Mike Rocchio

MSDS 422

04/25/2021

Assignment 3

General Note – I choose to do my assignment in DataBricks community edition because it has free cloud computing and storage and I am used to the environment from work. So, a few commands in the notebook may be unfamiliar.

Data preparation, exploration, visualization – This was a simpler dataset when it came to data wrangling but I found that the weight of the values is very lopsided and that certain features had much higher weights than others.

Review research design and modeling methods – I need to do a lot more reading on the classification methods, specifically Naïve Bayes. I had issues correctly weighing the features in the native bayes and that is why it was so off. I did not have many issues when doing logistic regression and am sure that I could get it to be more accurate. As for the NB method I was unsure on how to accurately tune my weighting.

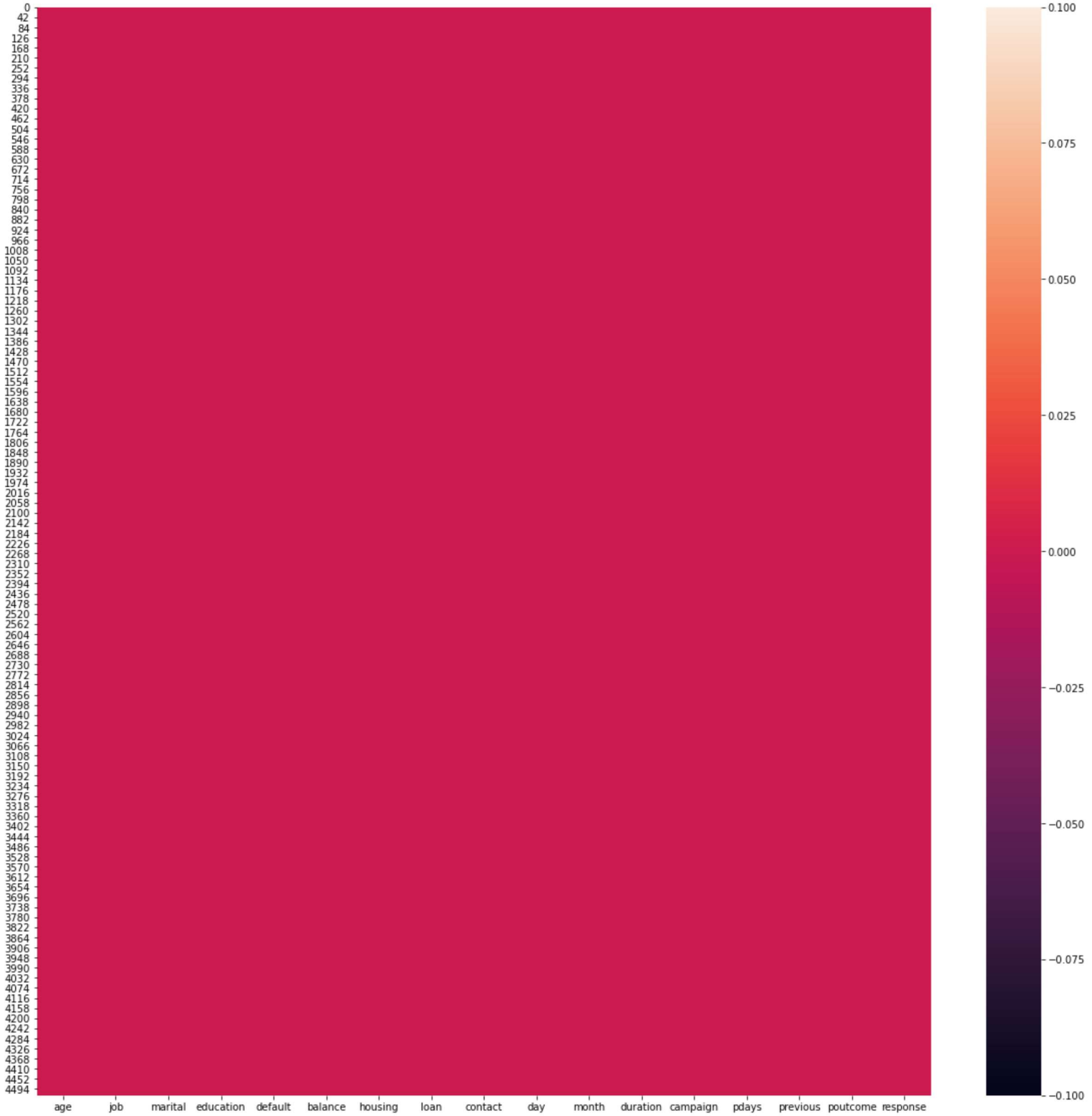
Review results, evaluate models – I would say that I did a decent job in feature engineering the datasets, it is a fickle are to do so and I am definitely a beginner. I also was a bit confused on how to do the ROC curve on the Naïve Bayes model.

Regarding the management problem, imagine that you are advising the bank about machine learning methods to guide telephone marketing campaigns. Which of the two

modeling methods would you recommend and why? And, given the results of your research, which group of banking clients appears to be the best target for direct marketing efforts (similar to those used with previous telephone campaigns)?

I would say that due to the unbalanced dataset that Naïve Bayes would be the best but also the most difficult method to utilize. I believe that done by a seasoned expert that the method would be have an efficacy in the high 90s.

```
from pyspark.sql import SparkSession
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import seaborn as sns
from sklearn.preprocessing import scale
import sklearn.linear_model as skl_lm
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
import statsmodels.formula.api as smf

bank = spark.read.option("delimiter",
";").format("csv").load("dbfs:/FileStore/shared_uploads/MichaelRocchio2023@u.northwestern.edu/bank
.csv", header=True)
# spark.sql("""Create Database ML_Assignment3""")
bank.write.saveAsTable('ML_Assignment3.bank', format='parquet', mode='overwrite')

bank=spark.sql("""
Select *
From ML_Assignment3.bank""").toPandas()
bank.dropna()

plt.figure(figsize=(20, 20))
sns.heatmap(bank.isnull())
plt.show()
```


0
42
84
126
168
210
252
294
336
378
420
462
504
546
588
630
672
714
756
798
840
882
924
966
1008
1050
1092
1134
1176
1218
1260
1302
1344
1386
1428
1470
1512
1554
1596
1638
1680
1722
1764
1806
1848
1890
1932
1974
2016
2058
2100
2142
2184
2226
2268
2310
2352
2394
2436
2478
2520
2562
2604
2646
2688
2730
2772
2814
2856
2898
2940
2982
3024
3066
3108
3150
3192
3234
3276
3318
3360
3402
3444
3486
3528
3570
3612
3654
3696
3738
3780
3822
3864
3906
3948
3990
4032
4074
4116
4158
4200
4242
4284
4326
4368
4410
4452
4494
age
job
marital
education
default
balance
housing
loan
contact
day
month
duration
campaign
pdays
previous
poutcome
response
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

```
#my code for binary conversion is cooler than yours professor :)

binary_fields=['default', 'housing', 'loan', 'response']

bank_train=bank

# del bank_train['day']

# del bank_train['pdays']

# del bank_train['duration']

for i in binary_fields:
  bank_train[i]=bank_train[i].apply(
    lambda x:1 if x=='yes' else 0 if x=='no' else x)

# bank_binary_x=bank_binary[['default', 'housing', 'loan']]
# bank_binary_y=bank_binary['response']

bank_train_cat = bank_train[['education', 'poutcome']]

bank_train_ncat=bank_train[['age', 'default', 'balance', 'housing', 'loan', 'campaign']]

bank_train_cat = pd.get_dummies(bank_train_cat)

y = bank_train['response'].to_numpy()

bank_train_fin=pd.concat([bank_train_ncat, bank_train_cat], axis=1)

bank_train_fin['age']=pd.to_numeric(bank_train_fin['age'])
age_mean = bank_train_fin['age'].mean()
age_std = bank_train_fin['age'].std()
age_useable=(bank_train_fin['age'].to_numpy() - age_mean) / age_std

bank_train_fin['balance']=pd.to_numeric(bank_train_fin['balance'])
balance_mean = bank_train_fin['balance'].mean()
balance_std = bank_train_fin['balance'].std()
balance_useable=(bank_train_fin['balance'].to_numpy() - balance_mean) / balance_std

bank_train_fin['campaign']=pd.to_numeric(bank_train_fin['campaign'])
campaign_mean = bank_train_fin['campaign'].mean()
campaign_std = bank_train_fin['campaign'].std()
campaign_useable=(bank_train_fin['campaign'].to_numpy() - campaign_mean) / campaign_std



bank_train_fin['age']=age_useable

bank_train_fin['balance']=balance_useable

bank_train_fin['campaign']=campaign_useable

X=bank_train_fin.to_numpy()
y = bank_train['response'].to_numpy()
```

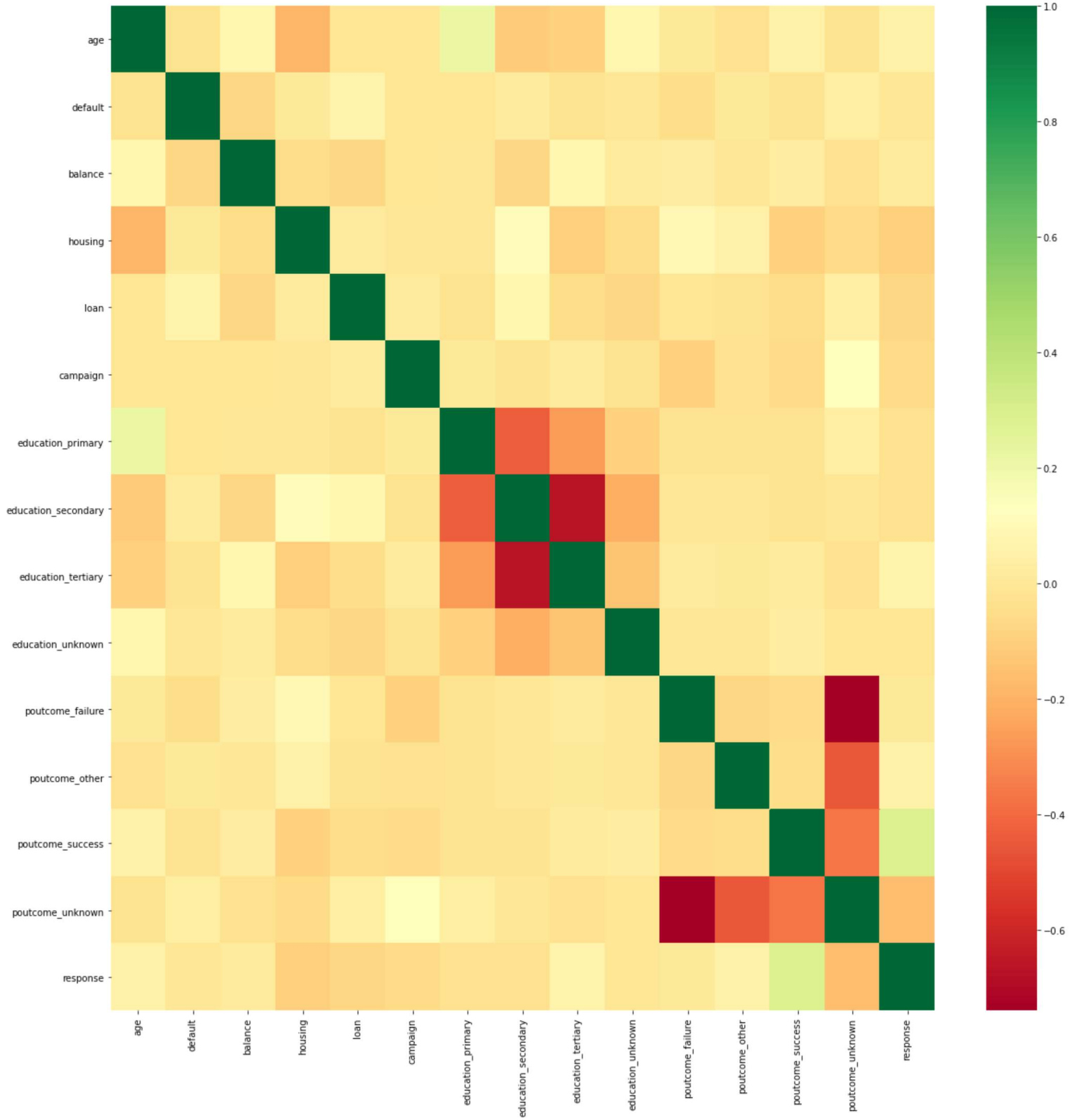
```
bank_train_fin
```

Out[305]:

	age	default	balance	housing	loan	campaign	education_primary	education_secondary	education_tertiary
0	-1.056153	0	0.121058	0	0	-0.576766	1	0	0
1	-0.772497	0	1.118521	1	1	-0.576766	0	1	0
2	-0.583394	0	-0.024142	1	0	-0.576766	0	0	1
3	-1.056153	0	0.017724	1	1	0.387925	0	0	1
4	1.685850	0	-0.472701	1	0	-0.576766	0	1	0
...	...	...	...	...	...	...	...	...	...
4516	-0.772497	0	-0.583345	1	0	0.709488	0	1	0
4517	1.496746	1	-1.573497	1	1	-0.576766	0	0	1
4518	1.496746	0	-0.374682	0	0	2.638868	0	1	0
4519	-1.245256	0	-0.094914	0	0	0.387925	0	1	0
4520	0.267573	0	-0.095247	1	1	-0.255202	0	0	1

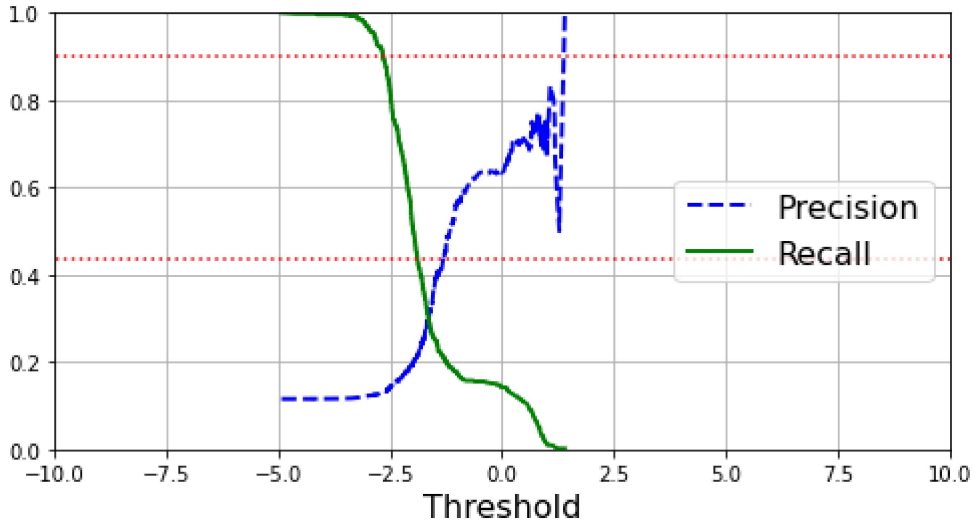
4521 rows × 14 columns

```
print(y.shape)
print(X.shape)

(4521,)
(4521, 14)

bank_train_fin1=bank_train_fin
bank_train_fin1['response']=y
plt.figure(figsize=(20, 20))
g = sns.heatmap(round(bank_train_fin1.corr(),2),annot=False,cmap="RdYlGn")
```

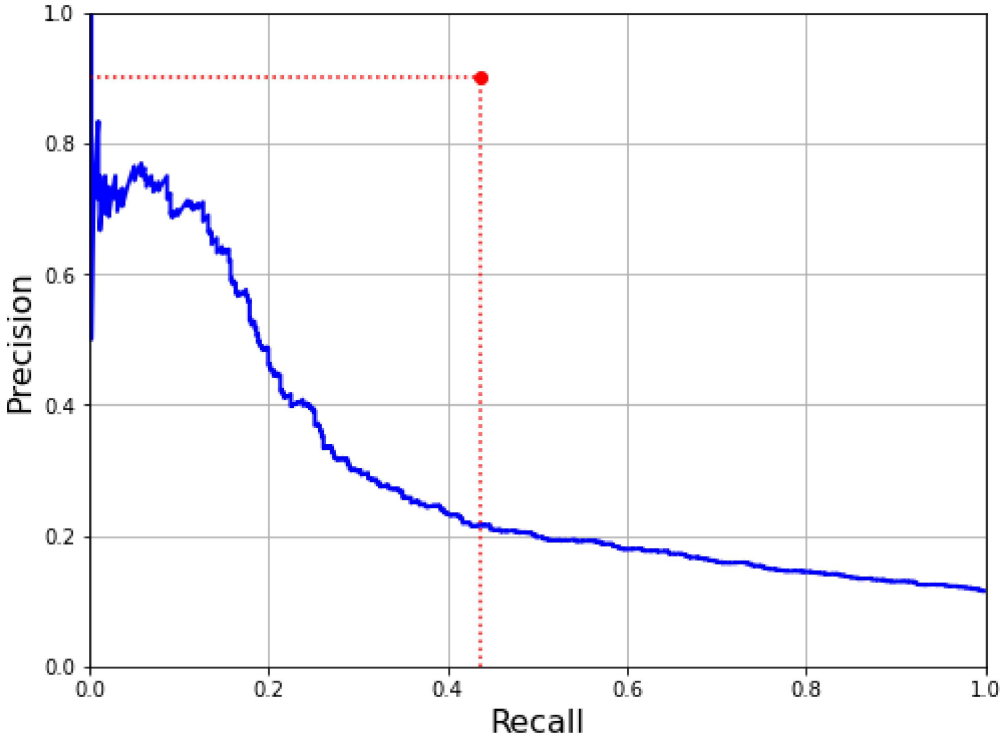



```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

lgmodel = LogisticRegression(random_state=0).fit(X, y)
lgmodel.predict(X)

gnb = GaussianNB().fit(X, y)
y_pred = gnb.predict(X)
```

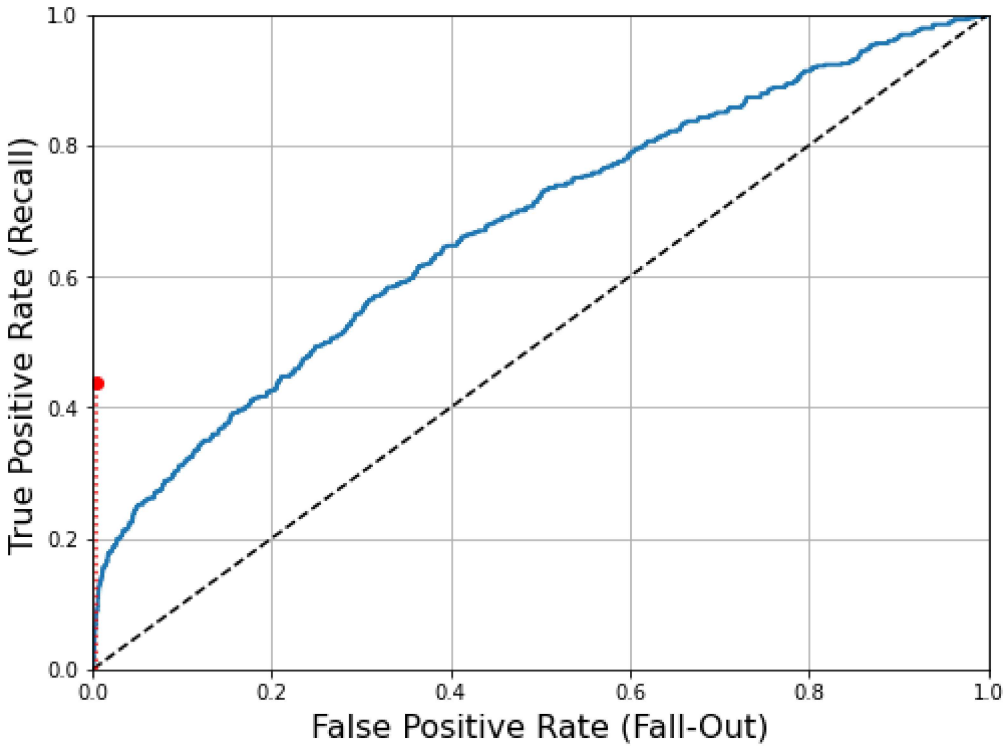
```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_recall_curve

# sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
# sgd_clf.fit(X, y)

y_scores = cross_val_predict(lgmodel, X, y, cv=5, method="decision_function")

precisions, recalls, thresholds = precision_recall_curve(y, y_scores)

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
    plt.legend(loc="center right", fontsize=16)
    plt.xlabel("Threshold", fontsize=16)
    plt.grid(True)
    plt.axis([-10, 10, 0, 1])

plt.figure(figsize=(8, 4))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.plot([7813, 7813], [0., 0.9], "r:")
plt.plot([-50000, 7813], [0.9, 0.9], "r:")
plt.plot([-50000, 7813], [0.4368, 0.4368], "r:")
plt.plot([7813], [0.9], "ro")
plt.plot([7813], [0.4368], "ro")
plt.show()
```



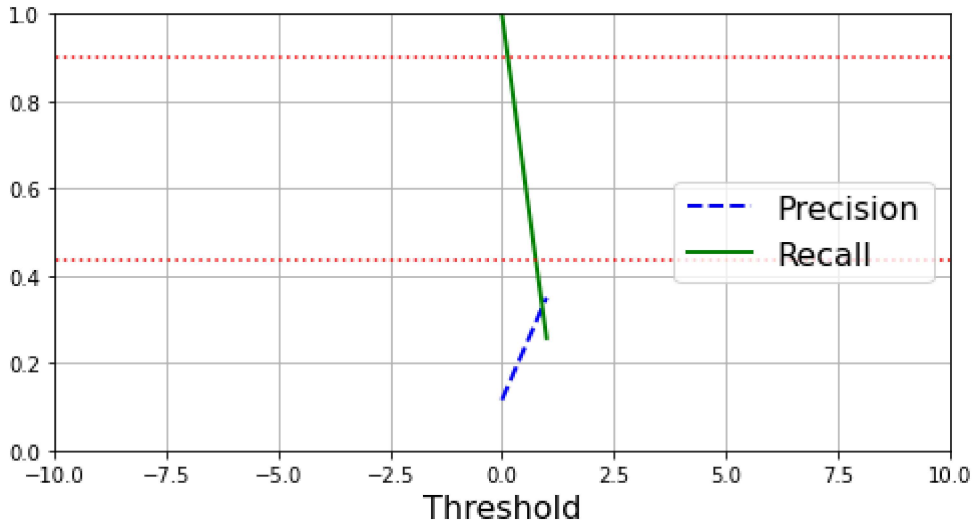
```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.plot([0.4368, 0.4368], [0., 0.9], "r:")
plt.plot([0.0, 0.4368], [0.9, 0.9], "r:")
plt.plot([0.4368], [0.9], "ro")
plt.show()
```



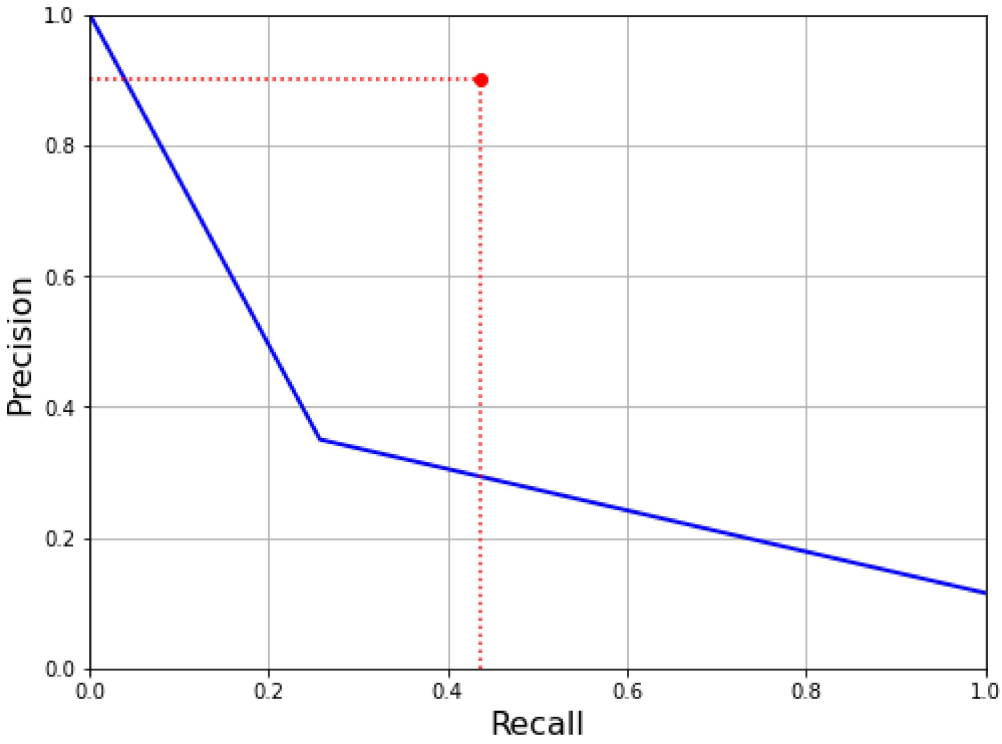
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y, y_scores)

def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16)
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
plt.plot([4.837e-3, 4.837e-3], [0., 0.4368], "r:")
plt.plot([0.0, 4.837e-3], [0.4368, 0.4368], "r:")
plt.plot([4.837e-3], [0.4368], "ro")

plt.show()
```



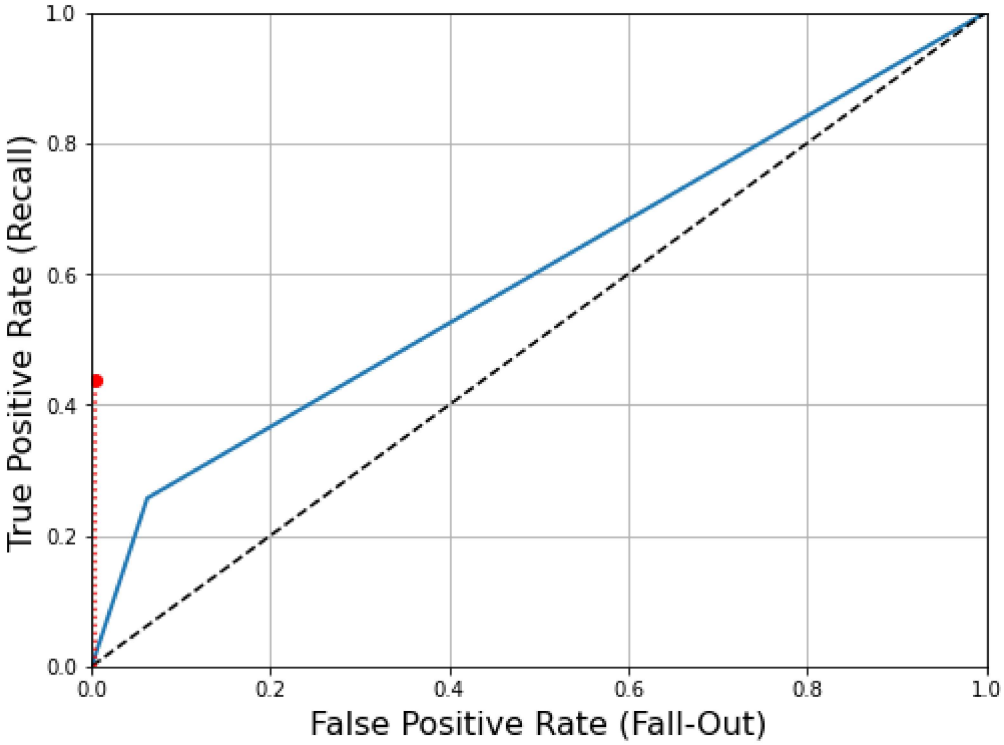
```
y_scores = cross_val_predict(gnb, X, y, cv=5)

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
    plt.legend(loc="center right", fontsize=16)
    plt.xlabel("Threshold", fontsize=16)
    plt.grid(True)
    plt.axis([-10, 10, 0, 1])

precisions, recalls, thresholds = precision_recall_curve(y, y_scores)
plt.figure(figsize=(8, 4))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.plot([7813, 7813], [0., 0.9], "r:")
plt.plot([-50000, 7813], [0.9, 0.9], "r:")
plt.plot([-50000, 7813], [0.4368, 0.4368], "r:")
plt.plot([7813], [0.9], "ro")
plt.plot([7813], [0.4368], "ro")
plt.show()
```




```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Recall", fontsize=16)
    plt.ylabel("Precision", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.plot([0.4368, 0.4368], [0., 0.9], "r:")
plt.plot([0.0, 0.4368], [0.9, 0.9], "r:")
plt.plot([0.4368], [0.9], "ro")
plt.show()
```




```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y, y_scores)

def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (Fall-Out)', fontsize=16)
    plt.ylabel('True Positive Rate (Recall)', fontsize=16)
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
plt.plot([4.837e-3, 4.837e-3], [0., 0.4368], "r:")
plt.plot([0.0, 4.837e-3], [0.4368, 0.4368], "r:")
plt.plot([4.837e-3], [0.4368], "ro")

plt.show()
```




```
model_predictions = pd.DataFrame({'y':y, 'y_Prediction_log':lgmodel.predict(X),
'y_prediction_NB':y_pred})
model_predictions['log_correct']=np.where(model_predictions['y']==model_predictions['y_Prediction_
log'],1,0)
model_predictions['nb_correct']=np.where(model_predictions['y']==model_predictions['y_prediction_N
B'],1,0)

y_Prediction_log=model_predictions['y_Prediction_log'].to_numpy()
y_prediction_NB=model_predictions['y_prediction_NB'].to_numpy()

print("Logistic Precision: {:.2f}%".format(100* precision_score(y, y_Prediction_log)))
print("Logistic Recall: {:.2f}%".format(100* precision_score(y, y_Prediction_log)))
print("Naive Bayes Precision: {:.2f}%".format(100*precision_score(y, y_prediction_NB)))
print("Naive Bayes Recall: {:.2f}%".format(100*recall_score(y, y_prediction_NB)))

Logistic Precision: 63.87%
Logistic Recall: 63.87%
Naive Bayes Precision: 36.64%
Naive Bayes Recall: 25.53%

lgmodel = LogisticRegression(random_state=0)
log_score = cross_val_score(lgmodel, X, y, cv=5, verbose=5)
log_score.mean()

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  ................................................................
[CV] ................................. , score=0.892, total=   0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.894, total=   0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:    0.0s
[CV]  ................................................................
```

```
[CV] ................................. , score=0.892, total=   0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.890, total=   0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.892, total=   0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s finished
Out[316]: 0.891838116657703

gnb = GaussianNB()
gnb_score = cross_val_score(gnb, X, y, cv=5, verbose=5)
gnb_score.mean()

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  ................................................................
[CV] ................................. , score=0.836, total=   0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.866, total=   0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.875, total=   0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.855, total=   0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[CV]  ................................................................
[CV] ................................. , score=0.864, total=   0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished
Out[317]: 0.8593282159096465

logistic percent correct: 89.21%
NB percent correct: 86.33%
```