Heart Disease Health Indicators Dataset

Xing Yang (Bill) Lan

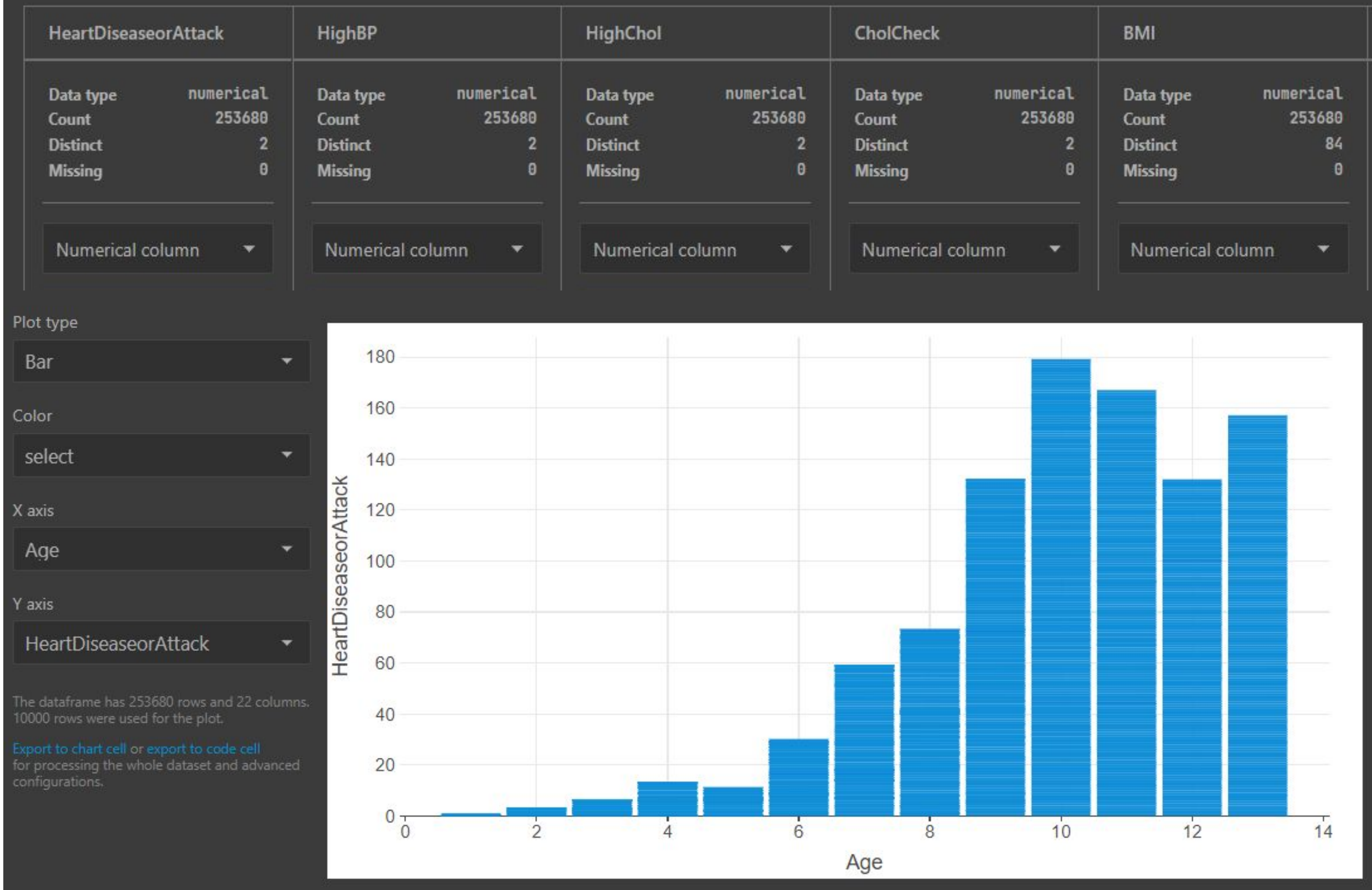
Logistic Regression & Lasso

“Having a cholesterol is bad”

Gradient Boosted Random Forest Ensemble

“Cholesterol is an Ordinally encoded categorical variable”

Everything is categorical, even age



	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI
Data type	numerical	numerical	numerical	numerical	numerical
Count	253680	253680	253680	253680	253680
Distinct	2	2	2	2	84
Missing	0	0	0	0	0
	Numerical column	Numerical column	Numerical column	Numerical column	Numerical column

Plot type: Bar

Color: select

X axis: Age

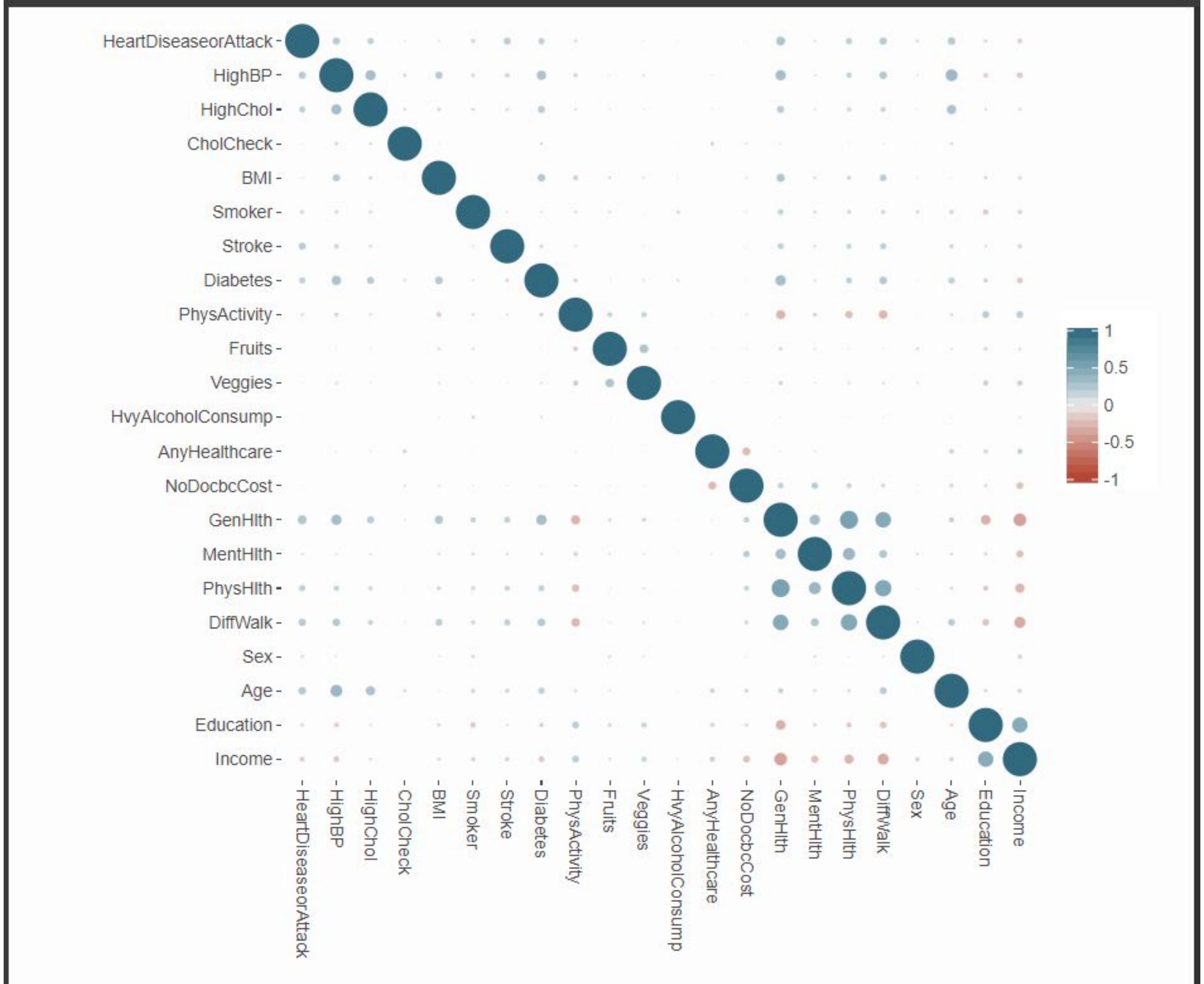
Y axis: HeartDiseaseorAttack

The dataframe has 253680 rows and 22 columns. 10000 rows were used for the plot.

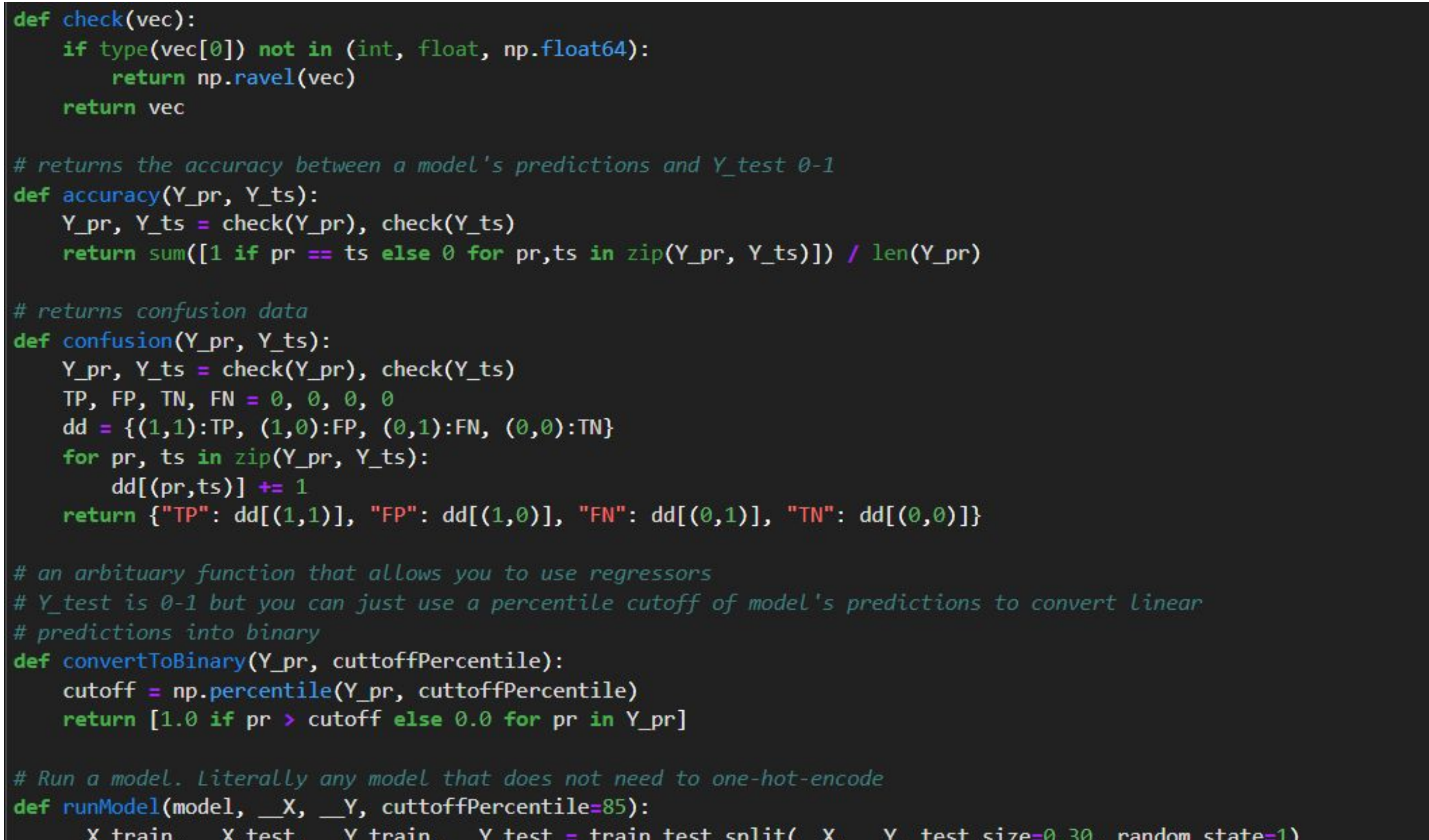
Export to chart cell or export to code cell for processing the whole dataset and advanced configurations.

Interaction Effects

“Fruit bad”?

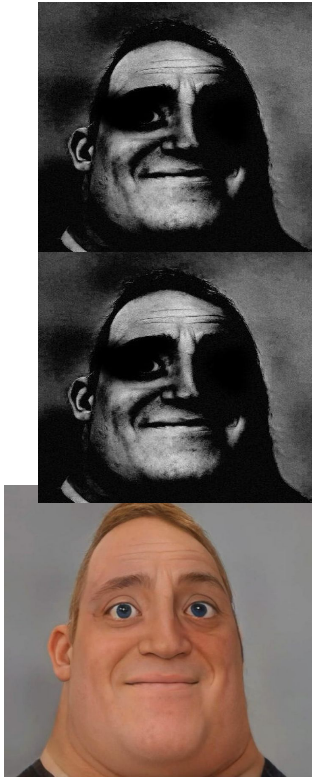


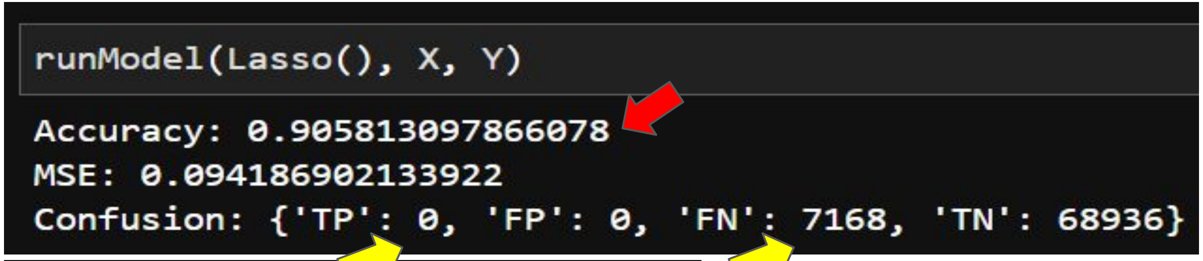
Convert results to Categorical to select larger Positives



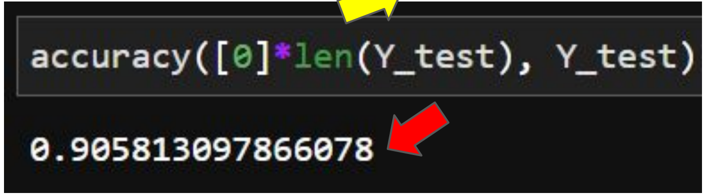
```python
def check(vec):
    if type(vec[0]) not in (int, float, np.float64):
        return np.ravel(vec)
    return vec

# returns the accuracy between a model's predictions and Y_test 0-1
def accuracy(Y_pr, Y_ts):
    Y_pr, Y_ts = check(Y_pr), check(Y_ts)
    return sum([1 if pr == ts else 0 for pr,ts in zip(Y_pr, Y_ts)]) / len(Y_pr)

# returns confusion data
def confusion(Y_pr, Y_ts):
    Y_pr, Y_ts = check(Y_pr), check(Y_ts)
    TP, FP, TN, FN = 0, 0, 0, 0
    dd = {(1,1):TP, (1,0):FP, (0,1):FN, (0,0):TN}
    for pr, ts in zip(Y_pr, Y_ts):
        dd[(pr,ts)] += 1
    return {"TP": dd[(1,1)], "FP": dd[(1,0)], "FN": dd[(0,1)], "TN": dd[(0,0)]}

# an arbituary function that allows you to use regressors
# Y_test is 0-1 but you can just use a percentile cutoff of model's predictions to convert linear
# predictions into binary
def convertToBinary(Y_pr, cuttoffPercentile):
    cutoff = np.percentile(Y_pr, cuttoffPercentile)
    return [1.0 if pr > cutoff else 0.0 for pr in Y_pr]

# Run a model. Literally any model that does not need to one-hot-encode
def runModel(model, __X, __Y, cuttoffPercentile=85):
    __X_train, __X_test, __Y_train, __Y_test = train_test_split(__X, __Y, test_size=0.30, random_state=1)
```

Lasso, Guessing 0, and Logistic Regression

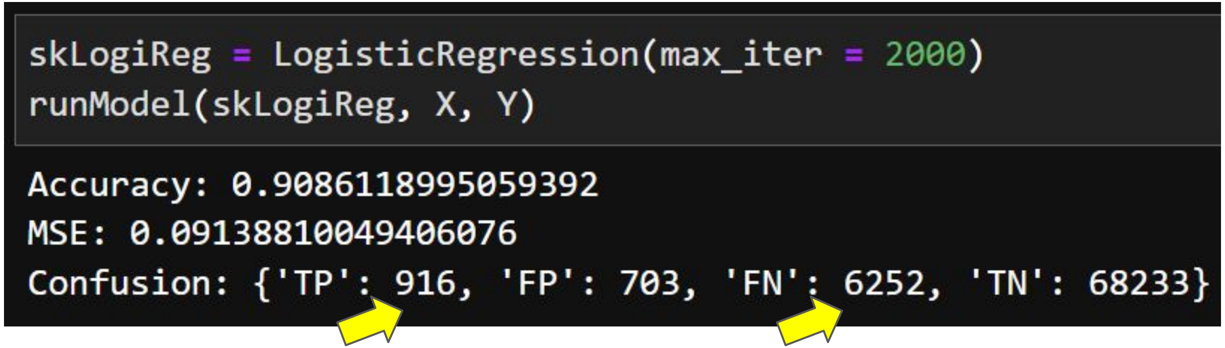




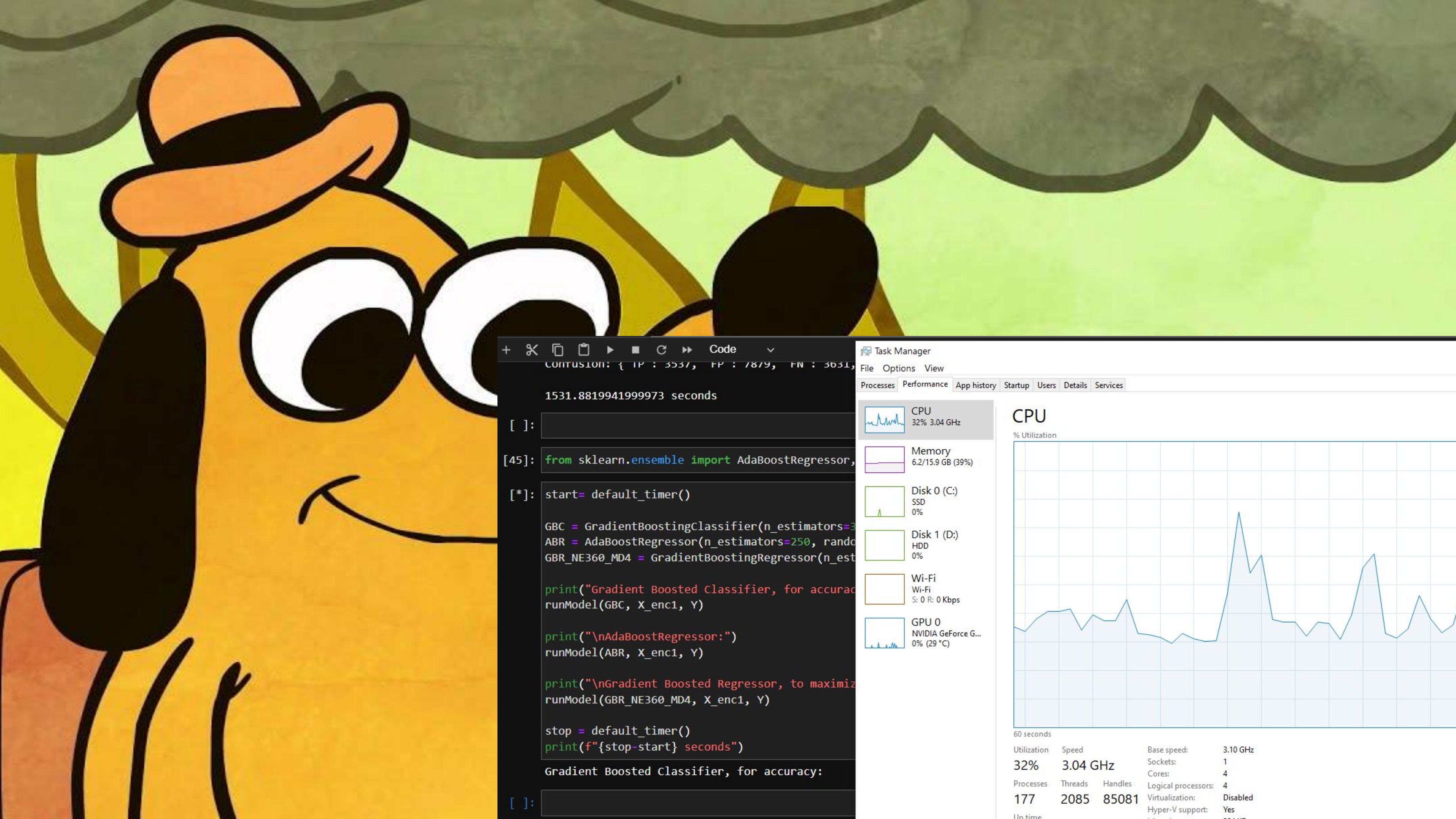
```
runModel(Lasso(), X, Y)

Accuracy: 0.905813097866078
MSE: 0.094186902133922
Confusion: {'TP': 0, 'FP': 0, 'FN': 7168, 'TN': 68936}
```



```
accuracy([0]*len(Y_test), Y_test)

0.905813097866078
```



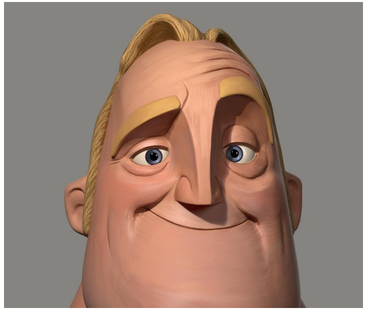
```
skLogiReg = LogisticRegression(max_iter = 2000)
runModel(skLogiReg, X, Y)

Accuracy: 0.9086118995059392
MSE: 0.09138810049406076
Confusion: {'TP': 916, 'FP': 703, 'FN': 6252, 'TN': 68233}
```

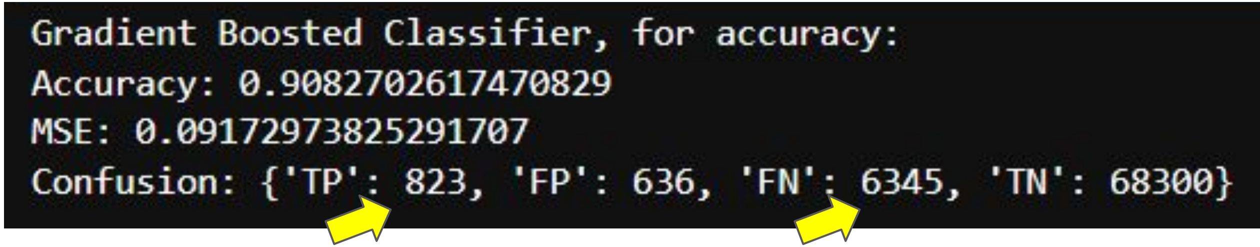


```
Confusion: { TP : 3537,  FP : 7879,  FN : 3631,

1531.8819941999973 seconds

[ ]:

[45]: from sklearn.ensemble import AdaBoostRegressor,

[*]: start= default_timer()

     GBC = GradientBoostingClassifier(n_estimators=3
     ABR = AdaBoostRegressor(n_estimators=250, rando
     GBR_NE360_MD4 = GradientBoostingRegressor(n_est

     print("Gradient Boosted Classifier, for accurac
     runModel(GBC, X_enc1, Y)

     print("\nAdaBoostRegressor:")
     runModel(ABR, X_enc1, Y)

     print("\nGradient Boosted Regressor, to maximiz
     runModel(GBR_NE360_MD4, X_enc1, Y)

     stop = default_timer()
     print(f"{stop-start} seconds")

     Gradient Boosted Classifier, for accuracy:

[ ]:
```

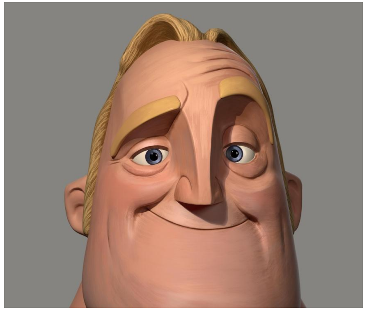
Utilization 32%
Speed 3.04 GHz
Processes 177
Threads 2085
Handles 85081
Up time

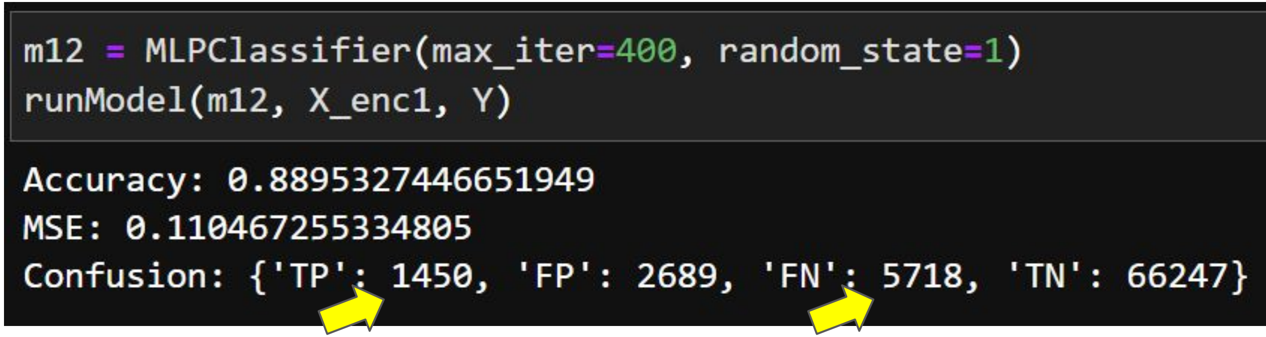
Base speed: 3.10 GHz
Sockets: 1
Cores: 4
Logical processors: 4
Virtualization: Disabled
Hyper-V support: Yes





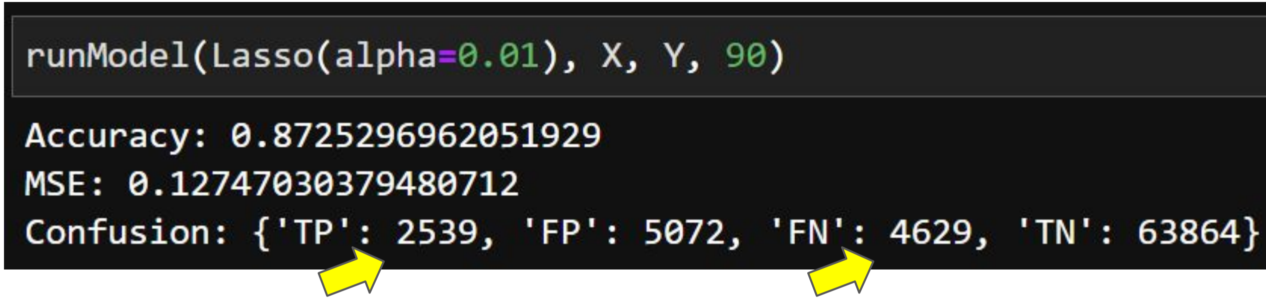
```
Gradient Boosted Classifier, for accuracy:
Accuracy: 0.9082702617470829
MSE: 0.09172973825291707
Confusion: {'TP': 823, 'FP': 636, 'FN': 6345, 'TN': 68300}
```





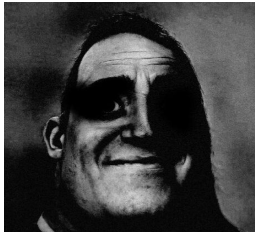
```
m12 = MLPClassifier(max_iter=400, random_state=1)
runModel(m12, X_enc1, Y)
```

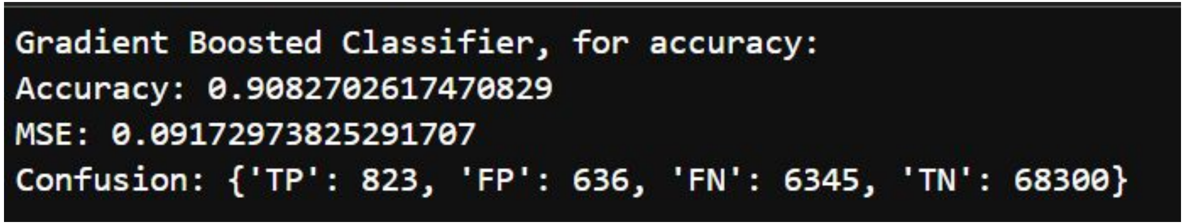
```
Accuracy: 0.8895327446651949
MSE: 0.110467255334805
Confusion: {'TP': 1450, 'FP': 2689, 'FN': 5718, 'TN': 66247}
```



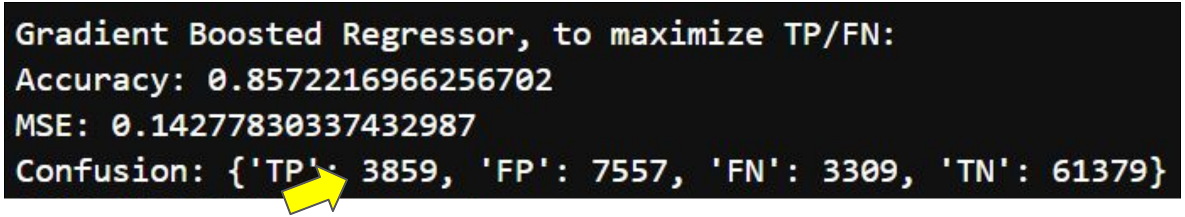
```
runModel(Lasso(alpha=0.01), X, Y, 90)
```

```
Accuracy: 0.8725296962051929
MSE: 0.12747030379480712
Confusion: {'TP': 2539, 'FP': 5072, 'FN': 4629, 'TN': 63864}
```



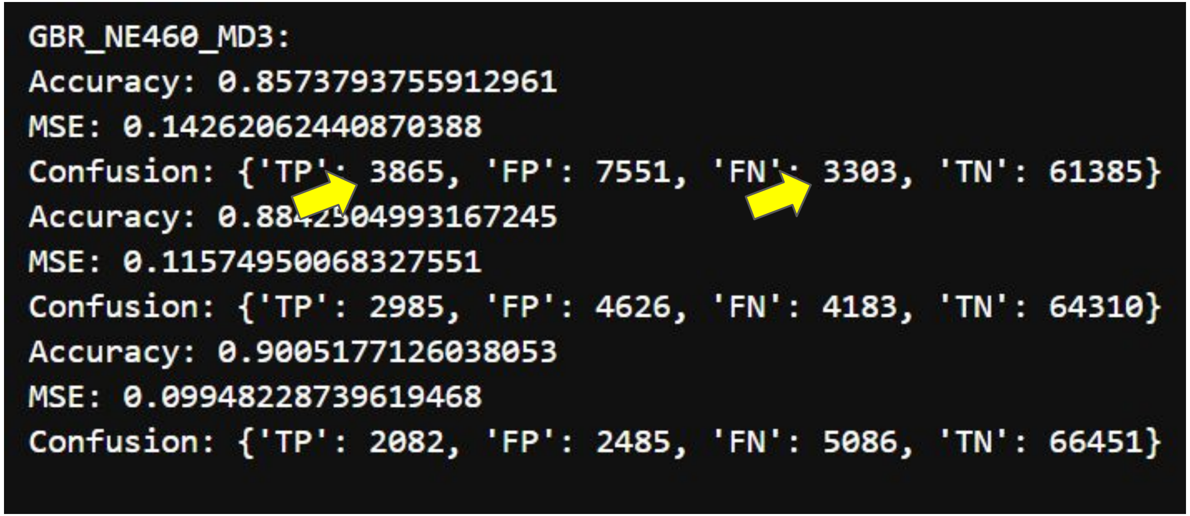


```
Gradient Boosted Classifier, for accuracy:
Accuracy: 0.9082702617470829
MSE: 0.09172973825291707
Confusion: {'TP': 823, 'FP': 636, 'FN': 6345, 'TN': 68300}
```

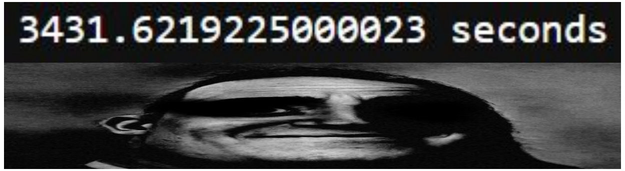


```
Gradient Boosted Regressor, to maximize TP/FN:
Accuracy: 0.8572216966256702
MSE: 0.14277830337432987
Confusion: {'TP': 3859, 'FP': 7557, 'FN': 3309, 'TN': 61379}
```

Gradient Boosted Regressor at Cutoffs 85%, 90%, and 94%



```
GBR_NE460_MD3:
Accuracy: 0.8573793755912961
MSE: 0.1426206244087038 8
Confusion: {'TP': 3865, 'FP': 7551, 'FN': 3303, 'TN': 61385}
Accuracy: 0.8842504993167245
MSE: 0.1157495006832755 1
Confusion: {'TP': 2985, 'FP': 4626, 'FN': 4183, 'TN': 64310}
Accuracy: 0.9005177126038053
MSE: 0.0994822873961946 8
Confusion: {'TP': 2082, 'FP': 2485, 'FN': 5086, 'TN': 66451}
```



```
3431.6219225000023 seconds
```