuracy,2), conf_matrix])
    print(table)

    return lr
```

In [30]:

```
lr = train_logistic_regression(X_training, y_training, X_test, y_test)
```

```
Best parameters for logistic regression: {'C': 2, 'penalty': 'l2'}
+---------------------+---------+----------+---------------+
|                     | auc_roc | accuracy |  conf_matrix  |
+---------------------+---------+----------+---------------+
| Logistic Regression |   0.61  |   0.62   |  [[9036 5611] |
|                     |         |          |  [ 941 1458]] |
+---------------------+---------+----------+---------------+
```

In [31]:

```
y_proba = lr.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

In [32]:

```
y_proba = lr.predict_proba(X_test)
#convert to dataframe
y_proba_df = pd.DataFrame(y_proba)
#rename the columns
y_proba_df.columns = ['No Conversion', 'Conversion']
#add the actual conversion column
y_proba_df['Actual Conversion'] = y_test.values
#add the predicted conversion column
y_proba_df['Predicted Conversion'] = lr.predict(X_test)
#add the predicted probability of conversion
y_proba_df['Predicted Probability of Conversion'] = y_proba_df['Conversion']
y_proba_df.sort_values(by='Predicted Probability of Conversion', ascending=False, inplace=True)
y_proba_df.head(20)
```

Out[32]:

	No Conversion	Conversion	Actual Conversion	Predicted Conversion	Predicted Probability of Conversion
14812	0.1214	0.8786	0	1	0.8786
9188	0.1228	0.8772	1	1	0.8772
2347	0.1257	0.8743	1	1	0.8743
6196	0.1269	0.8731	0	1	0.8731
16978	0.1271	0.8729	1	1	0.8729
16848	0.1276	0.8724	1	1	0.8724
8600	0.1277	0.8723	0	1	0.8723
13117	0.1302	0.8698	1	1	0.8698
4912	0.1307	0.8693	0	1	0.8693
2148	0.1396	0.8604	1	1	0.8604
5241	0.1397	0.8603	0	1	0.8603
8079	0.1402	0.8598	1	1	0.8598
15934	0.1455	0.8545	0	1	0.8545
9115	0.1456	0.8544	1	1	0.8544
14505	0.1464	0.8536	0	1	0.8536
8089	0.1488	0.8512	0	1	0.8512
1477	0.1503	0.8497	1	1	0.8497
6014	0.1509	0.8491	1	1	0.8491
1643	0.1510	0.8490	0	1	0.8490
5963	0.1510	0.8490	1	1	0.8490

We choose the importance of our input variables based on Boruta's ranking and Forward Stepwise selection. Leading us to use all the explanatory variables
This was not really necessary since we had only 8 input variables, but we did it to show the process.

Then we use the Logistic Regression model to fit the data. We use logistic regression to to its superior performance in binary classification problems.

We performed hyperparameter tuning using GridSearchCV to find the best parameters for our model among the alpha and l1 (Lasso)/ l2 (Ridge) norms, and the best parameters were alpha = 2 and l2 norm.

One important aspect / parameter was to choose the class weights as balanced. This is because the dataset is imbalanced, with only 15% of the data being negative, leading to a lot of false positives.

d) For each record in the validation set, compute the uplift defined as

$$uplift(x) = P(conversion = 1|offer = Discount, x) - P(conversion = 1|offer = NoOffer, x)$$

If a campaign has the resources to target 25% of the customers, what uplift cutoff should be used? If we are to target 25% of the customers in the validation set based on the uplift obtained from your model, how much better would we do compared to random targeting?

In [33]:

```python
def get_uplift_metrics(X_test, y_test, lr):
    # Create a new dataframe X_val_discount with all the offer_Discount values set to 1
    X_val_discount = X_test.copy()
    X_val_discount['offer_Discount'] = 1

    # Create a new dataframe X_val_no_offer with all the offer_Discount values set to 0
    X_val_no_offer = X_test.copy()
    X_val_no_offer['offer_Discount'] = 0

    # Predict the probabilities of conversion for each offer type
    proba_discount = lr.predict_proba(X_val_discount)[:, 1]
    proba_no_offer = lr.predict_proba(X_val_no_offer)[:, 1]

    # Compute the uplift for each record in the validation set
    uplift = proba_discount - proba_no_offer

    # Determine the uplift cutoff to target 25% of the customers
    uplift_cutoff = np.percentile(uplift, 75)
    print("Uplift cutoff:", round(uplift_cutoff,3))

    #Create a dataframe with the uplift and the actual conversion
    uplift_df = pd.DataFrame({'uplift': uplift, 'conversion': y_test, 'Discount Offer': X_test['offer_Discount']})

    # Find the number of treatment = 1 in the top 25% of the X_test_uplift dataframe
    top_25_percent = uplift_df[uplift_df['uplift'] > uplift_cutoff]['conversion'].sum()
    print("Number of customers who buy the product when using the uplift model:", round(top_25_percent,0))

    # Find the number of treatment = 1 in the whole X_test_uplift dataframe
    total_25_perfect = uplift_df['conversion'].sum() /4
    print("Number of customers who buy the product when using random targeting:", round(total_25_perfect,0))

    # Compute the increase in the number of customers who buy the product when using the uplift model compared to random targe
    increase = top_25_percent- total_25_perfect

    print("Increase in the number of customers who buy the product when using the uplift model compared to random targeting:",

    return uplift_df.sort_values(by='uplift', ascending=False)

uplift_df = get_uplift_metrics(X_test, y_test, lr)
```

```
Uplift cutoff: 0.164
Number of customers who buy the product when using the uplift model: 646
Number of customers who buy the product when using random targeting: 600.0
Increase in the number of customers who buy the product when using the uplift model compared to random targeting:
46.0
```

In [34]:

```python
uplift_df.head(20)
```

Out[34]:

	uplift	conversion	Discount Offer
49298	0.1647	1	0
4517	0.1647	0	0
28204	0.1647	0	1
57225	0.1647	0	1
26631	0.1647	0	1
5655	0.1647	0	1
17480	0.1647	0	0
2120	0.1647	0	0
35094	0.1647	0	1
54040	0.1647	0	0
27034	0.1647	1	0
48492	0.1647	0	0
31753	0.1647	1	1
28393	0.1647	0	1
52913	0.1647	0	0
1734	0.1647	0	1
44743	0.1647	0	1
42040	0.1647	0	1
25952	0.1647	0	0
22488	0.1647	1	1

We see that targeting customers with an offer using training targets with our logistic regression model, we get an uplift in the number of conversions.

Though the predictive power is on the lower side using our logistic regression model, we get aroun 46 customer uplift compared to random targeting on the top 25% of the customers.

We know that by default the logistic regression model predicts the '0's, thus the confusion matrix presented above is flipped as for us the relevant metric to predit is a +1 conversion.