```
# import the dataset
import pandas as pd
import numpy as np
master = pd.read_csv('cars.csv')

# Discard any categorical columns (can't easily be quantified)
master.drop(columns=['Engine Information.Engine Type', 'Identification.ID', 'Identification.Model Year'],
axis=1, inplace=True)

master.shape
```

```
(5076, 15)
```

```
# Add features to the data - is the car efficient or not?

# Rename select keys more readable column names

df = master.copy()

df = df.rename(columns=
                       {
                            "Fuel Information.City mpg": "cityMpg",
                            "Fuel Information.Highway mpg": "highwayMpg",
                            "Dimensions.Height": "height",
                            "Dimensions.Length": "length",
                            "Dimensions.Width": "width",
                            "Engine Information.Engine Statistics.Horsepower": "horsepower",
                           "Engine Information.Hybrid": "isHybrid",
                           "Engine Information.Number of Forward Gears": "numberGears"
                       }
                      )

# Determine whether efficient or not
df['isCityEfficient'] = df.apply(lambda row: (row['cityMpg'] >= 22), axis=1)
df['isHighwayEfficient'] = df.apply(lambda row: (row['highwayMpg'] >= 22), axis=1)
df['isCombinedEfficient'] = df.apply(lambda row: (row['isCityEfficient'] & row['isHighwayEfficient']) ,
axis=1)

df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	height	length	width	Engine Information.Driveline	isHybrid	numberGears	Engine Information.Transmission	cityMpg	Information.
5071	13	253	201	Front-wheel drive	True	5	5 Speed Automatic	18	Gasoline
5072	141	249	108	All-wheel drive	True	6	6 Speed Manual	12	Gasoline
5073	160	249	108	All-wheel drive	True	6	6 Speed Manual	12	Gasoline
5074	200	210	110	Rear-wheel drive	True	6	6 Speed Automatic Select Shift	17	Gasoline
5075	200	94	110	Rear-wheel drive	True	6	6 Speed Automatic Select Shift	17	Gasoline

```
# Determine features to include in dataset

# Remove any unneeded columns
df = df.drop([
    'Engine Information.Driveline',
    'Engine Information.Transmission',
    'Fuel Information.Fuel Type',
    'Identification.Classification',
    'Identification.Make',
    'Identification.Year',
    'Engine Information.Engine Statistics.Torque'
    ], axis=1)

dataset = df.copy()

dataset.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	height	length	width	isHybrid	numberGears	cityMpg	highwayMpg	horsepower	isCityEfficient	isHighwayEfficient	isCombinedEfficient
5071	13	253	201	True	5	18	25	250	False	True	False
5072	141	249	108	True	6	12	20	552	False	False	False
5073	160	249	108	True	6	12	20	552	False	False	False
5074	200	210	110	True	6	17	25	315	False	True	False
5075	200	94	110	True	6	17	25	315	False	True	False

```
# Split dataset into training and test set
from sklearn.model_selection import train_test_split
```

```
# Define our data and if they are efficient or not
# Train based of both mpg ratings, horsepower, and number of gears
data = dataset[['cityMpg', 'highwayMpg', 'horsepower', 'numberGears']].to_numpy()

labels = dataset['isCombinedEfficient'].to_numpy()
```

```
X_train, X_test, y_train, y_test = train_test_split(
    data,
    labels,
    test_size=0.2,
    random_state=42
)
```

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state = 21)

sgd_clf.fit(X_train, y_train)
```

```
SGDClassifier(random_state=21)
```

```
_mpg = data[0]
sgd_clf.predict([_mpg])
```

```
array([False])
```

```
# Run model on the test data
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring='accuracy')
```

```
array([0.91358936, 0.93643755, 0.93939394])
```

```
# Evaluate model and make apporpriate changes
from sklearn.model_selection import cross_val_predict

labels_train_predict = cross_val_predict(sgd_clf, X_train, y_train, cv=3)

labels_train_predict
```

```
array([ True, False, False, ..., False, False,  True])
```

```
# Precision/Recall?

from sklearn.metrics import precision_score, recall_score
precision_score(y_train, labels_train_predict)
```

```
0.9672897196261683
```

```
recall_score(y_train, labels_train_predict)
```

```
0.6043795620437956
```

```
from sklearn.metrics import f1_score
f1_score(y_train, labels_train_predict)
```

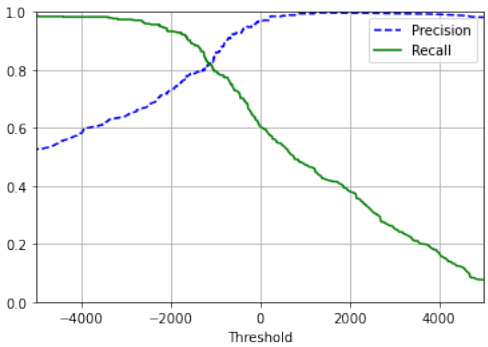
```
0.7439353099730458
```

```
# Decide on the threshold
labelsScores = cross_val_predict(sgd_clf, X_train, y_train, cv=3, method='decision_function')

labelsScores
```

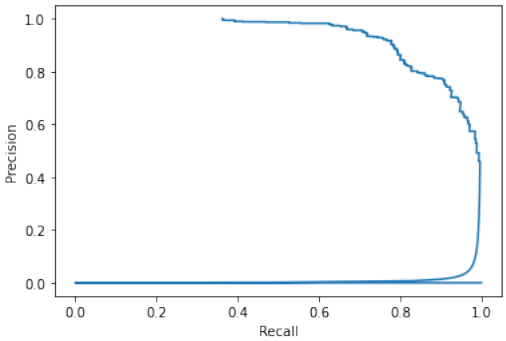
```
array([   768.23375126,  -4511.96179359, -15301.68876479, ...,
        -5744.85449862,  -6381.12049478,   3534.45478393])
```

```
# We can use theses cores with precision/recall to plot what the results would be with different thresholds
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train, labelsScores)
```

```
# Plot

import matplotlib as mpl
import matplotlib.pyplot as plt

def plotPrecisionRecallThreshold(pre, rec, thr):
    plt.plot(thr, pre[:-1], 'b--', label='Precision')
    plt.plot(thr, rec[:-1], 'g-', label='Recall')
    plt.grid()
    plt.xlabel('Threshold')
    plt.legend()
    plt.axis([-5000,5000,0,1])

plotPrecisionRecallThreshold(precisions, recalls, thresholds)
plt.show()
```




```
plt.plot(precisions, recalls)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```




```
# Want 91% precision
threshold91Precision = thresholds[np.argmax(precisions>=.91)]
threshold91Precision
```

```
-663.1249717147463
```

```
labelsTrainPred91 = (labelsScores >= threshold91Precision)
labelsTrainPred91
```

```
array([ True, False, False, ..., False, False,  True])
```

```
precision_score(y_train, labelsTrainPred91)
```

```
0.91005291005291
```

```
recall_score(y_train, labelsTrainPred91)
```

```
0.7532846715328467
```

```

```