MScBMI 33200 – Machine Learning for Biomedical Informatics
Assignment IV
<Troy Zhongyi Zhang>

Directions:
1. Fill out below information (tables and methods)
2. Submit this document along with your code in an HTML/PDF format

Section 1: EMR Bots 30-day Readmission study

Using the training datasets, create the following models:

1. Naïve model: This model utilizes only patient characteristics (age, gender and race) to predict 30-day readmission in a logistic regression framework

2. GLM model : This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions in a logistic regression framework.

3. ANN model: This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions using an artificial neural network. Feature engineering steps include balancing classes using SMOTE as well as data normalization/standardization of continuous variables.

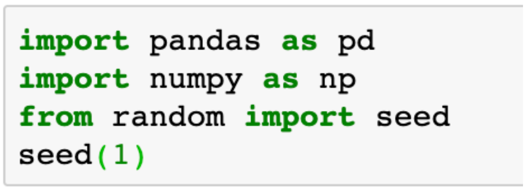
4. RF Model : This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions using a random forest.

5. GBM Model: This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions using a gradient boosted machine.

Utilize a five-fold cross-validation technique to build your model.
Calculate AUC on the test dataset. Fill out the following Table.

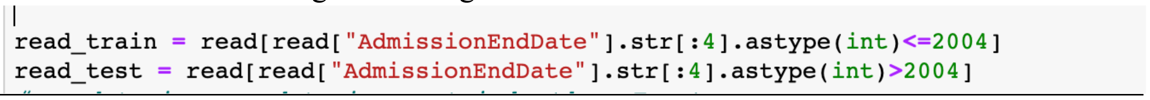
	AUC (95% CI)
Naïve Model	0.4906 (0.4103, 0.5708)
Logistic Regression	0.4779 (0.3918, 0.5640)
ANN	0.5075 (0.4281, 0.5869)
Random Forest	0.5018 (0.4295, 0.5741)
GBM	0.5061 (0.4298, 0.5823)

Insert details (packages, parameter selection, etc.) on the models that were developed in the space given below.
Methods:



```
import pandas as pd
import numpy as np
from random import seed
seed(1)
```

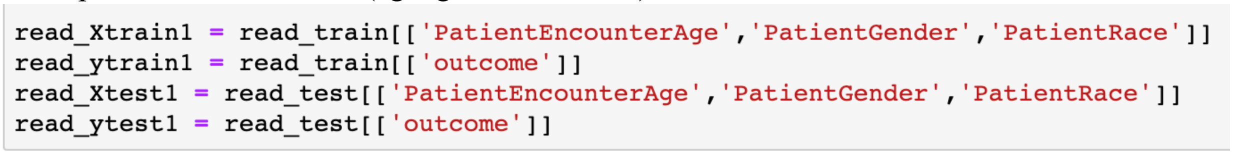
Firstly, import pandas as pd
import numpy as np
I separate the dataset into training and testing sets.



```
read_train = read[read["AdmissionEndDate"].str[:4].astype(int)<=2004]
read_test = read[read["AdmissionEndDate"].str[:4].astype(int)>2004]
```

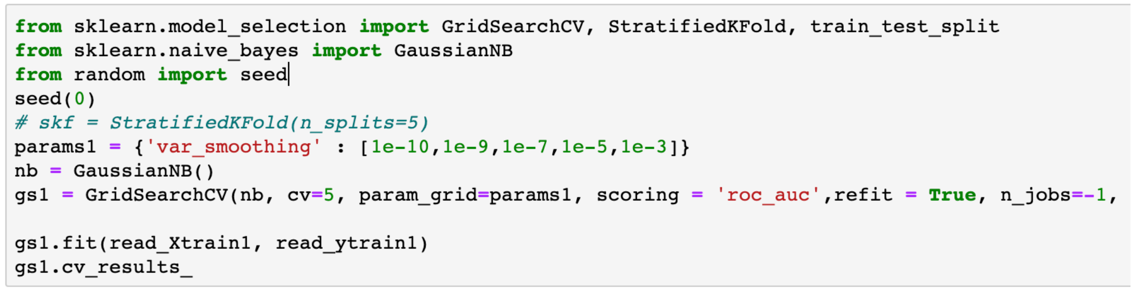
Naïve Model:

I select patient characteristics (age, gender and race) as the dataset



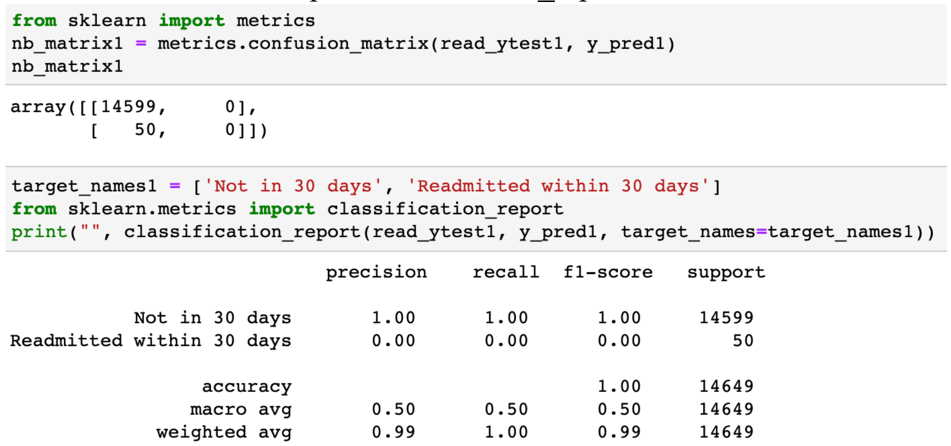
```
read_Xtrain1 = read_train[['PatientEncounterAge','PatientGender','PatientRace']]
read_ytrain1 = read_train[['outcome']]
read_Xtest1 = read_test[['PatientEncounterAge','PatientGender','PatientRace']]
read_ytest1 = read_test[['outcome']]
```

from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.naive_bayes import GaussianNB
from random import seed
and set seed(0)
I chose some parameters for GridSearchCV and I set "cv=5"



```
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.naive_bayes import GaussianNB
from random import seed
seed(0)
# skf = StratifiedKFold(n_splits=5)
params1 = {'var_smoothing' : [1e-10,1e-9,1e-7,1e-5,1e-3]}
nb = GaussianNB()
gs1 = GridSearchCV(nb, cv=5, param_grid=params1, scoring = 'roc_auc',refit = True, n_jobs=-1,

gs1.fit(read_Xtrain1, read_ytrain1)
gs1.cv_results_
```

I used "model.best_estimator_" to find the best parameters of the Naïve model according to the GridSearchCV and Cross validation.
from sklearn import metrics for confusion matrix
from sklearn.metrics import classification_report



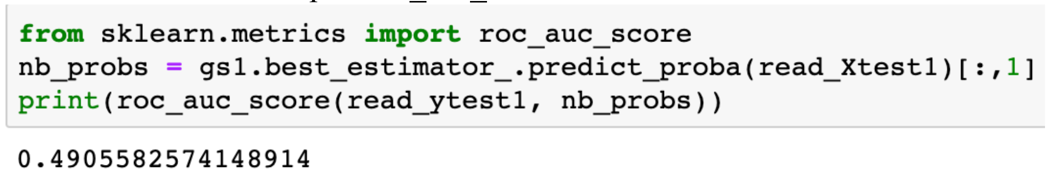
```
from sklearn import metrics
nb_matrix1 = metrics.confusion_matrix(read_ytest1, y_pred1)
nb_matrix1
```

```
array([[14599,     0],
       [   50,     0]])
```

```
target_names1 = ['Not in 30 days', 'Readmitted within 30 days']
from sklearn.metrics import classification_report
print("", classification_report(read_ytest1, y_pred1, target_names=target_names1))
```

```
                           precision    recall  f1-score   support

           Not in 30 days       1.00      1.00      1.00     14599
Readmitted within 30 days       0.00      0.00      0.00        50

                 accuracy                           1.00     14649
                macro avg       0.50      0.50      0.50     14649
             weighted avg       0.99      1.00      0.99     14649
```

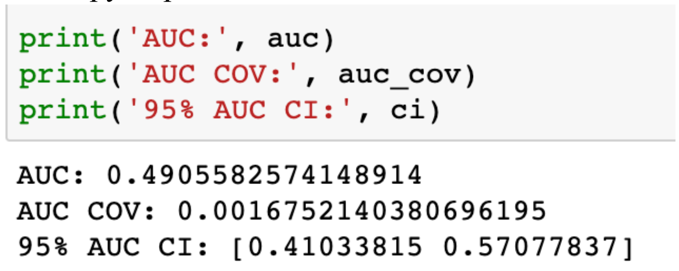
from sklearn.metrics import roc_auc_score to calculate 95% AUC



```
from sklearn.metrics import roc_auc_score
nb_probs = gs1.best_estimator_.predict_proba(read_Xtest1)[:,1]
print(roc_auc_score(read_ytest1, nb_probs))
```

```
0.4905582574148914
```

Since Python does not have the pROC package, I transplanted the DeLong function from R to calculate the confidence interval in Python
import numpy as np; import scipy.stats; from scipy import stats
#Transplanted the pROC package from R into Python for CI computation
import numpy as np
import scipy.stats
from scipy import stats

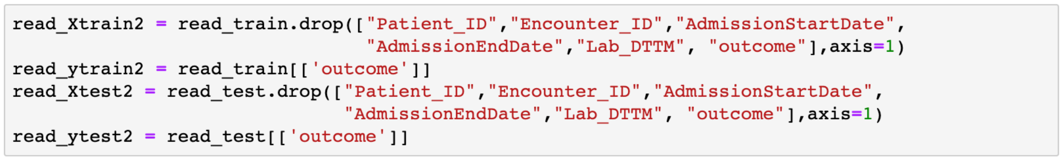


```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.4905582574148914
AUC COV: 0.0016752140380696195
95% AUC CI: [0.41033815 0.57077837]
```

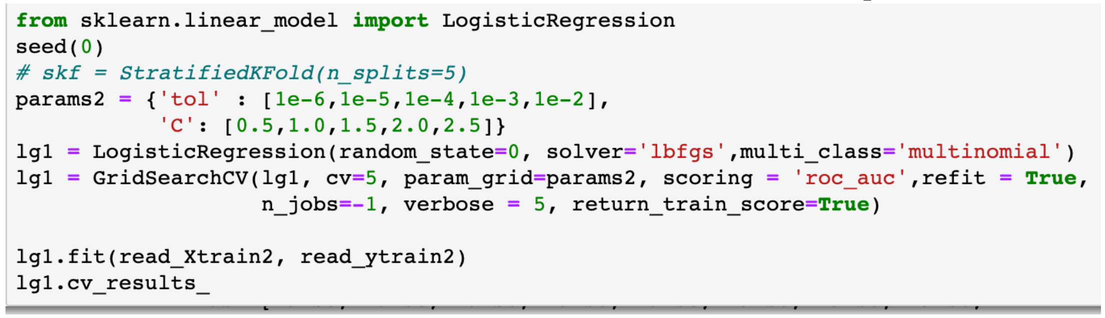
Logistic Regression:

I choose the patient characteristics and latest lab observations as my dataset.



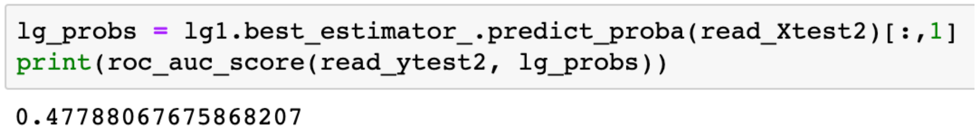
```
read_Xtrain2 = read_train.drop(["Patient_ID","Encounter_ID","AdmissionStartDate",
                                "AdmissionEndDate","Lab_DTTM", "outcome"],axis=1)
read_ytrain2 = read_train[['outcome']]
read_Xtest2 = read_test.drop(["Patient_ID","Encounter_ID","AdmissionStartDate",
                              "AdmissionEndDate","Lab_DTTM", "outcome"],axis=1)
read_ytest2 = read_test[['outcome']]
```

from sklearn.linear_model import LogisticRegression to fit a logistic regression model
Used GridSearchCV and set "cv=5" for 5 cross-validations to tune parameters.



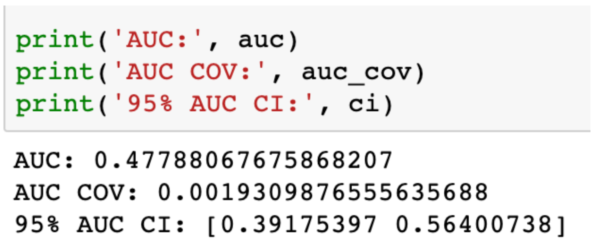
```
from sklearn.linear_model import LogisticRegression
seed(0)
# skf = StratifiedKFold(n_splits=5)
params2 = {'tol' : [1e-6,1e-5,1e-4,1e-3,1e-2],
           'C': [0.5,1.0,1.5,2.0,2.5]}
lg1 = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial')
lg1 = GridSearchCV(lg1, cv=5, param_grid=params2, scoring = 'roc_auc',refit = True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

lg1.fit(read_Xtrain2, read_ytrain2)
lg1.cv_results_
```

Then I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to apply the best parameters for prediction:



```
lg_probs = lg1.best_estimator_.predict_proba(read_Xtest2)[:,1]
print(roc_auc_score(read_ytest2, lg_probs))
```

```
0.47788067675868207
```

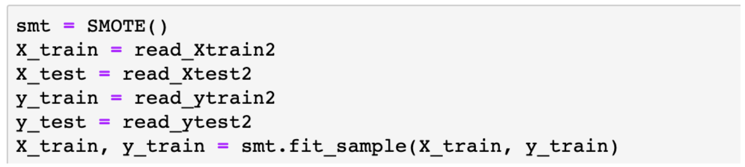


```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.47788067675868207
AUC COV: 0.0019309876555635688
95% AUC CI: [0.39175397 0.56400738]
```

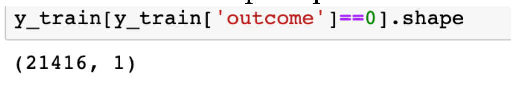
Artificial Neural Network:

from imblearn.over_sampling import SMOTE for using the SMOTE to balance the dataset.
Then I filled my training sets:

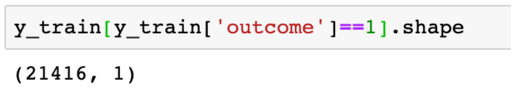


```
smt = SMOTE()
X_train = read_Xtrain2
X_test = read_Xtest2
y_train = read_ytrain2
y_test = read_ytest2
X_train, y_train = smt.fit_sample(X_train, y_train)
```

To test my new dataset after SMOTE is balanced: (Both outcome=1 and outcome=1 has the same number of rows. I upsampled the minority group to equal to the majority group)

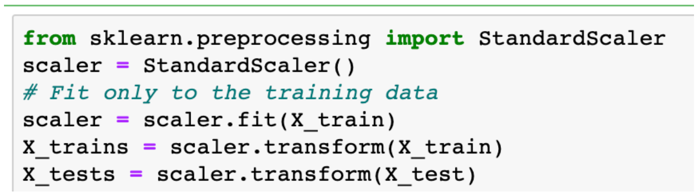


```
y_train[y_train['outcome']==0].shape

(21416, 1)
```



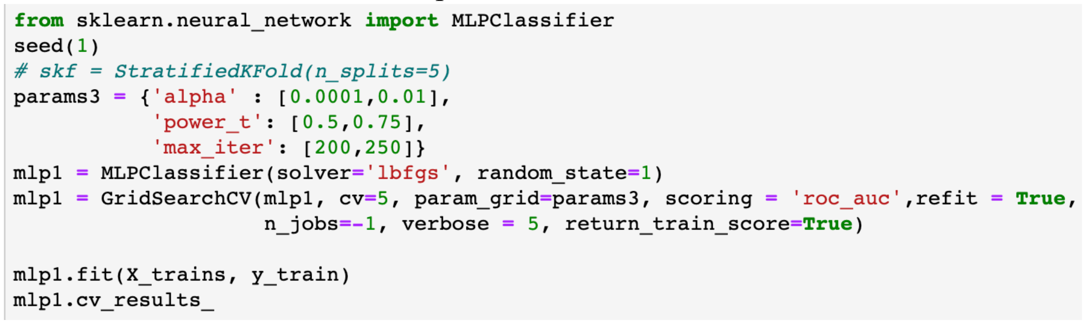
```
y_train[y_train['outcome']==1].shape

(21416, 1)
```

from sklearn.preprocessing import StandardScaler to scale my data. I fitted my scaler with the training set and transferred the scaling to my testing set to avoid different variances.



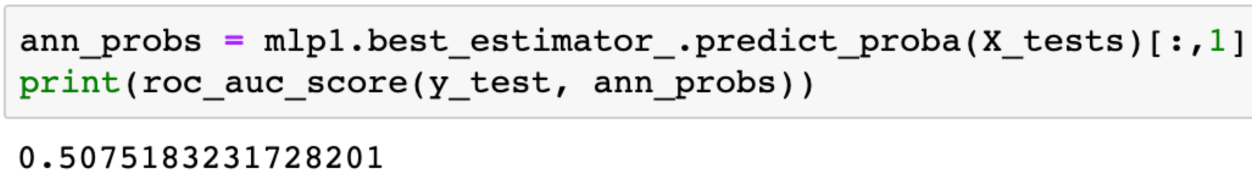
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(X_train)
X_trains = scaler.transform(X_train)
X_tests = scaler.transform(X_test)
```

from sklearn.neural_network import MLPClassifier to fit a neural network model.
The I used GridSearchCV to tune parameters with 5 cross-validation.

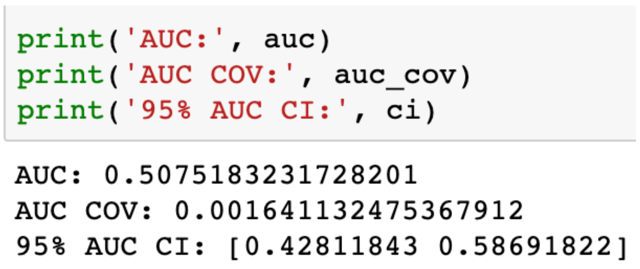


```
from sklearn.neural_network import MLPClassifier
seed(1)
# skf = StratifiedKFold(n_splits=5)
params3 = {'alpha' : [0.0001,0.01],
           'power_t': [0.5,0.75],
           'max_iter': [200,250]}
mlp1 = MLPClassifier(solver='lbfgs', random_state=1)
mlp1 = GridSearchCV(mlp1, cv=5, param_grid=params3, scoring = 'roc_auc',refit = True,
                    n_jobs=-1, verbose = 5, return_train_score=True)

mlp1.fit(X_trains, y_train)
mlp1.cv_results_
```

Lastly, I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to apply the best parameters for prediction:



```
ann_probs = mlp1.best_estimator_.predict_proba(X_tests)[:,1]
print(roc_auc_score(y_test, ann_probs))

0.5075183231728201
```



```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)

AUC: 0.5075183231728201
AUC COV: 0.001641132475367912
95% AUC CI: [0.42811843 0.58691822]
```

Random Forest:

from sklearn.ensemble import RandomForestClassifier to fit a Random Forest model. I used GridSearchCV with 5 cross-validations to tune parameters, and I used the same training and testing datasets as the logistic regression model's datasets. These datasets are before scaling:

```
from sklearn.ensemble import RandomForestClassifier
seed(42)

params4 = {'n_estimators' : [10,100,150],
           'min_samples_leaf': [1,2,3]}
rf1 = RandomForestClassifier(random_state=42)
rf1 = GridSearchCV(rf1, cv=5, param_grid=params4, scoring = 'roc_auc',refit = True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

rf1.fit(read_Xtrain2, read_ytrain2)
rf1.cv_results_
```

I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to apply the best parameters for prediction:

```
rf_probs = rf1.best_estimator_.predict_proba(read_Xtest2)[:,1]
print(roc_auc_score(read_ytest2, rf_probs))
```

```
0.5017823138571136
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.5017823138571135
AUC COV: 0.0013611946962794295
95% AUC CI: [0.42947067 0.57409395]
```

Gradient Boosting Machines:

from sklearn.ensemble import GradientBoostingClassifier to fit a Gradient Boosting model. I used GridSearchCV with 5 crross-validations to tune parameters, and I used the same training and testing datasets as the logistic regression model's datasets, which are before scaling:

```
from sklearn.ensemble import GradientBoostingClassifier
seed(10)

params5 = {'learning_rate' : [0.05,0.1,0.2],
           'n_estimators': [100,150],
           'min_samples_split': [2,3,4]}
gbm1 = GradientBoostingClassifier(random_state=10)
gbm1 = GridSearchCV(gbm1, cv=5, param_grid=params5, scoring = 'roc_auc',refit = True,
                    n_jobs=-1, verbose = 5, return_train_score=True)

gbm1.fit(read_Xtrain2, read_ytrain2)
gbm1.cv_results_
```

I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to apply the best parameters for prediction:

```
gbm_probs = gbm1.best_estimator_.predict_proba(read_Xtest2)[:,1]
print(roc_auc_score(read_ytest2, gbm_probs))
```

```
0.5060826083978355
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.5060826083978355
AUC COV: 0.001513806136731901
95% AUC CI: [0.42982499 0.58234022]
```

Section 2: Gusto study

Using the training datasets, create the following models:

1. GLM model: This model utilizes all features to predict 30-day mortality in a logistic regression framework.
2. Ridge Regression model: This model utilizes all features to predict 30-day mortality in a logistic regression framework with regularization.
3. ANN model: This model utilizes all features to predict 30-day mortality using an artificial neural network. Feature engineering steps should use normalization/standardization of continuous variables.
4. Random Forest model: This model utilizes all features to predict 30-day mortality using a random forest.
5. GBM model: This model utilizes all features to predict 30-day mortality using a gradient boosted machine.

Utilize a five-fold cross-validation technique to build your model.
Calculate AUC on the test dataset. Fill out the following Table.

	AUC (95% CI)
Logistic Regression	0.8302 (0.7938, 0.8666)
Ridge Regression	0.8280 (0.7912, 0.8647)
ANN	0.7596 (0.7145, 0.8047)
Random Forest	0.8853 (0.8523, 0.9183)
GBM Model	0.8721 (0.8403, 0.9039)

Insert details on the models that were developed in the space given below.
Methods: Firstly, same as the above question, import pandas as pd, import numpy as np

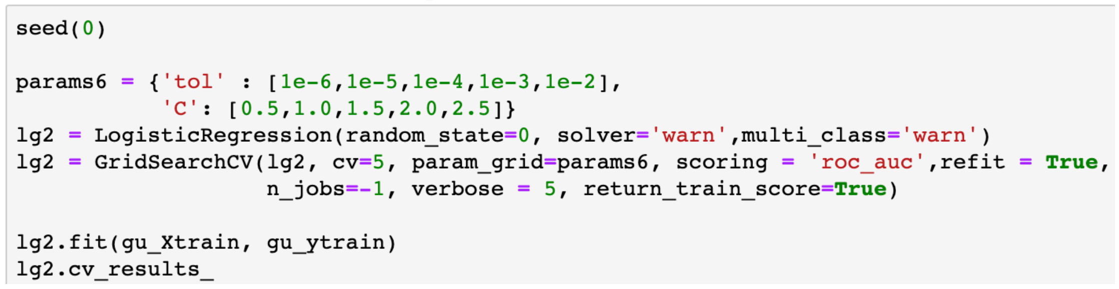
```
import pandas as pd
import numpy as np
from random import seed
seed(1)
```

Then I separated the GUSTO dataset into training and testing datasets:

```
gu_train = gusto.loc[(gusto['GROUP'] == 1
                     ) | (gusto['GROUP'] == 2
                         ) | (gusto['GROUP'] == 3)]
gu_test = gusto[gusto['GROUP']==0]

gu_Xtrain = gu_train.drop("DAY30",axis=1)
gu_ytrain = gu_train[['DAY30']]
gu_Xtest = gu_test.drop("DAY30",axis=1)
gu_ytest = gu_test[['DAY30']]
```

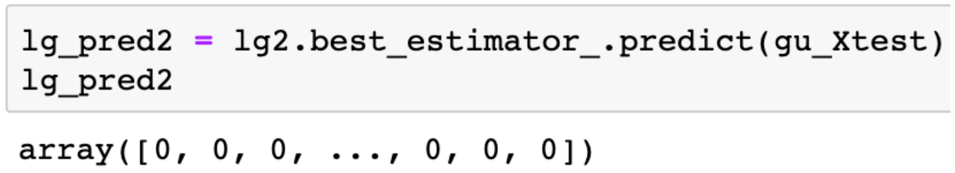
Logistic Regression:

from sklearn.linear_model import LogisticRegression to fit a logistic regression model. I used GridSearchCV with 5 cross validations to tune parameters.
from sklearn.model_selection import GridSearchCV



```
seed(0)

params6 = {'tol' : [1e-6,1e-5,1e-4,1e-3,1e-2],
           'C': [0.5,1.0,1.5,2.0,2.5]}
lg2 = LogisticRegression(random_state=0, solver='warn',multi_class='warn')
lg2 = GridSearchCV(lg2, cv=5, param_grid=params6, scoring = 'roc_auc',refit = True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

lg2.fit(gu_Xtrain, gu_ytrain)
lg2.cv_results_
```

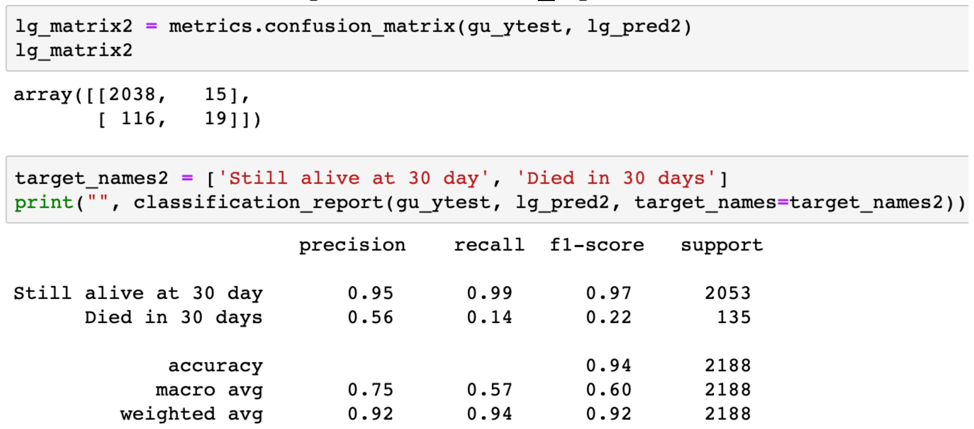
I used model.best_estimator_ to predict results.



```
lg_pred2 = lg2.best_estimator_.predict(gu_Xtest)
lg_pred2
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

Then I imported the metrics package to obtain the confusion matrix:
from sklearn import metrics for confusion matrix
from sklearn.metrics import classification_report



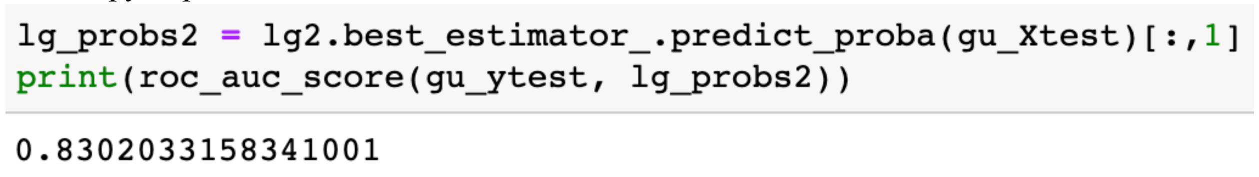
```
lg_matrix2 = metrics.confusion_matrix(gu_ytest, lg_pred2)
lg_matrix2
```

```
array([[2038,   15],
       [ 116,   19]])
```

```
target_names2 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, lg_pred2, target_names=target_names2))
```

```
                        precision    recall  f1-score   support

Still alive at 30 day       0.95      0.99      0.97      2053
      Died in 30 days       0.56      0.14      0.22       135

             accuracy                           0.94      2188
            macro avg       0.75      0.57      0.60      2188
         weighted avg       0.92      0.94      0.92      2188
```

Finally, I used same method to calculate the confidence interval and 95% CI with using model.best_parameter_ for predictions:
from sklearn.metrics import roc_auc_score to calculate 95% AUC
import numpy as np; import scipy.stats; from scipy import stats
#Transplanted the pROC package from R into Python for CI computation
import numpy as np
import scipy.stats
from scipy import stats



```
lg_probs2 = lg2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, lg_probs2))
```

```
0.8302033158341001
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.8302033158341
AUC COV: 0.0003444411856651161
95% AUC CI: [0.7938281  0.86657854]
```

Ridge Regression Model:

from sklearn.linear_model import RidgeCV
I directly set cv = 5 for 5 cross-validations to predict results

```
from sklearn.linear_model import RidgeCV
ridgecv = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 10], cv=5,
ridgecv=ridgecv.fit(gu_Xtrain,gu_ytrain)
gu_ridgecv = ridgecv.predict(gu_Xtest)
roc_auc_score(gu_ytest, gu_ridgecv)
```

```
0.8279590842669263
```

I used above method to calculate the AUC and 95% confidence interval:

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.8279590842669264
AUC COV: 0.0003523557006508896
95% AUC CI: [0.79116833 0.86474984]
```

Artificial Neural Network:

from sklearn.preprocessing import StandardScaler to scale my data. I fitted my scaler with the training set and transferred the scaling to my testing set to avoid different variances.

```
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(gu_Xtrain)
gu_Xtrains = scaler.transform(gu_Xtrain)
gu_Xtests = scaler.transform(gu_Xtest)
```

from sklearn.neural_network import MLPClassifier to fit a neural network model.
The I used GridSearchCV to tune parameters with 5 cross-validations.

```
seed(1)
# skf = StratifiedKFold(n_splits=5)
params7 = {'alpha' : [0.0001,0.01],
           'power_t': [0.5,0.75],
           'max_iter': [200,250]}
mlp2 = MLPClassifier(solver='lbfgs', random_state=1)
mlp2 = GridSearchCV(mlp2, cv=5, param_grid=params7, scoring = 'roc_auc',refit = True,
                    n_jobs=-1, verbose = 5, return_train_score=True)

mlp2.fit(gu_Xtrains, gu_ytrain)
mlp2.cv_results_
```

Lastly, I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to use the best parameters for prediction:

```
ann_probs2 = mlp2.best_estimator_.predict_proba(gu_Xtests)[:,1]
print(roc_auc_score(gu_ytest, ann_probs2))
```

```
0.759576771120853
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.759576771120853
AUC COV: 0.0005295762772071846
95% AUC CI: [0.71447305 0.80468049]
```

Random Forest:

from sklearn.ensemble import RandomForestClassifier to fit a Random Forest model. I used GridSearchCV with 5 cross-validations to tune parameters, and I used the same training and testing datasets as the logistic regression model's datasets. These datasets are before scaling:

```
seed(42)

params8 = {'n_estimators' : [10,100,150],
           'min_samples_leaf': [1,2,3]}
rf2 = RandomForestClassifier(random_state=42)
rf2 = GridSearchCV(rf2, cv=5, param_grid=params8, scoring = 'roc_auc',refit = True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

rf2.fit(gu_Xtrain,gu_ytrain)
rf2.cv_results_
```

I used above method to calculate the AUC and 95% CI with using "model.best_estimator_" to use the best parameters for prediction:

```
rf_probs2 = rf2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, rf_probs2))
```

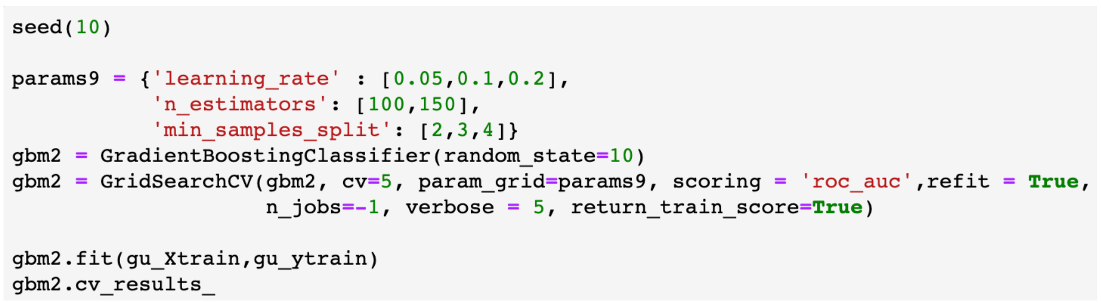
```
0.885331312803305
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.885331312803305
AUC COV: 0.0002835084573262921
95% AUC CI: [0.85233001 0.91833262]
```

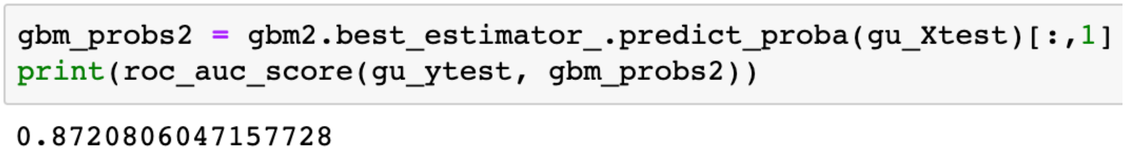
Gradient Boosting Machines:

from sklearn.ensemble import GradientBoostingClassifier to fit a Gradient Boosting model. I used GridSearchCV with 5 crross-validations to tune parameters, and I used the same training and testing datasets as the logistic regression model's datasets, which are before scaling:

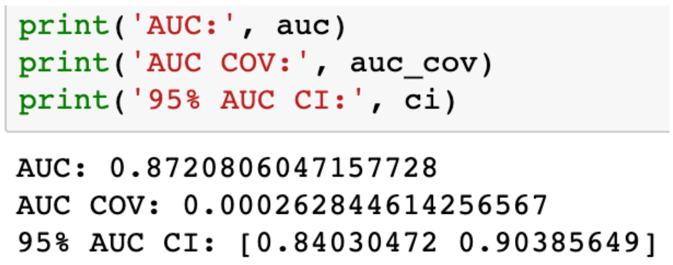


```
seed(10)

params9 = {'learning_rate' : [0.05,0.1,0.2],
           'n_estimators': [100,150],
           'min_samples_split': [2,3,4]}
gbm2 = GradientBoostingClassifier(random_state=10)
gbm2 = GridSearchCV(gbm2, cv=5, param_grid=params9, scoring = 'roc_auc',refit = True,
                    n_jobs=-1, verbose = 5, return_train_score=True)

gbm2.fit(gu_Xtrain,gu_ytrain)
gbm2.cv_results_
```

I used above method to calculate the AUC and 95% CI with using “model.best_estimator_” to apply the best parameters for prediction:



```
gbm_probs2 = gbm2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, gbm_probs2))

0.8720806047157728
```



```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)

AUC: 0.8720806047157728
AUC COV: 0.000262844614256567
95% AUC CI: [0.84030472 0.90385649]
```

Section 3: Short Answer Questions

Please answer the following questions briefly.

a. Explain briefly the intuition behind using information gain when building decision trees.’

Ans. Information gain measures how predictable the data became (or how much more information gained) by splitting the data according to the answer provided at the decision node. Information gained using feature F on data Y is defined by: IG(Y, F) = H(Y)-H(Y|F). The more powerful a feature is, the more information is gained by splitting the data on that feature.

b. What is difference between test error as measured by cross-validation and test error measured through bootstrap?

Ans. One difference is that cross-validation, like jackknife, uses all of the data points, whereas bootstrapping, which resamples the data randomly, may not hit all the points. The error comes down to variance and bias (as usual). CV tends to be less biased but K-fold CV has fairly large variance. On the other hand, bootstrapping tends to drastically reduce the variance but gives more biased results (they tend to be pessimistic). For large sample sizes, the variance issues become less important and the computational part is more of an issues. I still would stick by repeated CV for small and large sample sizes. Cross validation is a good tool when deciding on the model. However, when my model is fixed, using the bootstrap makes more sense.

c. **Why do we need de-correlated trees when developing random forests?**

Ans. Random Forests is a powerful ensemble method used to build predictive models by constructing a large number of de-correlated trees by showing different training sets to solve the problem of overfitting which involves averaging the predictor output from different tree models which are distributed across helping in reducing the variance.

d. **What does a varImp() on a random forest model indicate?**

Ans. The varImp function tracks the changes in model statistics, such as the GCV, for each predictor and accumulates the reduction in the statistic when each predictor's feature is added to the model. This total reduction is used as the variable importance measure. varrImp() is a generic method for calculating variable importance for objects produced by training set and method specific methods. For the out-of-bag sample of each tree calculate the accuracy of the tree, then permute the variables one after the other and measure the accuracy after permutation to calculate the decrease in accuracy without that variable.

e. **True or False: Boosting involves combining independent decision trees as learners.**

Ans. False. Bagging works on independent models but boosting works on sequential models.

f. **What is the interpretation of learning rate in a gradient boosting model?**

Ans. The learning rate corresponds to how quickly the error is corrected from each tree to the next and is a simple multiplier $0<LR\leq 1$. For example, if the current prediction for a particular example is 0.2 and the next tree predicts that it should actually be 0.8, the correction would be +0.6. At a learning rate of 1, the updated prediction would be the full $0.2+1*(0.6) = 0.8$, while a learning rate of 0.1 would update the prediction to be $0.2+0.1*(0.6) = 0$.

g. **Suppose I used the top 10 features as indicated by varImp() from my random forest model to create a multiple linear regression model. Is this a valid method? Why or why not?**

Ans. No, this will not be a valid method. Variables that are important in random forest don't necessarily have any sort of linearly additive relationship with the outcome. This remark shouldn't be surprising: it's what makes random forest so effective at discovering nonlinear relationships. In general, random forest can fit different models to different subsets of the data. It won't be obvious what's going on from a single plot at all, and building a linear model of similar predictive power will be even harder. Some of the features we got from varImp() through random forest may not fall in a linear decision boundary, and these features won't work well with multiple linear regression.

Some notes:

A method that first looks at univariate correlations for pre-identifying things that should go into a final model, will tend to do badly for a number of reasons: ignoring model uncertainy, using statistical significance/strength of correlation as a criterion to select, "falsely" identifying predictors in univariate correlations, and missing out on predictors. Not wrapping this into any form of bootstrapping/cross-validation/whatever to get a realistic assessment of my model uncertainty is likely to mislead me. Furthermore, treating continuous predictors as having linear effects can often be improved upon by methods that do not make such an assumption (e.g. RF).

Using RF as a pre-selection for a linear model is not such a good idea. Variable importance is really hard to interpret, and it is really hard to set a cut-off on it. I will not know whether variable importance is about the variable itself or about interactions, plus I am losing out on non-linear transformations of variables.