Assignment 2

Bernardo Carvalho

In this assignment, we will explore the dataset Tahoe's healthcare case. The case is about the costs incurred by hospital on the readmission of patients, imposed by the Obama care act. Ever since this program started, hospitals started being penalized in their costs refunding by the goverment on the readmission of patients. Motivated by that, Tahoe launched a program to try to reduce the number of readmited patients. However, this program has a cost of implementation. In this assignment, we will investigate if the use of a machine learning model can reduce target the patients with the highest likelihood of readmission for the application of the program.

We start by importing the necessary packages and datasets:

In [1]:
```
# Importing necessary packages

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

# Importing the full dataset
data = pd.read_csv("Tahoe_Healthcare_Data.csv")
```

(i) Estimating loss in reimbursement

The reasearch estimates the loss per readmited patient as being USD 8000. Therefore, to get to the total estimated loss we multiply the number of readmited patients by that value. We can identify the readmited patients by column "readmit30" of the dataset. Hence:

In [2]:
```
readmit_patients = data['readmit30'].sum()
loss_per_readmit = 8000

print("The total loss due to readmission is: USD {}".format(readmit_patients*loss_p
```

```
The total loss due to readmission is: USD 7984000
```

(ii) Estimating the net change in cost if Caretracker is applied to all patients

If we apply CareTracker to all patients, we would incur the cost of the program to all patients, and only reduce the readmission of 40%, according to the estimation of the article. Therefore:

In [3]:
```
total_patients = len(data)
readmit_patients = data['readmit30'].sum()
cost_program = 1200
benefit_program = 8000

net_benefit = readmit_patients*.4*benefit_program - total_patients*cost_program
```

```
print("The net benefit is USD {}".format(round(net_benefit,2)))
```

```
The net benefit is USD -2064800.0
```

This means, even though the reduction on the readmission rate looks promising, if we don't target its application the program becomes loss making

(iii) Estimating upper bound for cost savings

Assuming 100% accuracy in the predictions, that is, if we could perfectly predict who would be readmited, we would only need to apply the program to those patients, and therefore only incur the costs there.

In [4]:
```
# Calculating upper bound

upper_bound = readmit_patients*.4*benefit_program - readmit_patients*cost_program
print("The upper bound is USD {}".format(round(upper_bound,2)))
```

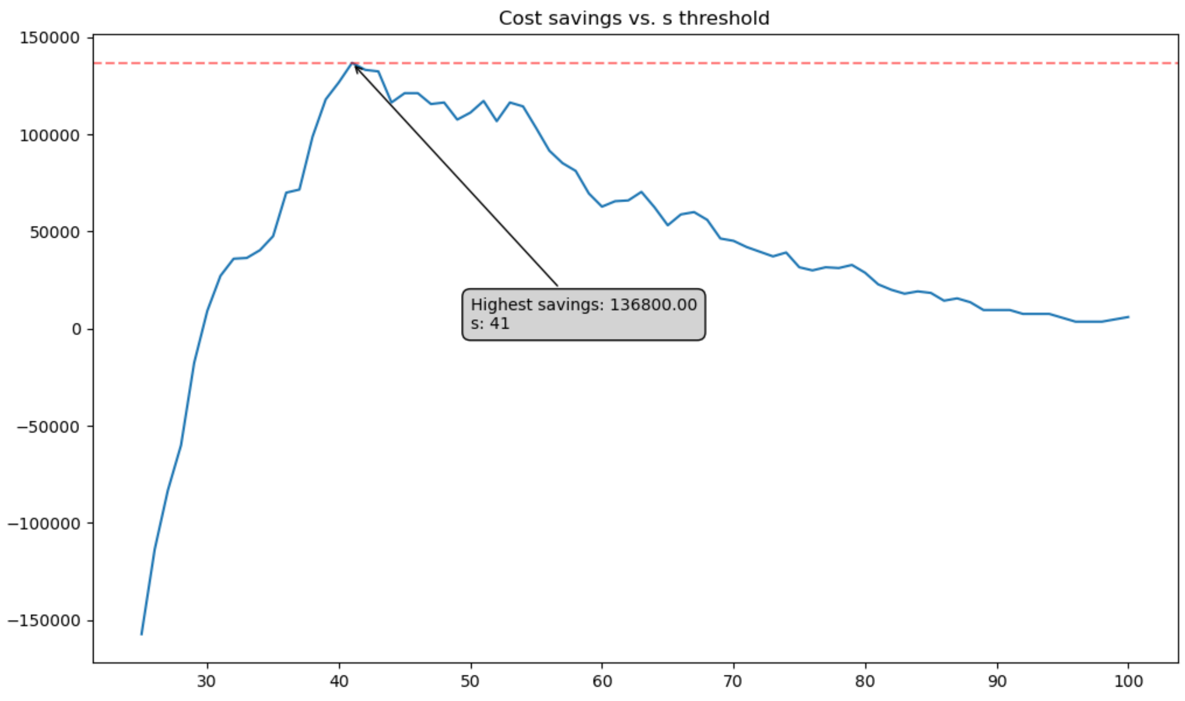
```
The upper bound is USD 1996000.0
```

(iv) Algorithm based only on the severity score

Now we create an algorithm that is solely based on the severity score of each observation. We notice that the severity score is on average much higher for the readmited patients then it is for the rest.

In [13]:
```
# Importing sklearn's accuracy score
from sklearn.metrics import accuracy_score, recall_score, precision_score

# Initialize empty lists
accuracy = []
cost_savings = []
def savings(s):
    # Get patients that have the severity score above the threshold
    n_patients_thresh = sum(data['severity score'] > s)
    # Get all patients for that score that were readmited
    n_patients_readmitted = sum((data['readmit30'] == 1) & (data['severity score']
    # Assume that 40% of them would not need to be readmited, then calculate the sa
    cost_saving = n_patients_readmitted*8000*.4 - n_patients_thresh*1200
    # As a plus, calulate the accuracy of the model for each s.
    prediction = [1 if x > s else 0 for x in data['severity score'].values]
    return prediction, cost_saving

for s in range(25,101):
    prediction, cost_saving = savings(s)
    cost_savings.append(cost_saving)
    accuracy.append(accuracy_score(prediction, data['readmit30']))
```

In [14]:
```
fig = plt.figure(figsize=(12,7))

best_s = cost_savings.index(max(cost_savings))+25
max_savings = cost_savings[cost_savings.index(max(cost_savings))]

plt.plot(range(25,101), cost_savings, label='Cost savings')

plt.title("Cost savings vs. s threshold")
plt.axhline(y=max_savings, color='r', alpha =0.5, linestyle='--', label='Highest Ac

# Annotate the point with an arrow and a text box
plt.annotate(f'Highest savings: {max_savings:.2f}\ns: {best_s}',
             xy=(best_s, max_savings),
             xytext=(50, 0),  # Adjust the position of the text box
             arrowprops=dict(arrowstyle='->'),
             bbox=dict(boxstyle='round,pad=0.5', facecolor='lightgrey', edgecolor='


plt.show()
```

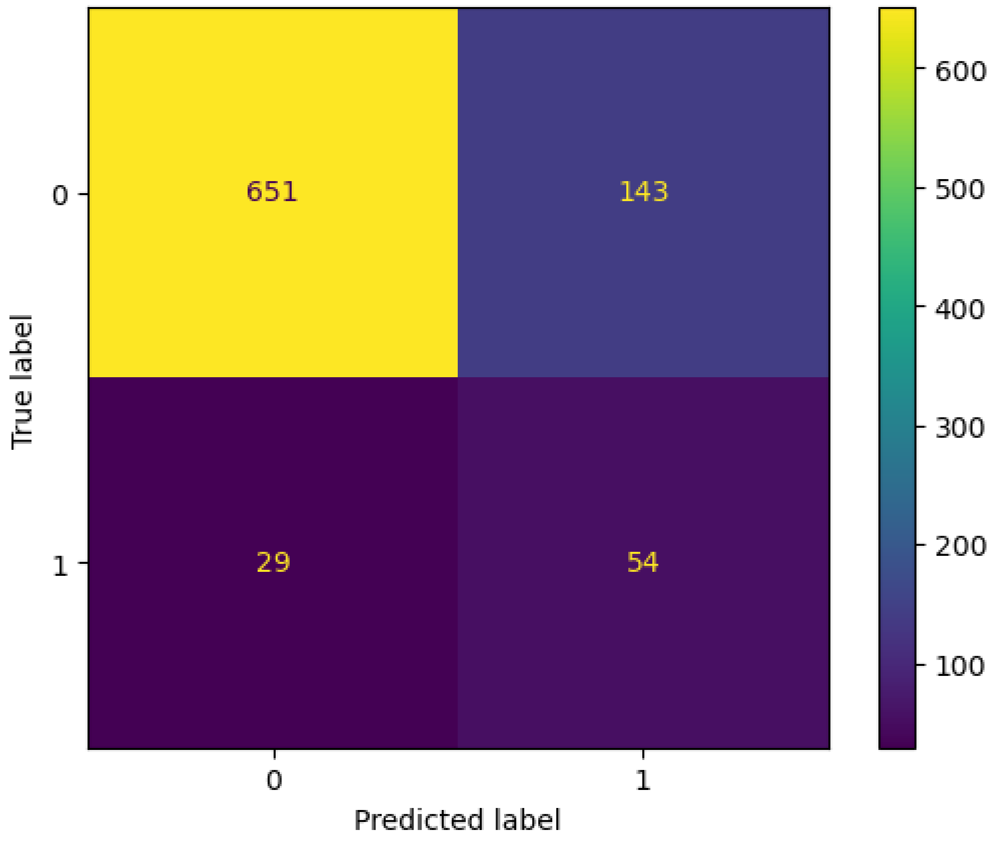


(v) Train a classfication algorithm

Finally, we will use the dataset to train a Logistic regression algorithm and fit the data. Using sklearn, the results are:

In [31]:
```
# Import necessary packages
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Preparing training data
X_train = np.array(train_data)[:,:-1]
y_train = np.array(train_data)[:,-1]

# Preparing testing data
X_test = np.array(test_data)[:,:-1]
y_test = np.array(test_data)[:,-1]

# Initialize the algorithm
clf = LogisticRegression()

# Fit model and predict outputs
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Get measures of interest
```

```
accuracy = accuracy_score(y_pred, y_test)
precision = precision_score(y_pred, y_test)
c_matrix = confusion_matrix(y_pred, y_test)

# Display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=c_matrix, display_labels=clf.classes
disp.plot()
plt.show()
```



In [51]:
```
# Calculating savings per person
precision*.4*(8000-1200)
```

Out[51]:
```
745.5837563451777
```

Note that, we are particularly interested in the prediction of the model, since the false negatives don't impose a high risk to us. Therefore, if we assume the test data precision to hold for the new data, we will get an unitary savings per person who receives the the CareTracker as:

$$Cost_{unit} = precision * .4 * (8000 - 1200) = 745.58$$