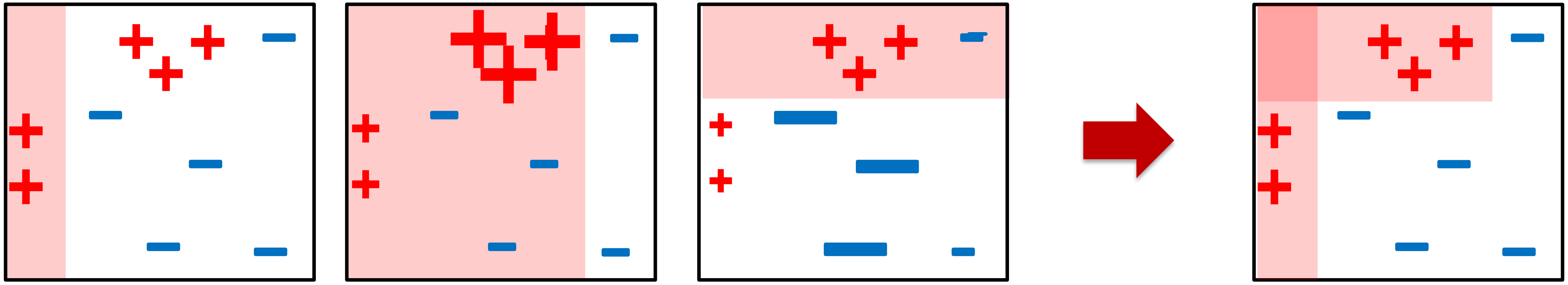
# Machine Learning for Biomedical/Healthcare Applications

Combining weak classifiers….



….to make strong ones

## Ensemble Learning : Random Forests and Boosting

Dr Emma Robinson

# Learning Outcomes from the tutorial

1. Understanding of how to implement Bagged ensembles of trees through
    1. Creating bootstrapped samples of a data set
    2. Training large numbers of trees
    3. Aggregating test predictions
2. Compare the performance of ensembles relative to decision Trees
3. Be able to deploy bagging, forests and boosting in Scikit-Learn
4. Be able to perform parameter optimization for ensemble methods

**Note**
**– notebooks folder recently updated as adaboost algorithm figures were not updated from last year (solutions notebook are correct however)**

## Exercises

1. Exercise 2 - **Building a Bagging Classifier (30 mins)**
    1. Complete `create_bagged_ensemble` to bootstrap samples from data and train ensemble of trees
    2. Complete `bagging_predict` to aggegate test predictions through majority voting

2. Exercise 3 - **Comparing our Bagged Predictor against our Decision Tree** (just run code – 5 mins)

3. Exercise 4 **- Comparing against Scikit learn (10-15 mins)**
    1. Build a decision tree and bagging classifier with scikit learn; fit model
    2. Return test prediction and accuracy (score) -> compare

4. Exercise 5: **Training Random Forests to Predict Gestational Age from Regional Brain Volumes¶ (20 mins)**
    - implement the training and testing of the DecisionTreeRegressor()
    - implement the training and testing of the RandomForestRegressor()
    - (optional) Return and plot feature importances see the scikit-learn tutorial for guidance

## Exercises

5. Exercise 6 (optional): Perform Parameter Optimisation for Random Forests using GridSearchCV (10-15 mins)

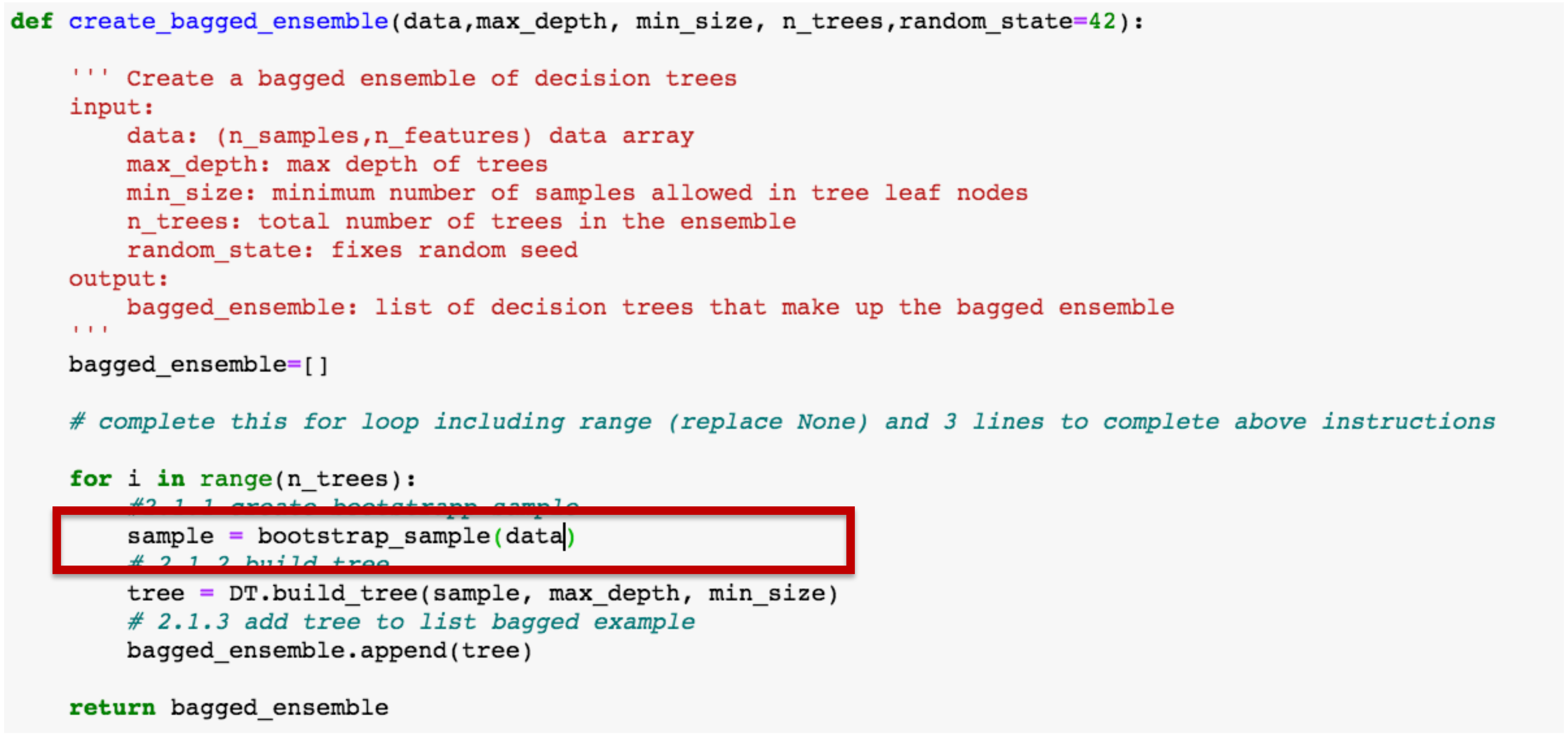
6. Exercise 7 (optional/homework) Building a Random Forest Classifier from scratch (20-30 mins)

7. Exercise 8: **Training Adaboost to Predict Gestational Age from Regional Brain Volumes (20-30 mins)¶**
   - Implement adaboost regression using scikit learn
   - Apply GridCV optimisation of the ensemble and base learner paramters

# Solutions Ex 2 Building a Bagging Classifier

For each tree in the ensemble

1. Create bootstrapped sample from data by calling the `bootstrap_sample` function you wrote during the video lectures
   - no need to set random state (except when debugging)

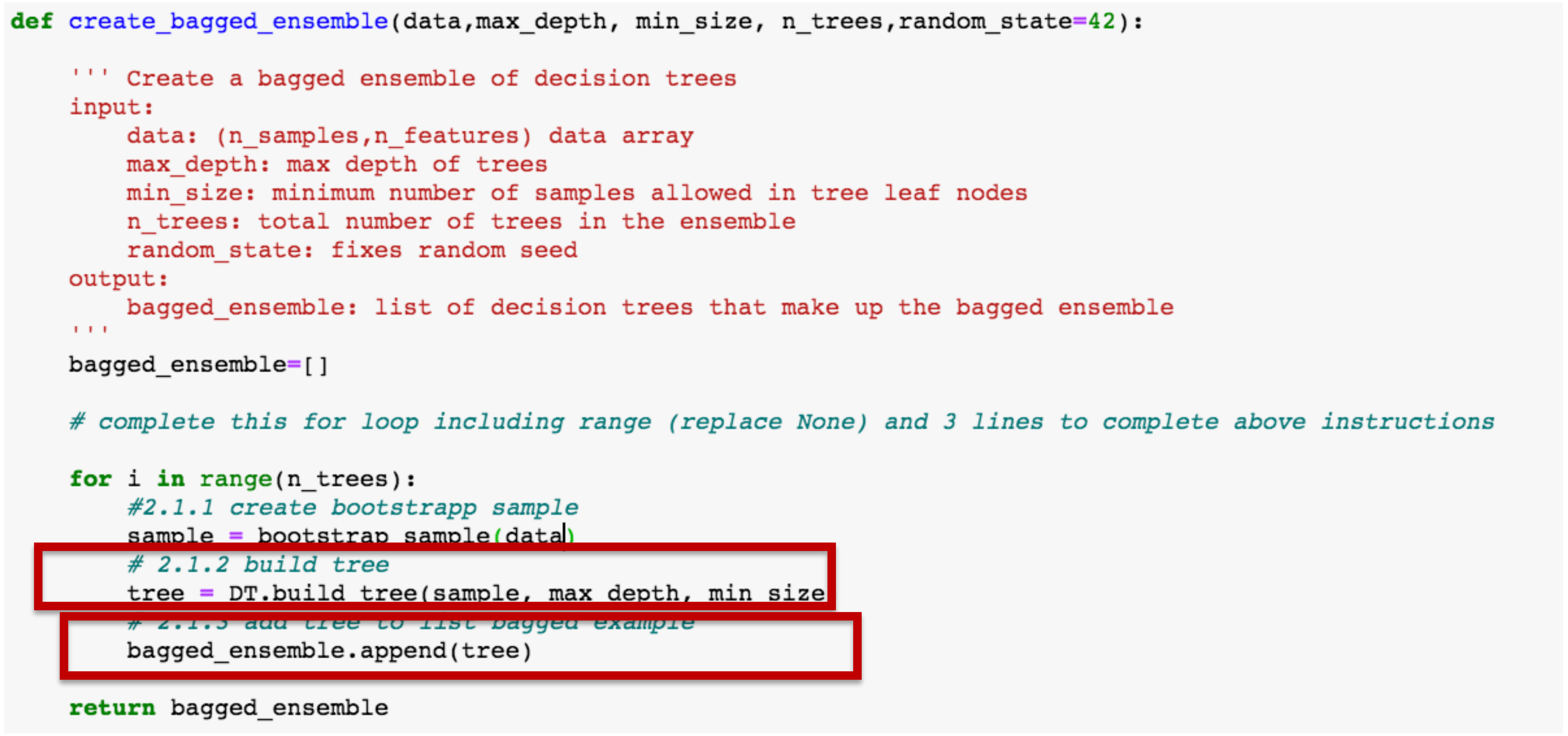


```
def create_bagged_ensemble(data,max_depth, min_size, n_trees,random_state=42):

    ''' Create a bagged ensemble of decision trees
    input:
        data: (n_samples,n_features) data array
        max_depth: max depth of trees
        min_size: minimum number of samples allowed in tree leaf nodes
        n_trees: total number of trees in the ensemble
        random_state: fixes random seed
    output:
        bagged_ensemble: list of decision trees that make up the bagged ensemble
    '''
    bagged_ensemble=[]

    # complete this for loop including range (replace None) and 3 lines to complete above instructions

    for i in range(n_trees):
        #2.1.1 create bootstrapp sample
        sample = bootstrap_sample(data)
        # 2.1.2 build tree
        tree = DT.build_tree(sample, max_depth, min_size)
        # 2.1.3 add tree to list bagged example
        bagged_ensemble.append(tree)

    return bagged_ensemble
```

# Solutions Ex 2 Building a Bagging Classifier

For each tree in the ensemble

2. Create  a decision tree by calling `build_tree`  from the module we created last week
   - Supply arguments!
3. Append the tree to the list `bagged_ensemble`



```python
def create_bagged_ensemble(data,max_depth, min_size, n_trees,random_state=42):

    ''' Create a bagged ensemble of decision trees
    input:
        data: (n_samples,n_features) data array
        max_depth: max depth of trees
        min_size: minimum number of samples allowed in tree leaf nodes
        n_trees: total number of trees in the ensemble
        random_state: fixes random seed
    output:
        bagged_ensemble: list of decision trees that make up the bagged ensemble
    '''
    bagged_ensemble=[]

    # complete this for loop including range (replace None) and 3 lines to complete above instructions

    for i in range(n_trees):
        #2.1.1 create bootstrapp sample
        sample = bootstrap_sample(data)
        # 2.1.2 build tree
        tree = DT.build_tree(sample, max_depth, min_size
        # 2.1.3 add tree to list bagged example
        bagged_ensemble.append(tree)

    return bagged_ensemble
```

# Solutions Ex 2 Building a Bagging Classifier

4. Run create 100 bootstrapped trees, setting max_depth= 3, and min_size= 1,

```
trees=create_bagged_ensemble(dataset,3, 1, 100)
```

5. Complete the predict function to implement majority voting

```
def bagging_predict(trees, testdata):

    predictions=[]
    # loop over all test examples (rows of the data matrix)
    for row in testdata:
        row_predictions=[]
        # loop over all trees
        for  tree in trees:
            # get a prediction for that row (example) and that trees
            # using function predict_row (line 241) of Decision Tree
            tree_prediction=DT.predict_row(tree, row)
            # append the prediction to the list of presdications for that example
            row_predictions.append(tree_prediction)

        # 2.2.1 return the total number of predictions for each class
        count=np.bincount(row_predictions)
        # 2.2.2 return predicted class
        row_label=np.argmax(count)
        # append this to list of predictions for all test examples
        predictions.append( row_label)

    return predictions
```

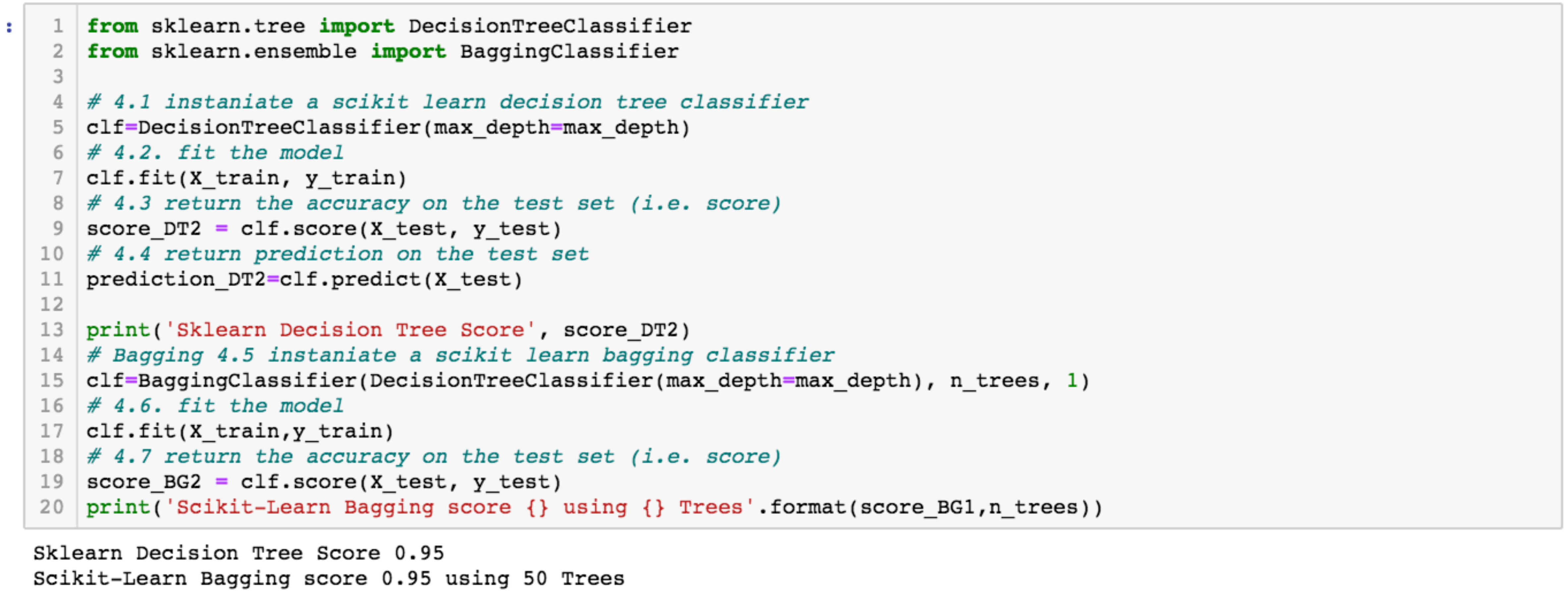
1. np.bincount counts unique values
   - For classes this means it returns a list (len(classes)) with total number of predictions for each class

1. np.armax takes that list and returns index of highest index (class)

# Solutions Ex 4 – comparing against scikit-learn

**As always**

- Instantiate (construct – optionally with arguments)
    - For fair comparison set parameters (max depth) to be the same
- Fit
- Predict/score (strictly you don't need both)

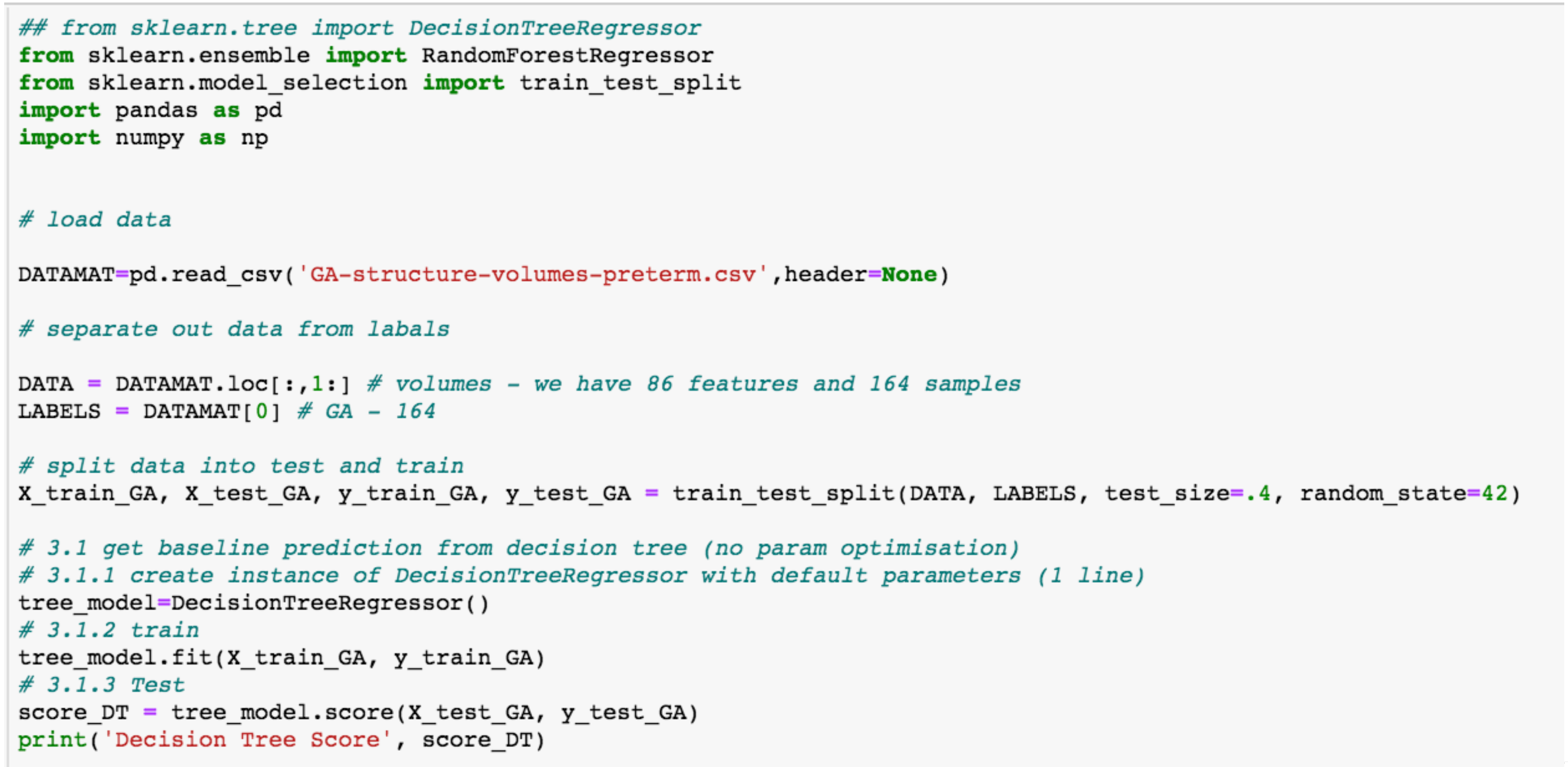


```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

# 4.1 instaniate a scikit learn decision tree classifier
clf=DecisionTreeClassifier(max_depth=max_depth)
# 4.2. fit the model
clf.fit(X_train, y_train)
# 4.3 return the accuracy on the test set (i.e. score)
score_DT2 = clf.score(X_test, y_test)
# 4.4 return prediction on the test set
prediction_DT2=clf.predict(X_test)

print('Sklearn Decision Tree Score', score_DT2)
# Bagging 4.5 instantiate a scikit learn bagging classifier
clf=BaggingClassifier(DecisionTreeClassifier(max_depth=max_depth), n_trees, 1)
# 4.6. fit the model
clf.fit(X_train,y_train)
# 4.7 return the accuracy on the test set (i.e. score)
score_BG2 = clf.score(X_test, y_test)
print('Scikit-Learn Bagging score {} using {} Trees'.format(score_BG1,n_trees))
```

```
Sklearn Decision Tree Score 0.95
Scikit-Learn Bagging score 0.95 using 50 Trees
```

# Solutions Ex 5 Random forests for GA

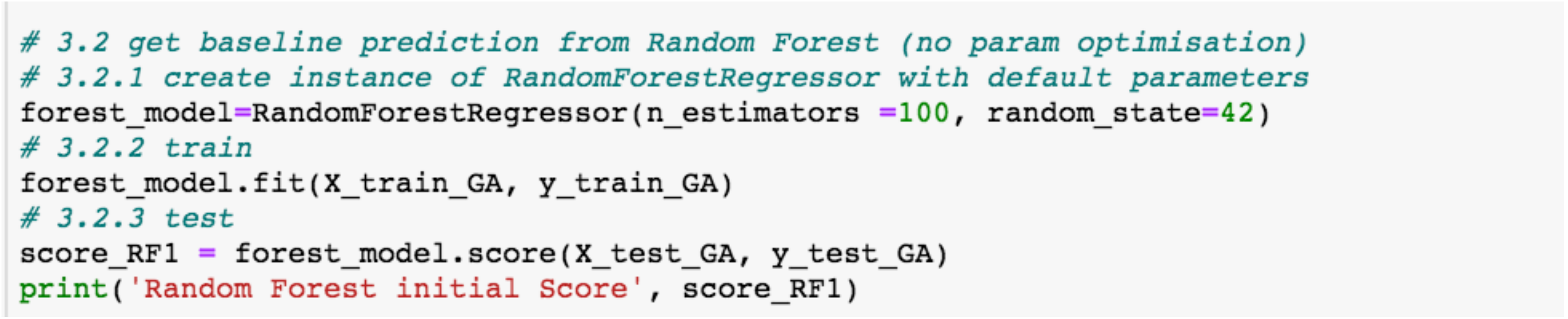
1. Implement decision tree regressor
   - Instantiate (construct – optionally with arguments)
   - Fit
   - Score (score also implements predict)



```
## from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np


# load data

DATAMAT=pd.read_csv('GA-structure-volumes-preterm.csv',header=None)

# separate out data from labals

DATA = DATAMAT.loc[:,1:] # volumes - we have 86 features and 164 samples
LABELS = DATAMAT[0] # GA - 164

# split data into test and train
X_train_GA, X_test_GA, y_train_GA, y_test_GA = train_test_split(DATA, LABELS, test_size=.4, random_state=42)

# 3.1 get baseline prediction from decision tree (no param optimisation)
# 3.1.1 create instance of DecisionTreeRegressor with default parameters (1 line)
tree_model=DecisionTreeRegressor()
# 3.1.2 train
tree_model.fit(X_train_GA, y_train_GA)
# 3.1.3 Test
score_DT = tree_model.score(X_test_GA, y_test_GA)
print('Decision Tree Score', score_DT)
```

# Solutions Ex 5 Random forests for GA

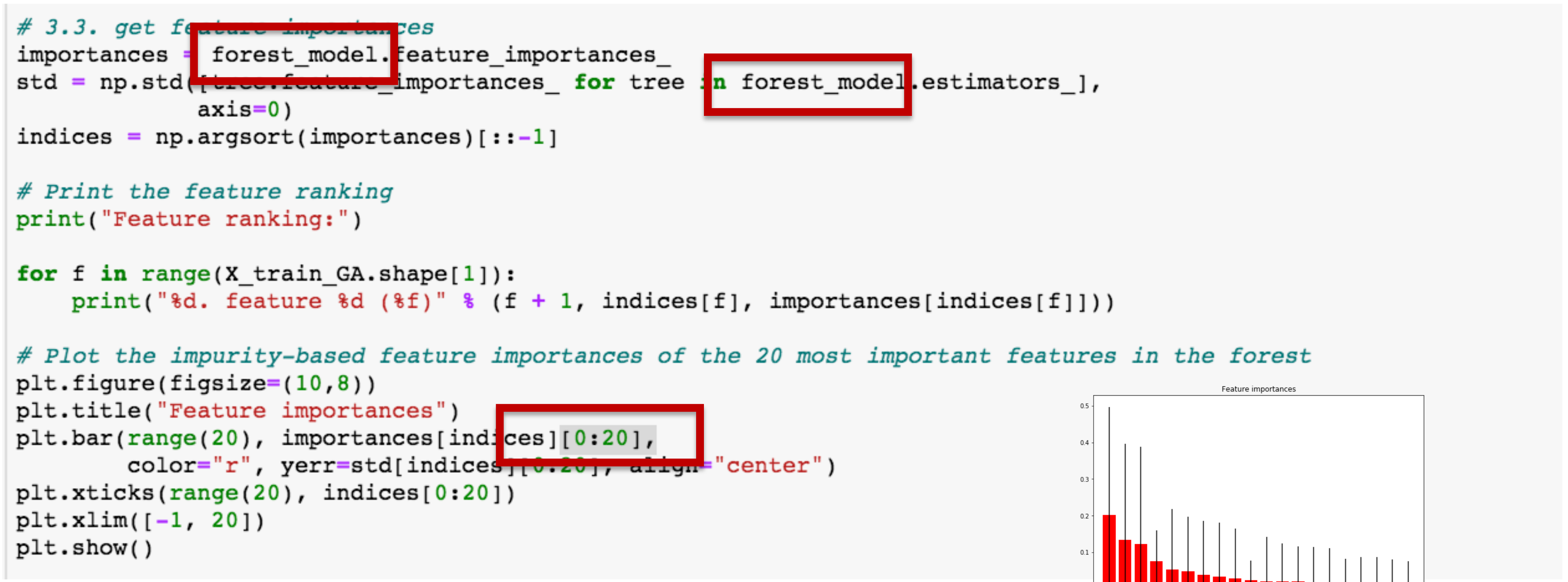
2. Implement forest
   - default n_trees is actually 100 so no need to set
   - random seed means that you get same result every time you run – good for debugging/model comparison/parameter optimization
   - Score fits prediction and calculates score



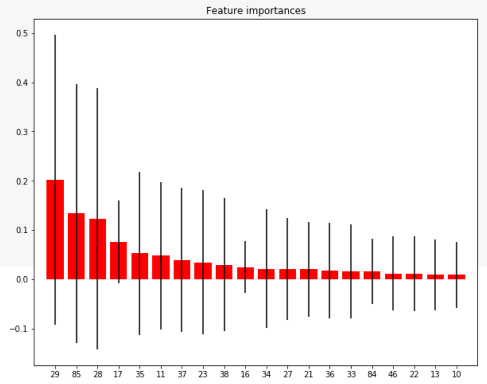
```python
# 3.2 get baseline prediction from Random Forest (no param optimisation)
# 3.2.1 create instance of RandomForestRegressor with default parameters
forest_model=RandomForestRegressor(n_estimators =100, random_state=42)
# 3.2.2 train
forest_model.fit(X_train_GA, y_train_GA)
# 3.2.3 test
score_RF1 = forest_model.score(X_test_GA, y_test_GA)
print('Random Forest initial Score', score_RF1)
```

# Solutions Ex 5 Random forests for GA

2. Plot importances – copied from https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html
   - Only need to change model name -> forest_model
   - I plotted only the first 20

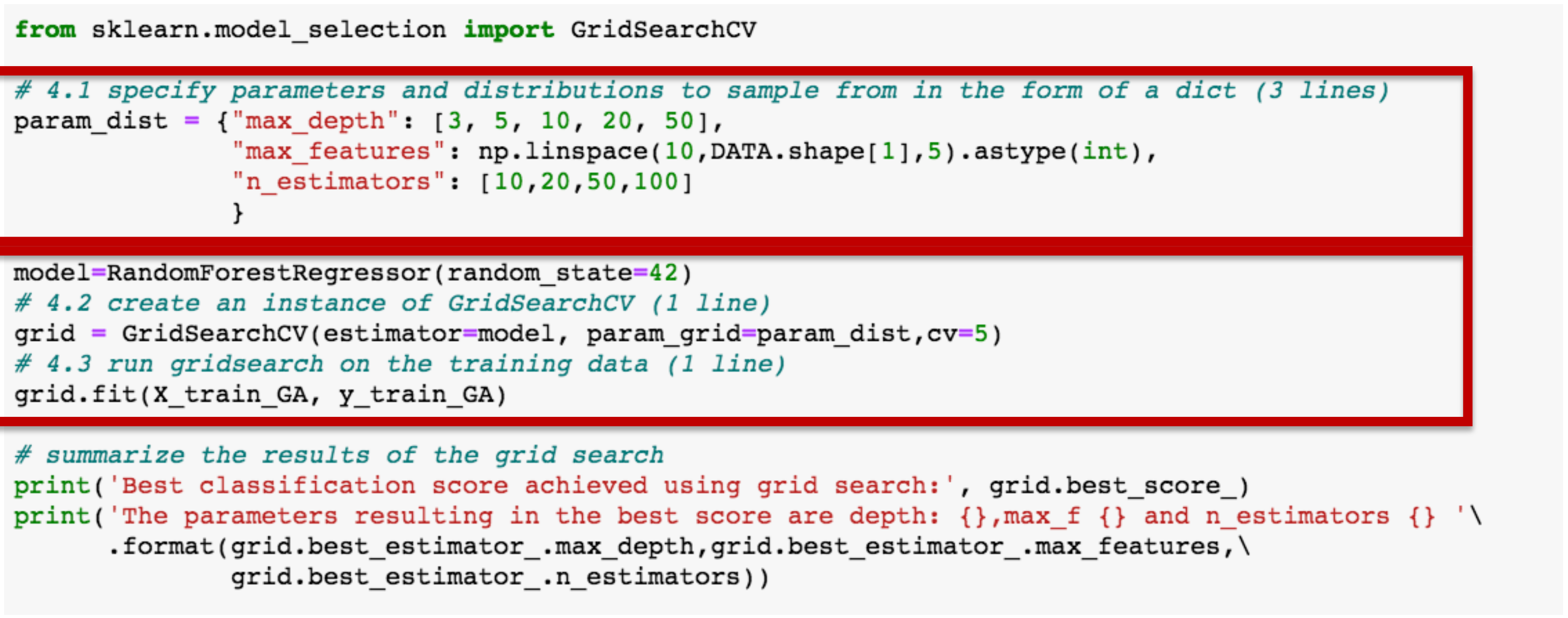


```
# 3.3. get feature importances
importances = forest_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_train_GA.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the impurity-based feature importances of the 20 most important features in the forest
plt.figure(figsize=(10,8))
plt.title("Feature importances")
plt.bar(range(20), importances[indices][0:20],
        color="r", yerr=std[indices][0:20], align="center")
plt.xticks(range(20), indices[0:20])
plt.xlim([-1, 20])
plt.show()
```



# Solutions 6 – Parameter optimization with gridsearch

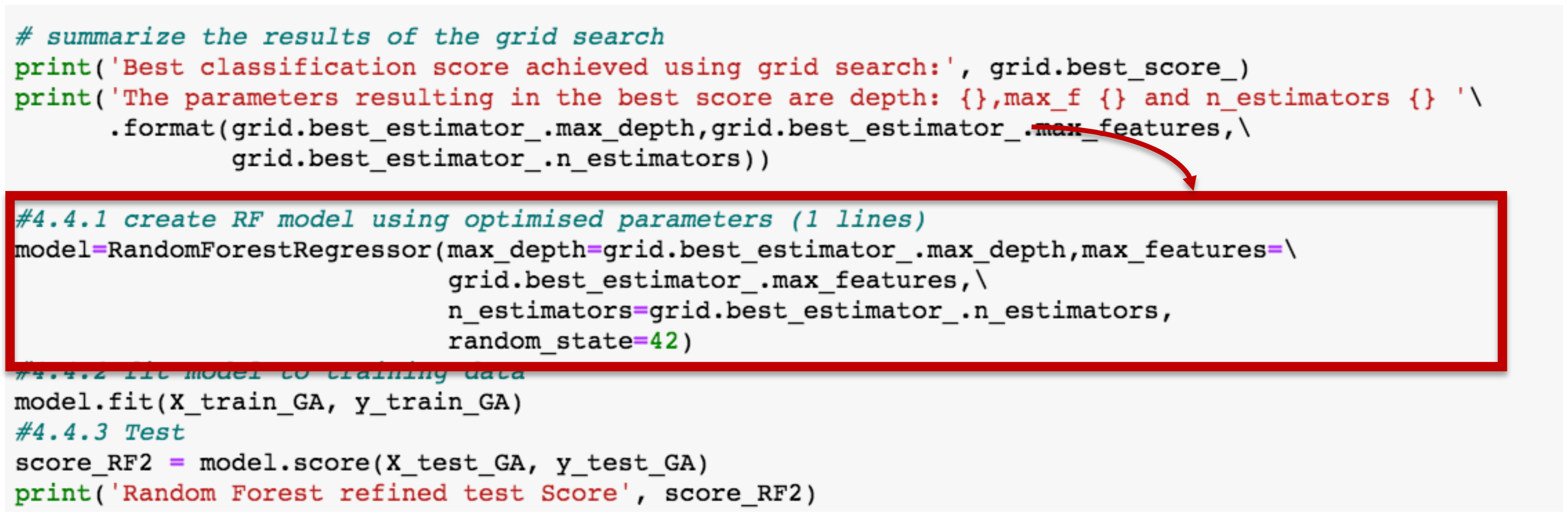
1. Create dictionary
2. create an instance (grid) of GridSearchCV
   1. with RandomForestRegressor() as the model
   2. param_dist as the search grid (with cross validation: cv=5);
   3. then run on the training data



```python
from sklearn.model_selection import GridSearchCV

# 4.1 specify parameters and distributions to sample from in the form of a dict (3 lines)
param_dist = {"max_depth": [3, 5, 10, 20, 50],
              "max_features": np.linspace(10,DATA.shape[1],5).astype(int),
              "n_estimators": [10,20,50,100]
              }

model=RandomForestRegressor(random_state=42)
# 4.2 create an instance of GridSearchCV (1 line)
grid = GridSearchCV(estimator=model, param_grid=param_dist,cv=5)
# 4.3 run gridsearch on the training data (1 line)
grid.fit(X_train_GA, y_train_GA)

# summarize the results of the grid search
print('Best classification score achieved using grid search:', grid.best_score_)
print('The parameters resulting in the best score are depth: {},max_f {} and n_estimators {} '\
      .format(grid.best_estimator_.max_depth,grid.best_estimator_.max_features,\
              grid.best_estimator_.n_estimators))
```

# Solutions 6 – Parameter optimization with gridsearch

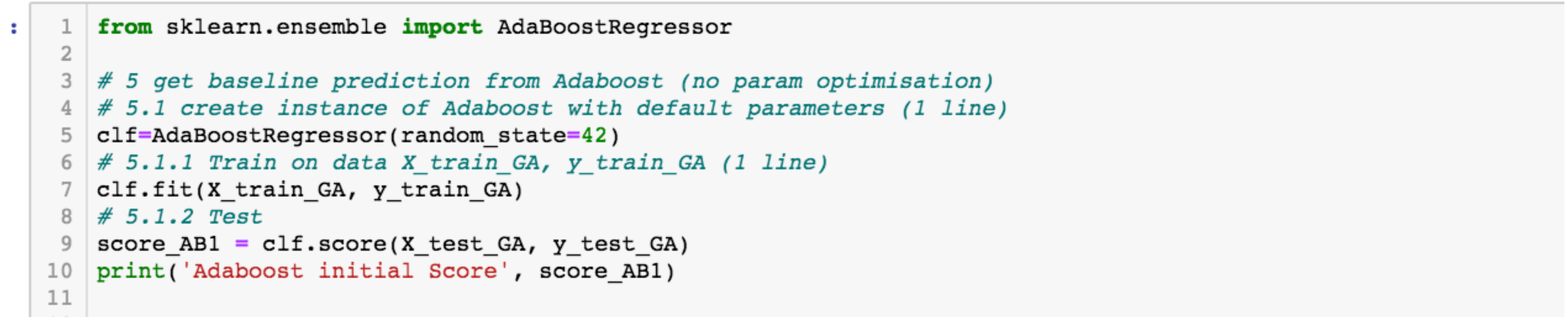
3. Apply the tuned parameters to the model and get a new test prediction
    3. Pass gridCV attributes as arguments to the random forest model



```
# summarize the results of the grid search
print('Best classification score achieved using grid search:', grid.best_score_)
print('The parameters resulting in the best score are depth: {},max_f {} and n_estimators {} '\
      .format(grid.best_estimator_.max_depth,grid.best_estimator_.max_features,\
              grid.best_estimator_.n_estimators))

#4.4.1 create RF model using optimised parameters (1 lines)
model=RandomForestRegressor(max_depth=grid.best_estimator_.max_depth,max_features=\
                            grid.best_estimator_.max_features,\
                            n_estimators=grid.best_estimator_.n_estimators,
                            random_state=42)
#4.4.2 fit model to training data
model.fit(X_train_GA, y_train_GA)
#4.4.3 Test
score_RF2 = model.score(X_test_GA, y_test_GA)
print('Random Forest refined test Score', score_RF2)
```

# Solutions 8 - adaboost

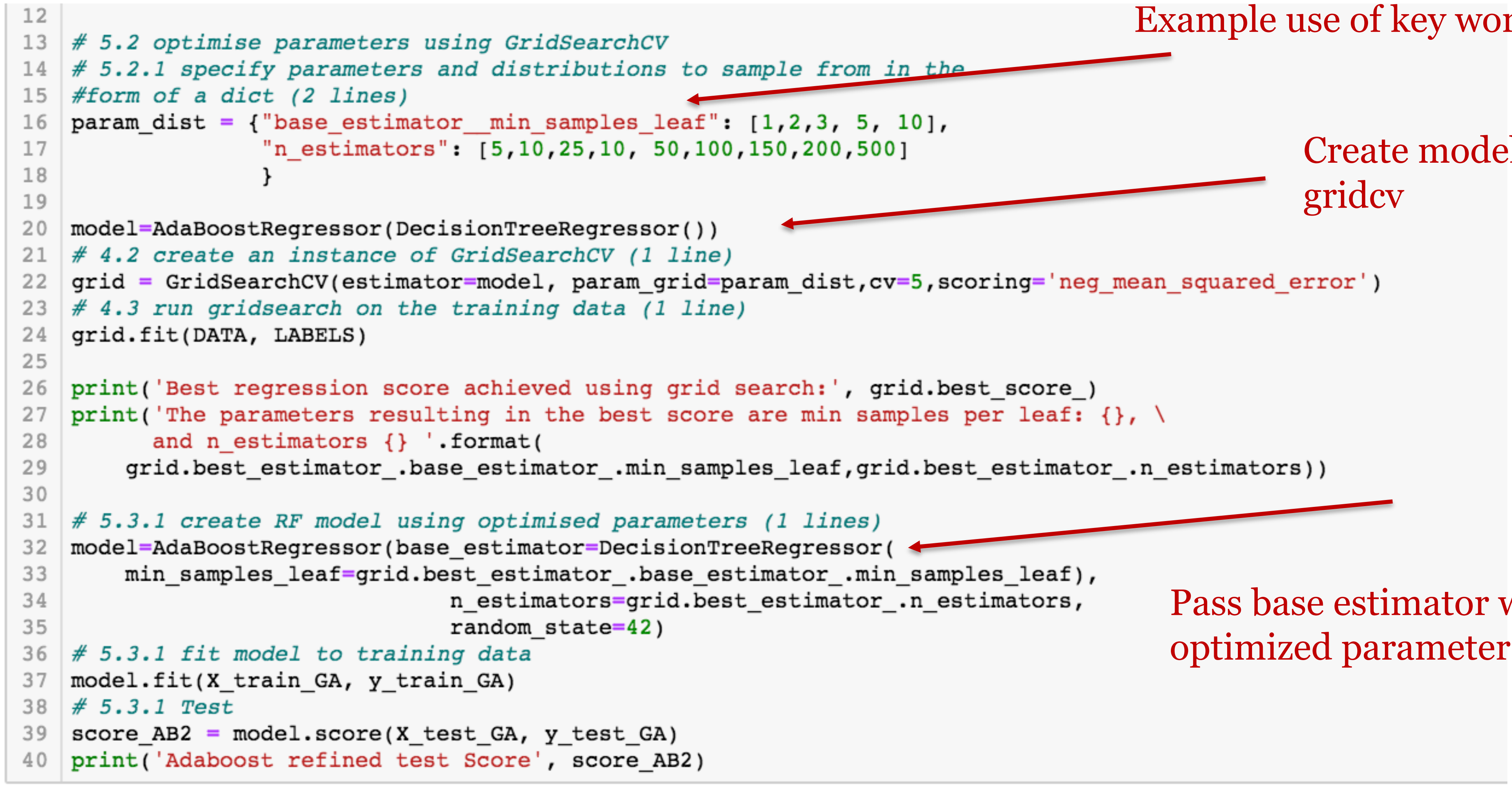
1. Fit with default parameters



```python
from sklearn.ensemble import AdaBoostRegressor

# 5 get baseline prediction from Adaboost (no param optimisation)
# 5.1 create instance of Adaboost with default parameters (1 line)
clf=AdaBoostRegressor(random_state=42)
# 5.1.1 Train on data X_train_GA, y_train_GA (1 line)
clf.fit(X_train_GA, y_train_GA)
# 5.1.2 Test
score_AB1 = clf.score(X_test_GA, y_test_GA)
print('Adaboost initial Score', score_AB1)

```

# Solutions 8 - adaboost

- Fit gridsearch while optimization parameters of base estimator (tree)



```python

# 5.2 optimise parameters using GridSearchCV
# 5.2.1 specify parameters and distributions to sample from in the
#form of a dict (2 lines)
param_dist = {"base_estimator__min_samples_leaf": [1,2,3, 5, 10],
              "n_estimators": [5,10,25,10, 50,100,150,200,500]
              }

model=AdaBoostRegressor(DecisionTreeRegressor())
# 4.2 create an instance of GridSearchCV (1 line)
grid = GridSearchCV(estimator=model, param_grid=param_dist,cv=5,scoring='neg_mean_squared_error')
# 4.3 run gridsearch on the training data (1 line)
grid.fit(DATA, LABELS)

print('Best regression score achieved using grid search:', grid.best_score_)
print('The parameters resulting in the best score are min samples per leaf: {}, \
   and n_estimators {} '.format(
  grid.best_estimator_.base_estimator_.min_samples_leaf,grid.best_estimator_.n_estimators))

# 5.3.1 create RF model using optimised parameters (1 lines)
model=AdaBoostRegressor(base_estimator=DecisionTreeRegressor(
  min_samples_leaf=grid.best_estimator_.base_estimator_.min_samples_leaf),
              n_estimators=grid.best_estimator_.n_estimators,
              random_state=42)
# 5.3.1 fit model to training data
model.fit(X_train_GA, y_train_GA)
# 5.3.1 Test
score_AB2 = model.score(X_test_GA, y_test_GA)
print('Adaboost refined test Score', score_AB2)
```

Example use of key word

Create model and pass to gridcv

Pass base estimator with optimized parameter