Modelling and Evaluation

In [1]:
```
# importing Necessary Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

import pickle
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import precision_recall_fscore_support, classification_report, roc_auc_score, confusion
```

Importing Cleaned and Transformed Data

In [2]:
```
df = pd.read_csv('../Data/data_ready.csv')
df.drop(df.columns[0], inplace = True, axis = 1)
df.head()
```

Out[2]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_base_bill_ele	forecast_cons	forecast_cons_12m	forecast_cons_year	forecast_discount_e
0	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
1	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
2	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
3	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
4	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	

5 rows × 487 columns

Splitting The Data into Train and Test Set

In [3]:
```
X = df.drop('churn', axis = 1)
y = df['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Preparing Evaluation Metrics

In this analysis, we'll be looking at some metrics that we'll use to evaluate how good our model is, we'll made a function so we can use it on different models without writting the code again.

In [4]:
```
def report(y_test, y_pred, model_name, model):

    print('The ROC AUC Score for', model_name, 'is: {:.2f}'.format(roc_auc_score(y_test, y_pred)))
    print('\nConfusion Matrix: ')
    conf_mat = pd.DataFrame(confusion_matrix(y_test, y_pred, labels = [0, 1]), columns = ['Predict 0', 'Pred
    print(conf_mat)

    precision, recall, fscore, support = precision_recall_fscore_support(y_test, y_pred)
    print('\nMetrics Reports')
    print(classification_report(y_test, y_pred))

    try:
        feature_impr = pd.DataFrame(data = model.feature_importances_*100,
                       index=df.drop('churn', axis = 1).columns.to_list()).rename(columns = {0: 'Feature Impor

        ax = feature_impr.iloc[0:11, :].plot(kind = 'barh', figsize=(6,6), xlim = [0, feature_impr.iloc[0:11
        plt.title('Top 10 Features for the ' + model_name  + ' Model')

        for i, v in enumerate(feature_impr.iloc[0:11, 0]):
            ax.text(v + 0.25, i + .25, str(round(v, 2)) + ' %', color='black', fontweight='bold')

        plt.gca().invert_yaxis()
        plt.show()
    except:
        print('')
```

Logistic Regression

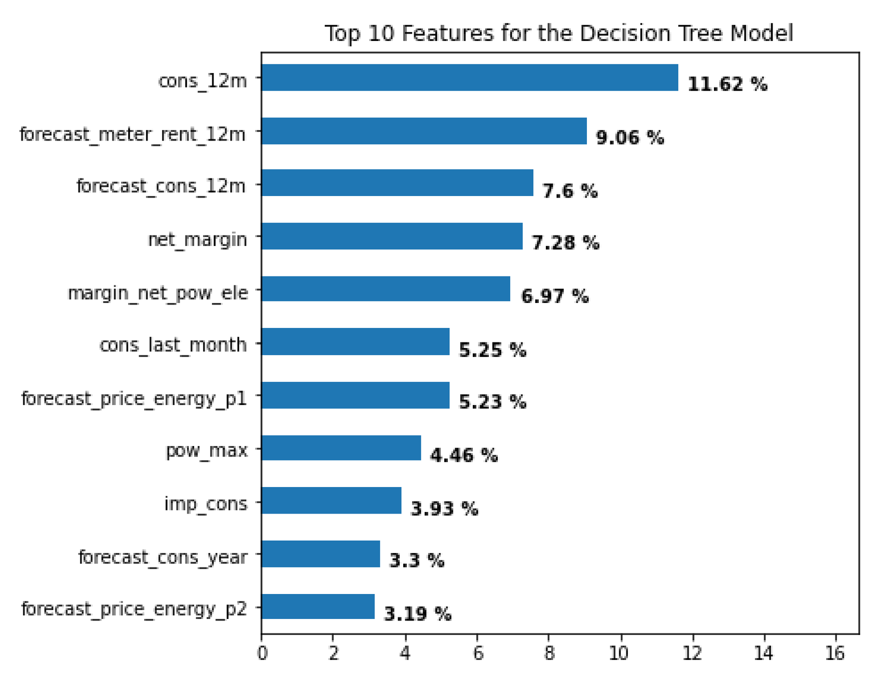
In [5]:
```
from sklearn.linear_model import LogisticRegression

clf_lr = LogisticRegression().fit(X_train, y_train)
lr_prediction = clf_lr.predict(X_test)

report(y_test, lr_prediction, 'Logistic Regression', clf_lr)
```

```
The ROC AUC Score for Logistic Regression is: 0.50

Confusion Matrix:
         Predict 0  Predict 1
Class 0      34800          0
Class 1       3801          0

Metrics Reports
              precision    recall  f1-score   support

           0       0.90      1.00      0.95     34800
           1       0.00      0.00      0.00      3801

    accuracy                           0.90     38601
   macro avg       0.45      0.50      0.47     38601
weighted avg       0.81      0.90      0.85     38601
```

Decision Tree

In [6]:
```
from sklearn.tree import DecisionTreeClassifier

clf_tree = DecisionTreeClassifier().fit(X_train, y_train)

tree_prediction = clf_tree.predict(X_test)

report(y_test, tree_prediction, 'Decision Tree', clf_tree)
```

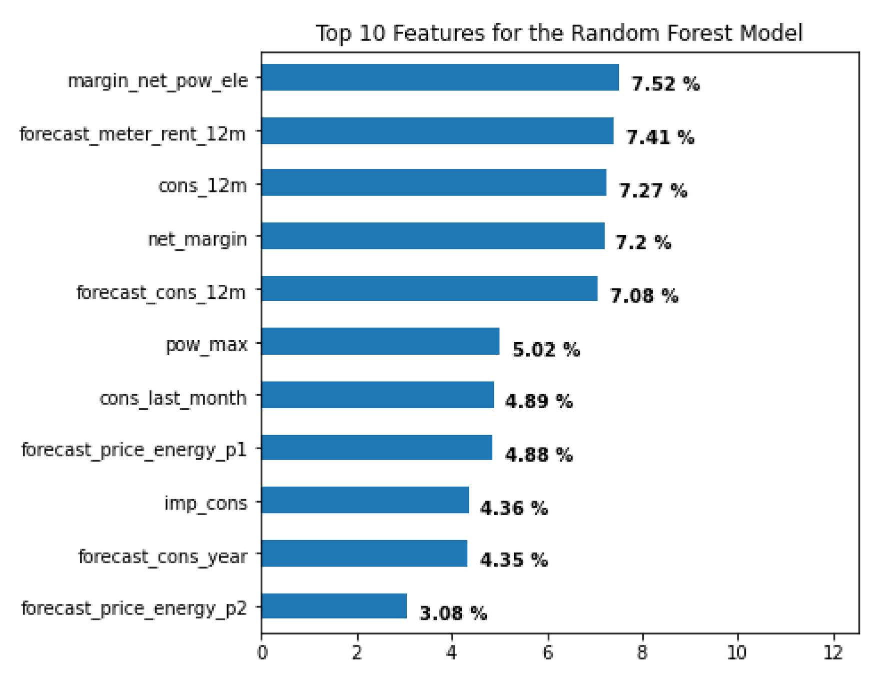
```
The ROC AUC Score for Decision Tree is: 1.00

Confusion Matrix:
         Predict 0  Predict 1
Class 0      34788         12
Class 1          8       3793

Metrics Reports
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     34800
           1       1.00      1.00      1.00      3801

    accuracy                           1.00     38601
   macro avg       1.00      1.00      1.00     38601
weighted avg       1.00      1.00      1.00     38601
```



Random Forest

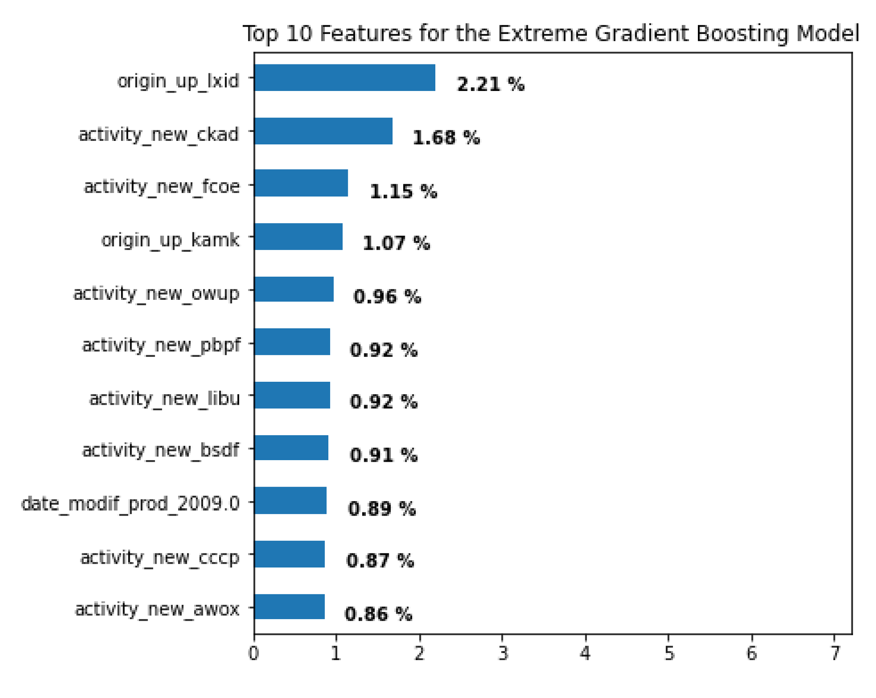
In [7]:
```
from sklearn.ensemble import RandomForestClassifier

clf_rf = RandomForestClassifier().fit(X_train, y_train)

rf_prediction = clf_rf.predict(X_test)

report(y_test, rf_prediction, 'Random Forest', clf_rf)
```

```
The ROC AUC Score for Random Forest is: 1.00

Confusion Matrix:
         Predict 0  Predict 1
Class 0      34794          6
Class 1         25       3776

Metrics Reports
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     34800
           1       1.00      0.99      1.00      3801

    accuracy                           1.00     38601
   macro avg       1.00      1.00      1.00     38601
weighted avg       1.00      1.00      1.00     38601
```



In [8]:
```
from xgboost import XGBClassifier

clf_xgb = XGBClassifier().fit(X_train, y_train, eval_metric = 'logloss')

xgb_prediction = clf_xgb.predict(X_test)

report(y_test, xgb_prediction, 'Extreme Gradient Boosting', clf_xgb)
```

```
The ROC AUC Score for Extreme Gradient Boosting is: 0.73

Confusion Matrix:
         Predict 0  Predict 1
Class 0      34792          8
Class 1       2066       1735

Metrics Reports
              precision    recall  f1-score   support

           0       0.94      1.00      0.97     34800
           1       1.00      0.46      0.63      3801

    accuracy                           0.95     38601
   macro avg       0.97      0.73      0.80     38601
weighted avg       0.95      0.95      0.94     38601
```



From our simple modelling process, we can already see that Random Forest Model can fit to the data pretty well. So in the next step, we shall see whether this model is robust enough by doing cross validation. (Cross Validation will fit the model on different bits of the data, then we'll see the average performance score of these models.)

Cross Validation

In [9]:
```
# Stratified K-fold Cross Validation

from sklearn.model_selection import KFold, cross_val_score, StratifiedKFold

kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
lst_accu_stratified = []

cv = KFold(n_splits=10, random_state=1, shuffle=True)

clf_lr = RandomForestClassifier()

# evaluate model
scores = cross_val_score(clf_lr, X, y, scoring='roc_auc', cv=cv, n_jobs=-1, )

# report performance
print('ROC AUC Score: Average of %.3f with standard deviation of %.3f' % (np.mean(scores), np.std(scores)))
```

```
ROC AUC Score: Average of 1.000 with standard deviation of 0.000
```

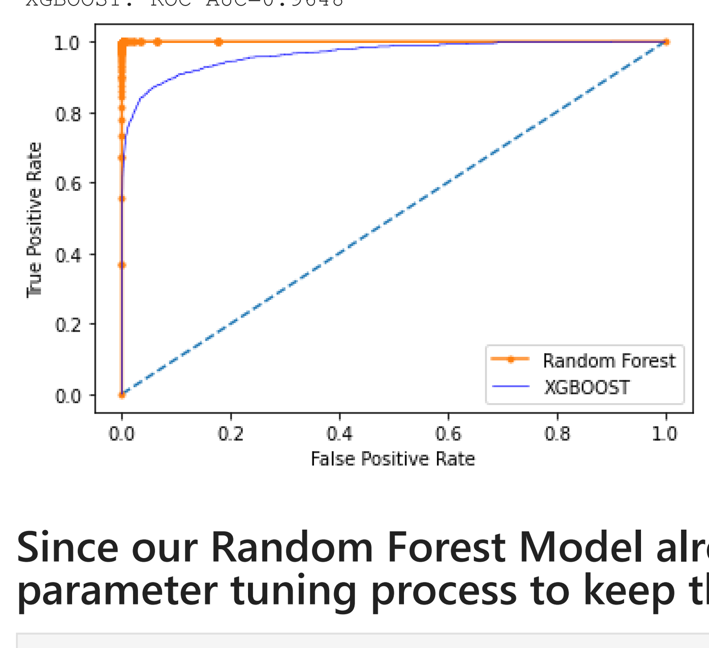
Based on this Cross Validation process, we get a really good classifier performance score. Note that this is a rather optimistic score, when we deploy this model into an unseen data, the performance could drop to a certain degree.

Receiver Operating Characteristic curve and Area Under Curve Plot

A ROC Curve will show the ability of a classifier to diagnose target variable, the AUC score will give us an idea of this classifier performance. A 0.5 AUC score means a classifier is just guessing with a 50:50 chance of getting the right answer,, a good AUC Score is the one that is closer to one.

In [10]:
```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot


ns_probs = [0 for _ in range(len(y_test))]

# predict probabilities
rf_probs = clf_rf.predict_proba(X_test)
xgb_probs = clf_xgb.predict_proba(X_test)

# keep probabilities for the positive outcome only
rf_probs = rf_probs[:, 1]
xgb_probs = xgb_probs[:, 1]

# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
rf_auc = roc_auc_score(y_test, rf_probs)
xgb_auc = roc_auc_score(y_test, xgb_probs)

# summarize scores
# print(': ROC AUC=%.3f' % (ns_auc))
print('Random Forest: ROC AUC=%.4f' % (rf_auc))
print('XGBOOST: ROC AUC=%.4f' % (xgb_auc))

# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, rf_probs)
xgb_fpr, xgb_tpr, _ = roc_curve(y_test, xgb_probs)

# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Random Forest')
pyplot.plot(xgb_fpr, xgb_tpr, color = 'blue', linewidth = 0.6, label='XGBOOST', )

# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')

# show the legend
pyplot.legend()
# show the plot
pyplot.show()
```

```
Random Forest: ROC AUC=1.0000
XGBOOST: ROC AUC=0.9648
```



Since our Random Forest Model already perform good, we won't need parameter tuning process to keep this analysis time effecient.

In [11]:
```
# Save Model

import joblib
filename = './random_forest_classifier.joblib.pkl'
_ = joblib.dump(clf_rf, filename, compress=9)
```

After our model is created, we can interpret it and get business values out of it in the next step of our workflow, which is the evaluation process.

Evaluation Process

In this document, we will discuss the use of the model that has just been built and interpret the results for the business.

In [1]:
```python
import pandas as pd
import joblib
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
```

In [2]:
```python
df = pd.read_csv('../Data/data_ready.csv')
df.drop(df.columns[0], axis = 1, inplace = True)
df.head(4)
```

Out[2]:

	cons_12m	cons_gas_12m	cons_last_month	forecast_base_bill_ele	forecast_cons	forecast_cons_12m	forecast_cons_year	forecast_discount_e
0	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
1	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
2	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	
3	4.310267	0.0	4.001128	335.807483	206.800605	3.179547	2.932604	

4 rows × 487 columns

In [3]:
```python
# Load Model

filename = './random_forest_classifier.joblib.pkl'
model_ = joblib.load(filename)
```

In [4]:
```python
df['predictions'] = model_.predict(df.drop('churn', axis = 1))
df['predictions_prob'] = model_.predict_proba(df.drop(['churn', 'predictions'], axis = 1))[:, 1]
```

In [5]:
```python
predictions = df[['churn', 'predictions', 'predictions_prob']]
```

Final Data Showcase

In [6]:
```python
df = pd.read_csv('../Data/cleaned_data.csv')
df.drop(df.columns[0], axis = 1, inplace = True)
```

In [7]:
```python
df = df.join(predictions.drop('churn', axis = 1))
df.to_csv('../Data/final_data.csv')
```

This is the final data that will be used for the evaluation process. It include the original information before feature engineering and predictions results from the Random Forest Model

Evaluation on Discount Policy

We will now evaluate whether giving a certain amount of discount will increase the company's profability.

There are several assumption needed for this analysis that will be mentioned to prevent bias judgement.

1. Since we already have a predictions probability on whether a certain customer will churn or not, we will choose a cutoff point to handle customer with predictions probability to churn bigger than our cutoff to receive discount policy.
2. After the customer was given the discount, we'll assume that certain customer won't churn, which is sometimes not the case.
3. To get the revenue received by the company, we calculate the energy consumption with the price added with the meter rent. The calculation will be using the forecasted value for 12 months. (Note: Since we don't know when will the customer churn during the year, we will use the average revenues lost based on the period of time between 1 January 2016 and the start of March 2016, with 100% lost revenue if customer churned from January and 83.9% lost if the customer churned at the end of February, #guided by the BCG Inside Sherpa Report)

The data used for the final analysis is the final showcase data.

In [8]:
```python
df = pd.read_csv('../Data/final_data.csv')
df.drop(df.columns[0], axis = 1, inplace = True)
```

1. Produce Baseline Revenue for Further Benchmarking Process

To calculate revenue, we need the forecasted consumption for 12 months times the price for the first period added with the forecasted meter rent for 12 months. This will be the baseline revenues for PowerCo if there are no customer that ended up churning during the time period.

In [9]:
```python
df['baseline_revenue'] = df['forecast_cons_12m'] * df['forecast_price_energy_p1'] + df['forecast_meter_rent_

print('The baseline revenues before accounting' \
      + ' for people churning is ${:.2f} millions'.format(df['baseline_revenue'].sum() / 10**6))
```

```
The baseline revenues before accounting for people churning is $71.69 millions
```

2. Calculate Baseline Revenue After churn

Since people that churned meant that the revenue from that specific customer will be lost, We should calculate the estimate revenues that PowerCo got after accounting for people churning.

In [10]:
```python
df['baseline_revenue_after_churn'] = df['baseline_revenue'] * (1 - 0.919 * df['churn'])
```

In [11]:
```python
print('The baseline revenues after accounting' \
      + ' for people churning is ${:.2f} millions'.format(df['baseline_revenue_after_churn'].sum() / 10**6))
```

```
The baseline revenues after accounting for people churning is $64.85 millions
```

3. Calculate the difference for the revenue after the discount has been given

First, create another data that gathers information about revenue generated from the customer, we can use the baseline revenue after churning for this part.

Given a prediction probabilites higher than a cutoff point, PowerCo will give a discount policy reducing the revenues that is generated from that specific customer, but preventing them from churning. Then we can calculate the total revenues and compare it to the revenue if discount policy isn't given.

In [12]:
```python
def diff_revenue(data = df, cutoff = 0.6, disc = 0.25):

    df['disc_rev'] = df['baseline_revenue_after_churn']

    # locate user with churn probabilities higher than cutoff.
    df.loc[df['predictions_prob'] >= cutoff, 'disc_rev'] = df['baseline_revenue'] * (1- disc)

    # Return the difference before and after discount is imposed
    return df['disc_rev'].sum() - df['baseline_revenue_after_churn'].sum()

print('Difference between revenues before and after giving discount to consumer' \
      + ' is ${:.2f} millions'.format(diff_revenue()/10**6))
```

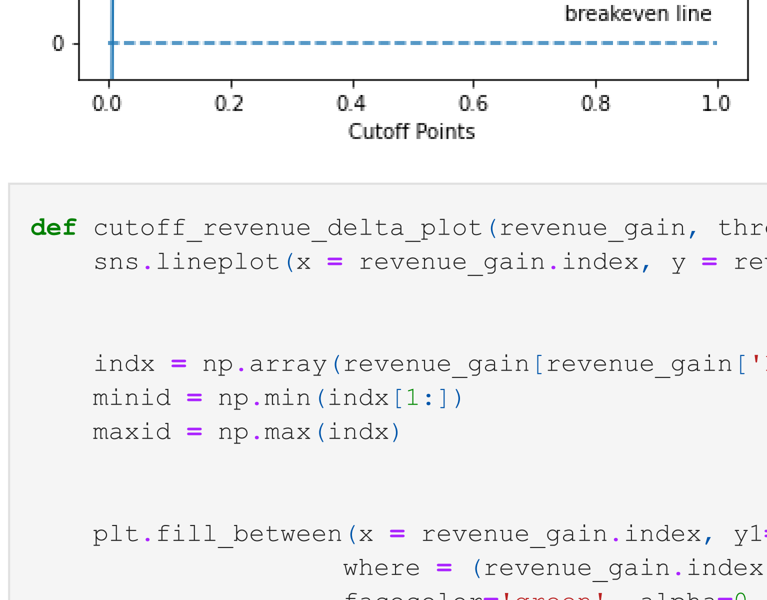
```
Difference between revenues before and after giving discount to consumer is $4.94 millions
```

Since cutoff can affect this analysis, we can plot a graph that showed the relationship between revenue differences and cutoff points.

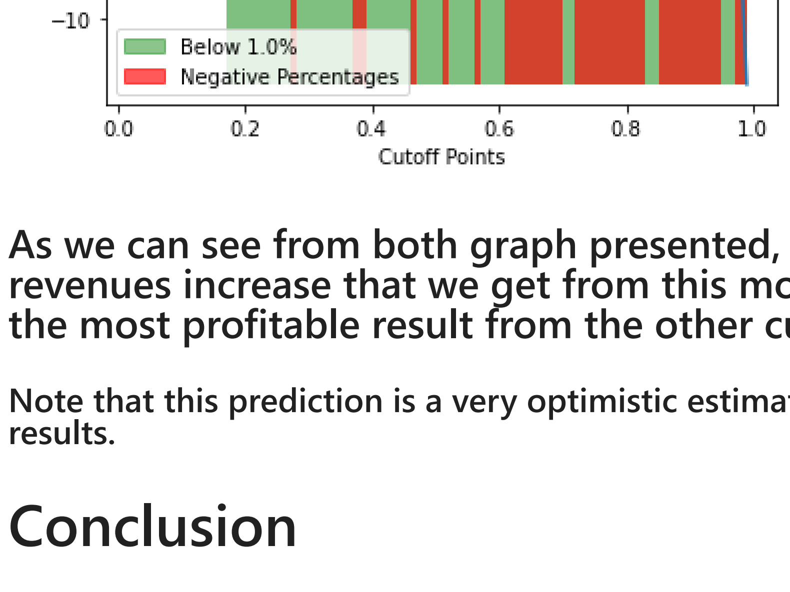
In [13]:
```python
revenue_gain = pd.DataFrame([diff_revenue(df, cutoff = x) for x in np.arange(0, 1, 0.01)],
                            index = np.arange(0, 1, 0.01), columns = ['Revenue Gain']) / 10**6
revenue_gain['Different Compared to Previous Cutoff'] = 0
for index in range(1, len(revenue_gain)):
    rev_normalized = revenue_gain['Revenue Gain'] + np.min(revenue_gain['Revenue Gain'])
    increase = 100*((revenue_gain.iloc[index, 0] - revenue_gain.iloc[index-1, 0])/revenue_gain.iloc[index, 0
    revenue_gain.iloc[index, 1] = increase
```

In [14]:
```python
def plot_rev_increase(revenue_gain):
    sns.lineplot(x = revenue_gain.index, y = revenue_gain['Revenue Gain'])
    plt.scatter(revenue_gain['Revenue Gain'].idxmax(), np.max(revenue_gain['Revenue Gain']), s= 50, color =
    plt.ylim(-.5, np.max(revenue_gain['Revenue Gain']) + 0.5)
    plt.ylabel('Million dollars')
    plt.xlabel('Cutoff Points')
    plt.hlines(0, 0, 1, linestyles = 'dashed')

    plt.annotate(text = '{:.2f} Millions'.format(np.max(revenue_gain['Revenue Gain'])),
                 xy = (revenue_gain['Revenue Gain'].idxmax(),
                 np.max(revenue_gain['Revenue Gain']) - 0.4))

    plt.annotate(text = 'breakeven line',xy = (0.75, 0.3))

    plt.title('Revenue Gained Relative to\nThe Cutoff Points for Giving Discounts.');

    print('The best cutoff point with the highest reward is {}'.format(revenue_gain['Revenue Gain'].idxmax()
plot_rev_increase(revenue_gain)
```

```
The best cutoff point with the highest reward is 0.25
```



In [15]:
```python
def cutoff_revenue_delta_plot(revenue_gain, thres):
    sns.lineplot(x = revenue_gain.index, y = revenue_gain['Different Compared to Previous Cutoff'], )


    indx = np.array(revenue_gain[revenue_gain['Different Compared to Previous Cutoff'] <= thres].index)
    minid = np.min(indx[1:])
    maxid = np.max(indx)


    plt.fill_between(x = revenue_gain.index, y1= np.min(revenue_gain.iloc[:, 1]), y2 = np.max(revenue_gain.i
                     where = (revenue_gain.index >= minid) & (revenue_gain.index <= maxid),
                     facecolor='green', alpha=0.5)

    plt.fill_between(x = revenue_gain.index, y1= np.min(revenue_gain.iloc[:, 1]), y2 = np.max(revenue_gain.i
                     where = revenue_gain['Different Compared to Previous Cutoff'] < 0,
                     facecolor='red', alpha=0.65)

    plt.title('Revenue Increase(%) Relative to\nThe Cutoff Points for Giving Discounts.')
    plt.xlabel('Cutoff Points')

    plt.ylabel('Revenue Increase(%)')


    legend_elements = [Patch(facecolor='green', edgecolor='green', alpha = 0.45,
                             label='Below {}%'.format(thres*100)),
                       Patch(facecolor='red', edgecolor='r', alpha = 0.65,
                             label='Negative Percentages')]

    plt.legend(handles=legend_elements, loc='lower left')
    plt.show()

cutoff_revenue_delta_plot(revenue_gain.iloc[3:, :], 0.01)
```



As we can see from both graph presented, selecting cutoff can affect the total amount of revenues increase that we get from this model, We will choosef 0.32 as the cutoff as it gave the most profitable result from the other cutoff points.

Note that this prediction is a very optimistic estimate. The deployment on real world case can yield various results.

Conclusion

Based on the analysis that we did, we can conclude that imposing discount for customer that have the probability to churn higher than 0.32 can increase our total revenues by an estimate of 4.97 Million Dollars.