Homework #3 Part 2

Neil Jiang, Hayley Huang, Foster Mosden, Icy Wang

Total grade: ________ out of ___100___ points

ATTENTION: HW3 has two parts. Please first complete the Quiz "HW3_Part1" on Canvas. Then, proceed with Part 2 in the following page. You will need to submit (a) a PDF file with your answers and screenshots of Python code snippets and (b) the Python code.

(100 points) [Mining publicly available data] Use Python for this Exercise.

Please use the dataset on breast cancer research from this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data We have worked with this dataset in HW2. The description of the data and attributes can be found at this link: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names . Each record of the data set represents a different case of breast cancer. Each case is described with 30 real-valued attributes: attribute 1 represents case id, attributes 3-32 represent various physiological characteristics, and attribute 2 represents the type (benign or malignant). If the dataset has records with missing values, you can filter out these records using Python. Alternatively, if the data set has missing values, you could infer the missing values.

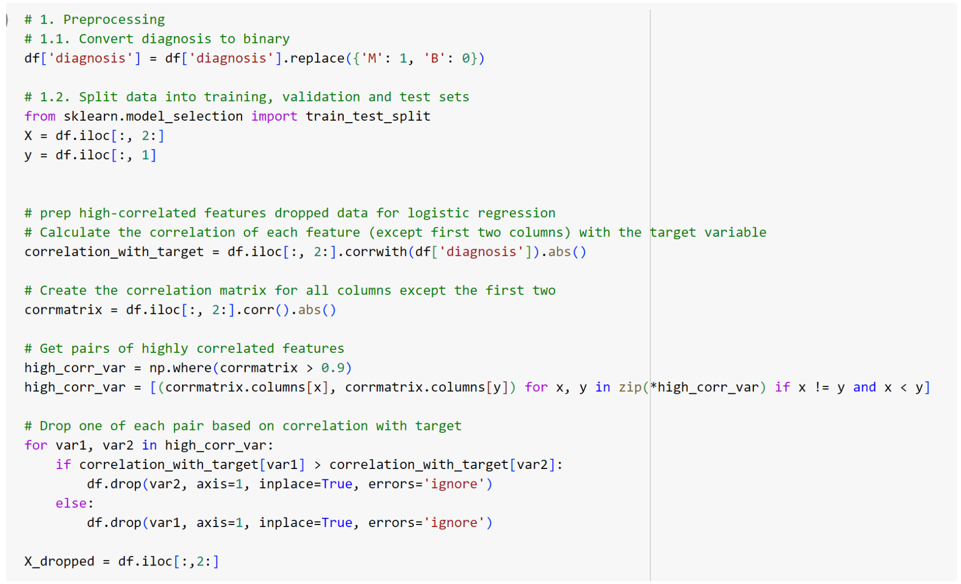
[We have seen this data before – No need to explore the data for this exercise]

Most codes adapted from Fall23_BusinessAnalytics_ModelEvaluation.ipynb by Prof. Vilma Todri.

a) **We would like to perform a predictive modeling analysis on this same dataset using the a) decision tree, b) the k-NN technique and c) the logistic regression technique. Using the nested cross-validation technique, try to optimize the parameters of your classifiers in order to improve the performance of your classifiers (i.e., f1-score) as much as possible. Please make sure to always use a random state of "42" whenever applicable. What are your optimal parameters and what is the corresponding performance of these classifiers? Please provide screenshots of your code and explain the process you have followed.**

a.1 Data Preparation

For part a, we have prepared two different versions of the original dataset. For decision tree and k-NN classification where multicollinearity is not of great concern while multicollinearity messes with the logistic regression model. As a result, when conducting logistic regression, we opt to remove the variable that has a higher correlation with the target variable when two features are highly correlated, defined as an absolute correlation coefficient greater than 0.9.



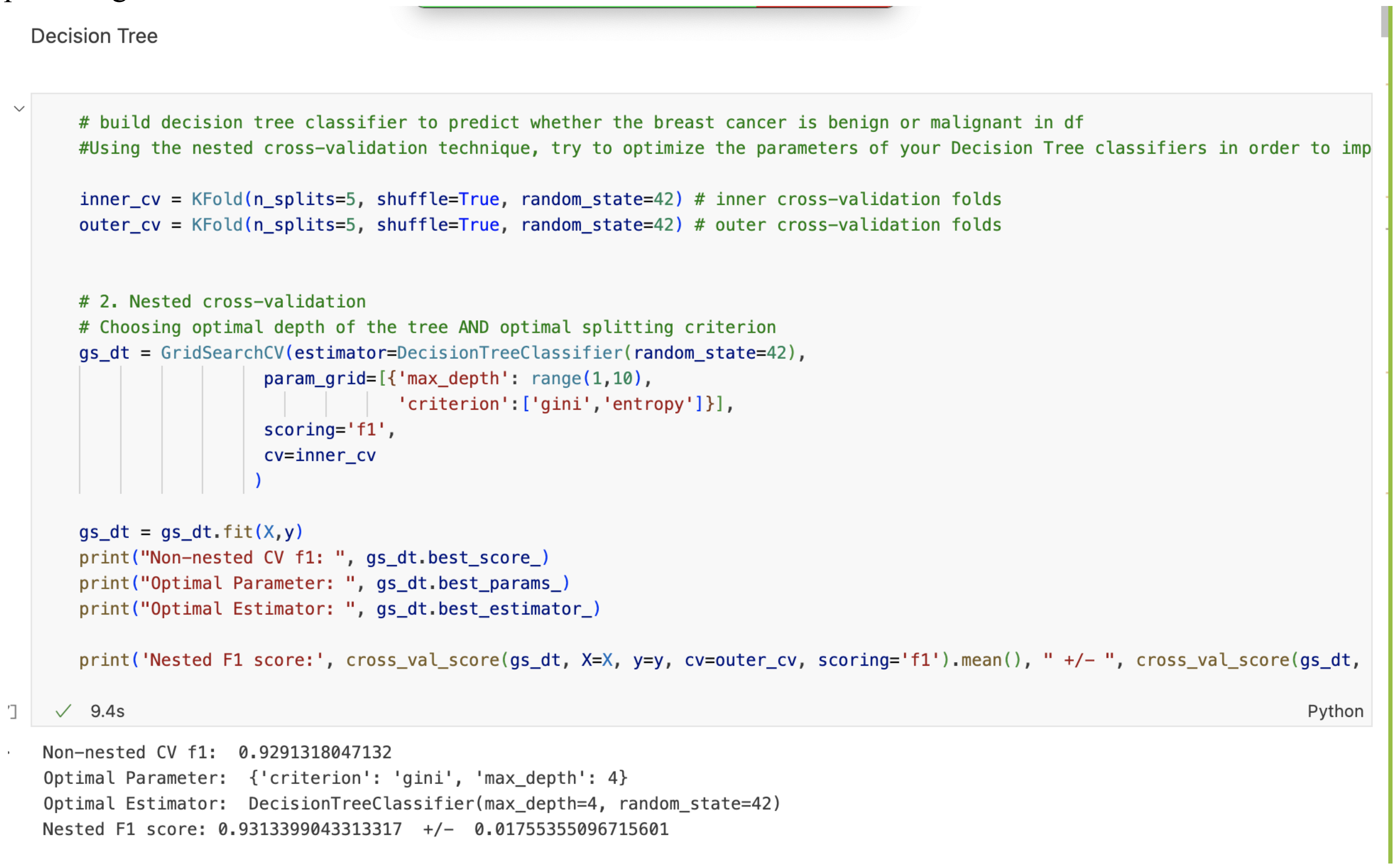
```
# 1. Preprocessing
# 1.1. Convert diagnosis to binary
df['diagnosis'] = df['diagnosis'].replace({'M': 1, 'B': 0})

# 1.2. Split data into training, validation and test sets
from sklearn.model_selection import train_test_split
X = df.iloc[:, 2:]
y = df.iloc[:, 1]


# prep high-correlated features dropped data for logistic regression
# Calculate the correlation of each feature (except first two columns) with the target variable
correlation_with_target = df.iloc[:, 2:].corrwith(df['diagnosis']).abs()

# Create the correlation matrix for all columns except the first two
corrmatrix = df.iloc[:, 2:].corr().abs()

# Get pairs of highly correlated features
high_corr_var = np.where(corrmatrix > 0.9)
high_corr_var = [(corrmatrix.columns[x], corrmatrix.columns[y]) for x, y in zip(*high_corr_var) if x != y and x < y]

# Drop one of each pair based on correlation with target
for var1, var2 in high_corr_var:
    if correlation_with_target[var1] > correlation_with_target[var2]:
        df.drop(var2, axis=1, inplace=True, errors='ignore')
    else:
        df.drop(var1, axis=1, inplace=True, errors='ignore')

X_dropped = df.iloc[:,2:]
```

a.2 Metric selection and justification

In cancer detection, both false negatives (missing a cancer case) and false positives (incorrectly labeling a non-cancer case as cancerous) have significant costs. While recall is focused on minimizing false negatives, it doesn't take into account the false positives, which can lead to unnecessary medical tests and emotional distress for patients. Using F1 score instead of just recall provides a more balanced view because it considers both precision and recall in its calculation. This is important for giving a fuller picture of model performance.

In part c we also included ROC and AUC metric, which are more focused on the model's ability to discriminate between classes, while the F1 score is concerned with the model's accuracy for each class. Our assessment of the models uses F1 in conjunction with ROC and AUC, allowing for a more nuanced and comprehensive evaluation of the model's performance in cancer detection scenarios.

a.3 Model and model evaluations
a.3.1 Decision Tree:
We performed nested cross-validation on a Decision Tree classifier to predict the type of breast cancer (benign or malignant). It uses two 5-fold cross-validation sets: `inner_cv` for hyperparameter tuning and `outer_cv` for performance evaluation. GridSearchCV searches for the optimal tree depth and splitting criterion, either 'gini' or 'entropy', aiming to maximize the F1 score. The best hyperparameters are identified and used to evaluate the model's performance in the outer loop, providing both the mean and standard deviation of the F1 scores.



Decision Tree

```
# build decision tree classifier to predict whether the breast cancer is benign or malignant in df
#Using the nested cross-validation technique, try to optimize the parameters of your Decision Tree classifiers in order to imp

inner_cv = KFold(n_splits=5, shuffle=True, random_state=42) # inner cross-validation folds
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42) # outer cross-validation folds


# 2. Nested cross-validation
# Choosing optimal depth of the tree AND optimal splitting criterion
gs_dt = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                     param_grid=[{'max_depth': range(1,10),
                                  'criterion':['gini','entropy']}],
                     scoring='f1',
                     cv=inner_cv
                    )

gs_dt = gs_dt.fit(X,y)
print("Non-nested CV f1: ", gs_dt.best_score_)
print("Optimal Parameter: ", gs_dt.best_params_)
print("Optimal Estimator: ", gs_dt.best_estimator_)

print('Nested F1 score:', cross_val_score(gs_dt, X=X, y=y, cv=outer_cv, scoring='f1').mean(), " +/- ", cross_val_score(gs_dt,
```

✓ 9.4s Python

```
Non-nested CV f1:  0.9291318047132
Optimal Parameter:  {'criterion': 'gini', 'max_depth': 4}
Optimal Estimator:  DecisionTreeClassifier(max_depth=4, random_state=42)
Nested F1 score: 0.9313399043313317  +/-  0.01755355096715601
```

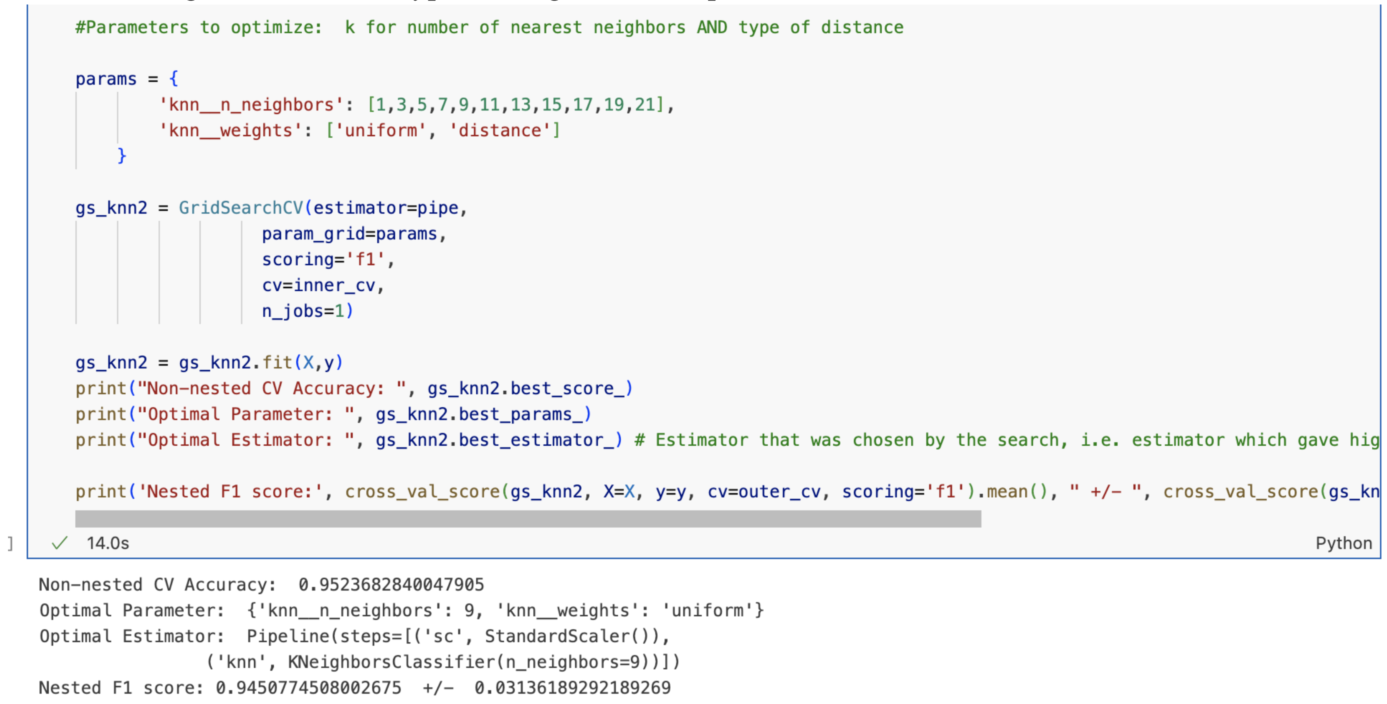
The printed results are the best parameter and the according mean f1 score results according to inner and outer cross validation.

a.3.2
K-nn technique:

Before the inner and outer cross validation like in the Decision Tree method, we first adjusted the scale of the features.

```
#Normalize Data
pipe = Pipeline([
        ('sc', StandardScaler()),
        ('knn', KNeighborsClassifier(p=2,
                                     metric='minkowski'))
      ])
```

After feature normalization, we conducted the K-nn inner cross validation for assessing the optimal number of neighbors and what type of weights for this particular dataset.

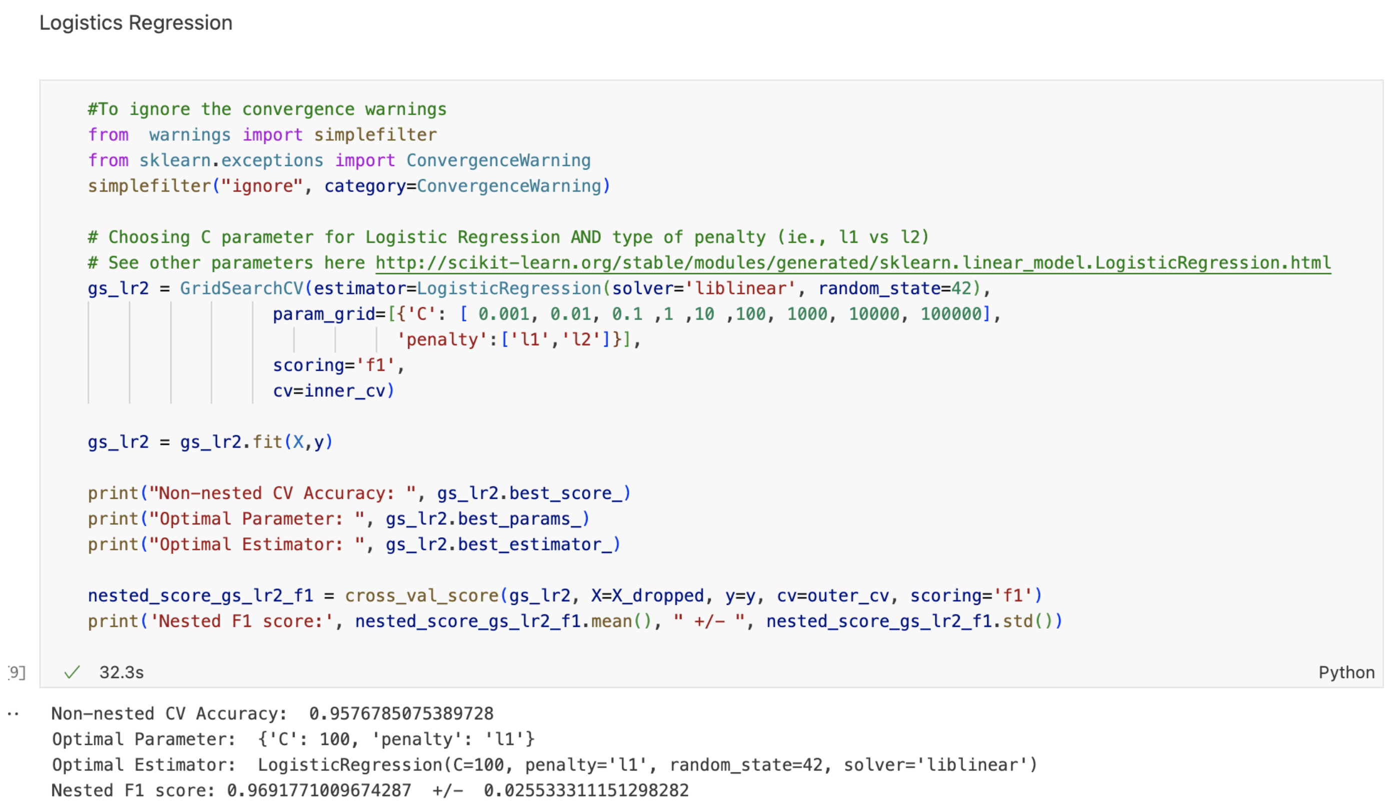


```
#Parameters to optimize:  k for number of nearest neighbors AND type of distance

params = {
        'knn__n_neighbors': [1,3,5,7,9,11,13,15,17,19,21],
        'knn__weights': ['uniform', 'distance']
    }

gs_knn2 = GridSearchCV(estimator=pipe,
                        param_grid=params,
                        scoring='f1',
                        cv=inner_cv,
                        n_jobs=1)

gs_knn2 = gs_knn2.fit(X,y)
print("Non-nested CV Accuracy: ", gs_knn2.best_score_)
print("Optimal Parameter: ", gs_knn2.best_params_)
print("Optimal Estimator: ", gs_knn2.best_estimator_) # Estimator that was chosen by the search, i.e. estimator which gave hig

print('Nested F1 score:', cross_val_score(gs_knn2, X=X, y=y, cv=outer_cv, scoring='f1').mean(), " +/- ", cross_val_score(gs_kn
```

```
Non-nested CV Accuracy:  0.9523682840047905
Optimal Parameter:  {'knn__n_neighbors': 9, 'knn__weights': 'uniform'}
Optimal Estimator:  Pipeline(steps=[('sc', StandardScaler()),
                ('knn', KNeighborsClassifier(n_neighbors=9))])
Nested F1 score: 0.9450774508002675  +/-  0.03136189292189269
```

Outer cross validation similar to the decision tree section.

a.3.3
Logistic regression:
We conducted the K-nn inner validation for assessing the optimal c as a strength for regularization penalty and what type of penalty to apply.


Logistics Regression

```
#To ignore the convergence warnings
from  warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)

# Choosing C parameter for Logistic Regression AND type of penalty (ie., l1 vs l2)
# See other parameters here http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
gs_lr2 = GridSearchCV(estimator=LogisticRegression(solver='liblinear', random_state=42),
                    param_grid=[{'C': [ 0.001, 0.01, 0.1 ,1 ,10 ,100, 1000, 10000, 100000],
                                'penalty':['l1','l2']}],
                    scoring='f1',
                    cv=inner_cv)

gs_lr2 = gs_lr2.fit(X,y)

print("Non-nested CV Accuracy: ", gs_lr2.best_score_)
print("Optimal Parameter: ", gs_lr2.best_params_)
print("Optimal Estimator: ", gs_lr2.best_estimator_)

nested_score_gs_lr2_f1 = cross_val_score(gs_lr2, X=X_dropped, y=y, cv=outer_cv, scoring='f1')
print('Nested F1 score:', nested_score_gs_lr2_f1.mean(), " +/- ", nested_score_gs_lr2_f1.std())
```

9] ✓ 32.3s Python

```
Non-nested CV Accuracy:  0.9576785075389728
Optimal Parameter:  {'C': 100, 'penalty': 'l1'}
Optimal Estimator:  LogisticRegression(C=100, penalty='l1', random_state=42, solver='liblinear')
Nested F1 score: 0.9691771009674287  +/-  0.025533311151298282
```

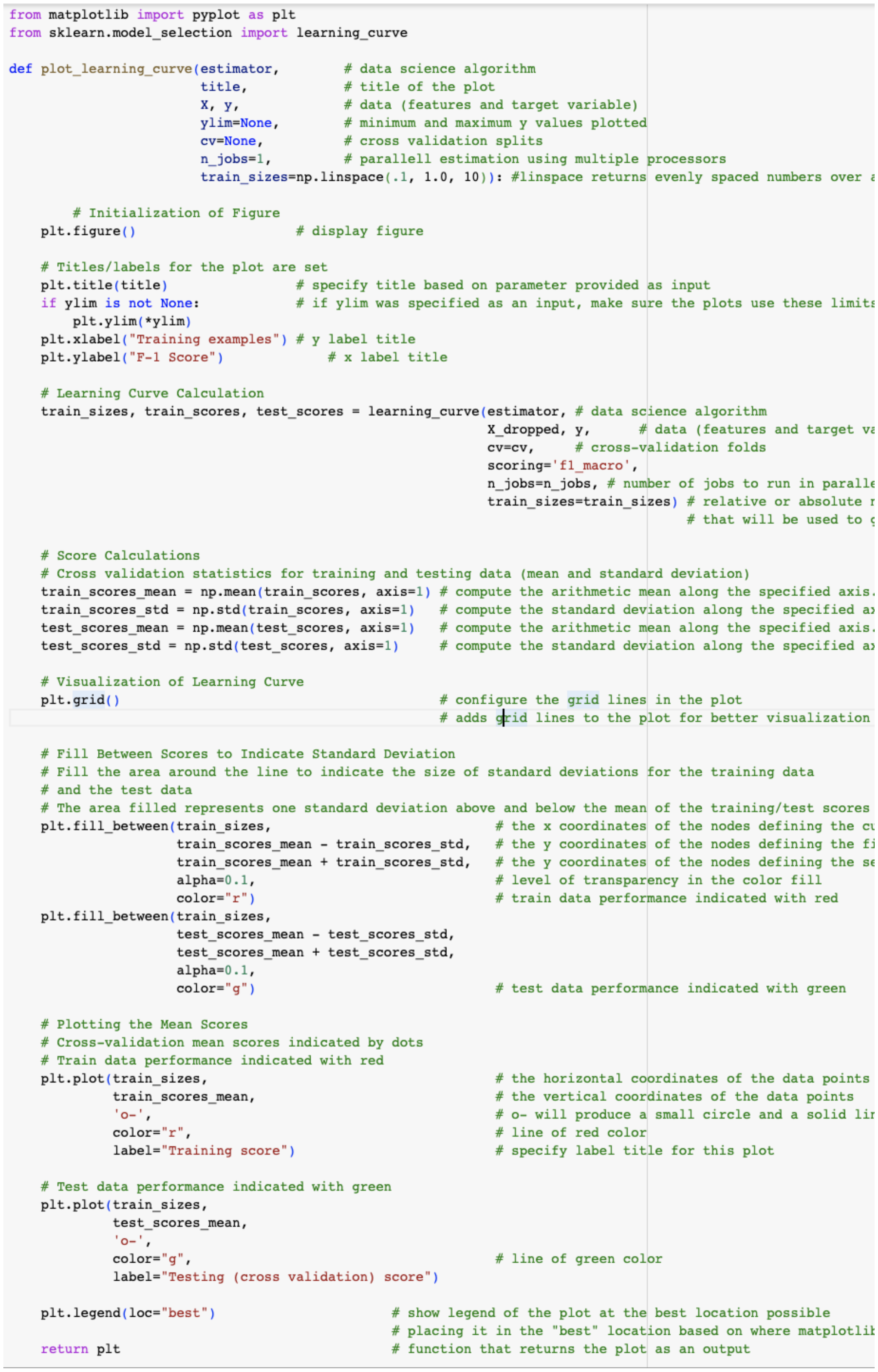
Outer cross validation similar to the decision tree section.

a.4 Evaluation Results
In the case of the **Decision Tree model**, the optimal parameter combination that yielded the highest nested F1 score of 0.9313399043313317 +/- 0.01755355096715601 is set with 'criterion': '**gini**' and **'max_depth': 4**. Here, 'criterion' refers to the function used to measure the quality of a split, with 'gini' aiming to minimize the probability of misclassification. 'Max_depth': 4 signifies that the tree is limited to a depth of three layers, preventing it from becoming overly complex and overfitting.

For the **K-Nearest Neighbors (K-NN) model**, the optimal parameters are **'knn__n_neighbors': 9** and **'knn__weights': 'uniform'**. These produced a nested F1 score of 0.9450774508002675 +/- 0.03136189292189269. Here, 'knn__n_neighbors': 11 means that the algorithm considers the 11 nearest data points when making a classification, and 'knn__weights': 'uniform' indicates that all neighbors are equally weighted in the voting process.

Lastly, the **Logistic Regression model** performs optimally with a regularization setting of {**'C': 100, 'penalty': 'l1'**}, resulting in a nested F1 score of 0.9450774508002675 +/- 0.03136189292189269. In this setting, 'C': 1000 refers to the inverse of regularization strength, indicating less regularization and therefore a more flexible model. The 'penalty': 'l1' signifies that L1 regularization is used, which can drive some feature coefficients to zero, effectively performing feature selection.

b) **Build and visualize a learning curve for the logistic regression technique (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.**

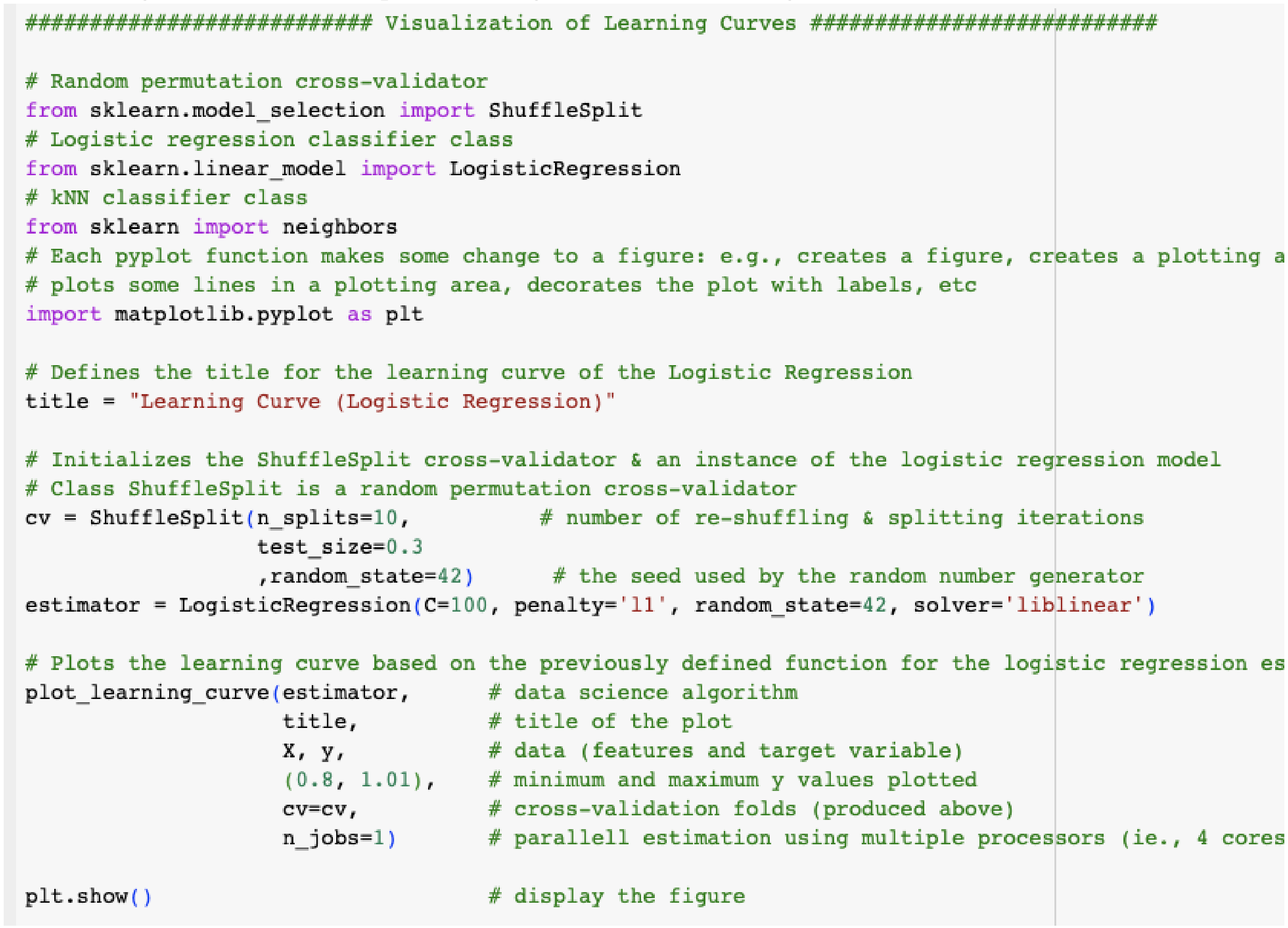


```python
from matplotlib import pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator,        # data science algorithm
                        title,            # title of the plot
                        X, y,             # data (features and target variable)
                        ylim=None,        # minimum and maximum y values plotted
                        cv=None,          # cross validation splits
                        n_jobs=1,         # parallell estimation using multiple processors
                        train_sizes=np.linspace(.1, 1.0, 10)): #linspace returns evenly spaced numbers over a

        # Initialization of Figure
    plt.figure()                    # display figure

    # Titles/labels for the plot are set
    plt.title(title)                # specify title based on parameter provided as input
    if ylim is not None:            # if ylim was specified as an input, make sure the plots use these limits
        plt.ylim(*ylim)
    plt.xlabel("Training examples") # y label title
    plt.ylabel("F-1 Score")             # x label title

    # Learning Curve Calculation
    train_sizes, train_scores, test_scores = learning_curve(estimator, # data science algorithm
                                                            X_dropped, y,      # data (features and target va
                                                            cv=cv,     # cross-validation folds
                                                            scoring='f1_macro',
                                                            n_jobs=n_jobs, # number of jobs to run in paralle
                                                            train_sizes=train_sizes) # relative or absolute r
                                                                         # that will be used to g

    # Score Calculations
    # Cross validation statistics for training and testing data (mean and standard deviation)
    train_scores_mean = np.mean(train_scores, axis=1) # compute the arithmetic mean along the specified axis.
    train_scores_std = np.std(train_scores, axis=1)   # compute the standard deviation along the specified ax
    test_scores_mean = np.mean(test_scores, axis=1)   # compute the arithmetic mean along the specified axis.
    test_scores_std = np.std(test_scores, axis=1)     # compute the standard deviation along the specified ax

    # Visualization of Learning Curve
    plt.grid()                                        # configure the grid lines in the plot
                                                      # adds grid lines to the plot for better visualization

    # Fill Between Scores to Indicate Standard Deviation
    # Fill the area around the line to indicate the size of standard deviations for the training data
    # and the test data
    # The area filled represents one standard deviation above and below the mean of the training/test scores
    plt.fill_between(train_sizes,                           # the x coordinates of the nodes defining the cu
                     train_scores_mean - train_scores_std,  # the y coordinates of the nodes defining the fi
                     train_scores_mean + train_scores_std,  # the y coordinates of the nodes defining the se
                     alpha=0.1,                             # level of transparency in the color fill
                     color="r")                             # train data performance indicated with red
    plt.fill_between(train_sizes,
                     test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std,
                     alpha=0.1,
                     color="g")                             # test data performance indicated with green

    # Plotting the Mean Scores
    # Cross-validation mean scores indicated by dots
    # Train data performance indicated with red
    plt.plot(train_sizes,                                   # the horizontal coordinates of the data points
             train_scores_mean,                             # the vertical coordinates of the data points
             'o-',                                          # o- will produce a small circle and a solid lin
             color="r",                                     # line of red color
             label="Training score")                        # specify label title for this plot

    # Test data performance indicated with green
    plt.plot(train_sizes,
             test_scores_mean,
             'o-',
             color="g",                                     # line of green color
             label="Testing (cross validation) score")

    plt.legend(loc="best")                      # show legend of the plot at the best location possible
                                                # placing it in the "best" location based on where matplotlib
    return plt                                  # function that returns the plot as an output
```

After importing necessary library, we defined the plot_learning_curve function:

- Initiated a matplotlib format to store the to-be-plot learning curve

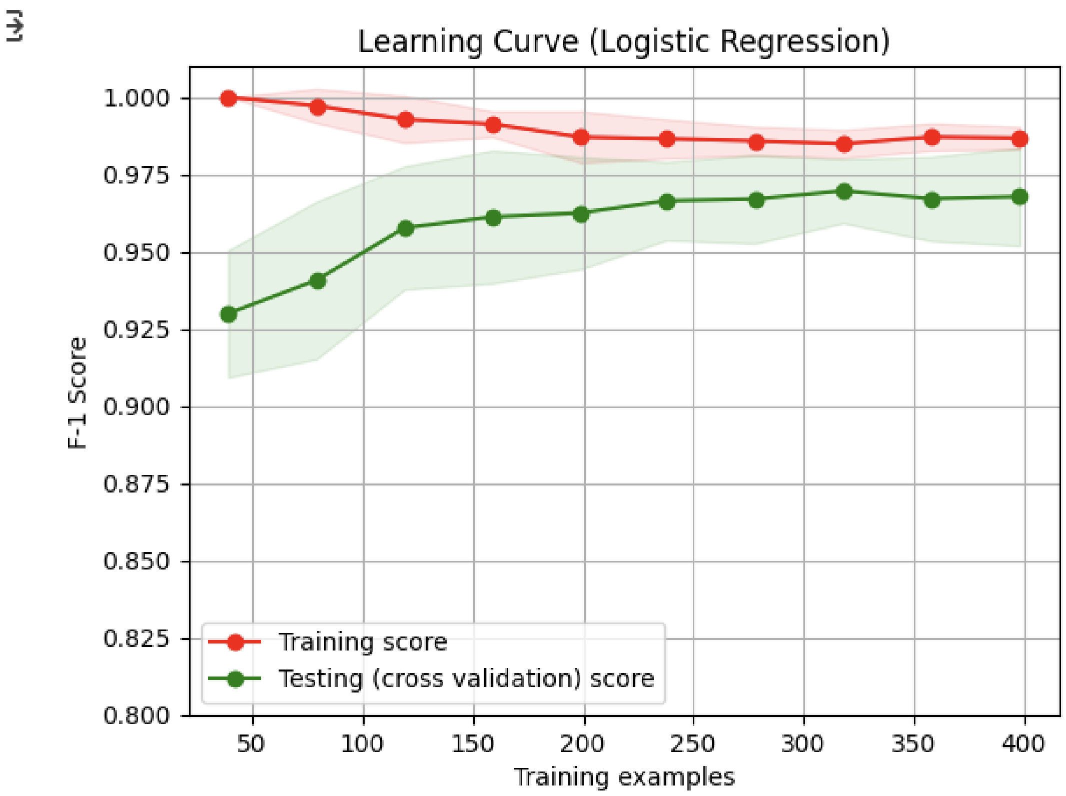
- The learning curve (Cross-validation scores for training and testing data) is calculated using the learning_curve function from scikit-learn that determines cross-validated training and test scores for different training set sizes
    - The train_sizes parameter in the learning_curve function, we can control the partial or relative number of training samples that are used to generate the learning curves. Here, we set train_sizes = np.linspace(0.1, 1.0 , 10) to use 10 evenly spaced relative intervals for the training set sizes.
    - train_scores = scores on training sets (array)
    - test_scores = scores on test set (array)
- Mean and standard deviation of these scores are computed.
- The learning curve is visualized with filled areas indicating standard deviations and dots representing the mean scores for both training and cross-validation.
- A legend is added to the plot to distinguish between training and cross-validation scores.



```
########################### Visualization of Learning Curves ###########################

# Random permutation cross-validator
from sklearn.model_selection import ShuffleSplit
# Logistic regression classifier class
from sklearn.linear_model import LogisticRegression
# kNN classifier class
from sklearn import neighbors
# Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting a
# plots some lines in a plotting area, decorates the plot with labels, etc
import matplotlib.pyplot as plt

# Defines the title for the learning curve of the Logistic Regression
title = "Learning Curve (Logistic Regression)"

# Initializes the ShuffleSplit cross-validator & an instance of the logistic regression model
# Class ShuffleSplit is a random permutation cross-validator
cv = ShuffleSplit(n_splits=10,          # number of re-shuffling & splitting iterations
                  test_size=0.3
                  ,random_state=42)      # the seed used by the random number generator
estimator = LogisticRegression(C=100, penalty='l1', random_state=42, solver='liblinear')

# Plots the learning curve based on the previously defined function for the logistic regression es
plot_learning_curve(estimator,      # data science algorithm
                    title,          # title of the plot
                    X, y,           # data (features and target variable)
                    (0.8, 1.01),    # minimum and maximum y values plotted
                    cv=cv,          # cross-validation folds (produced above)
                    n_jobs=1)       # parallell estimation using multiple processors (ie., 4 cores

plt.show()                          # display the figure
```

Plotting our specific learning curve:

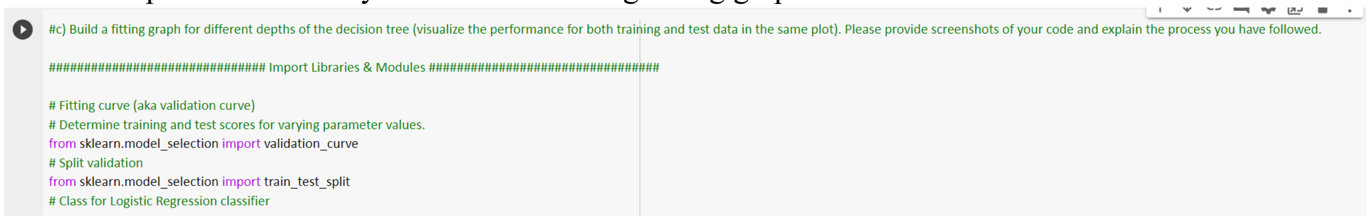
- Used ShuffleSplit as the cross-validator, which generates random permutations of the dataset for each iteration. It's set to perform 10 splits with a test size of 30%.
- Initialize the logistic regression estimator corresponding to the result of hyperparameter tuning in previous question
- Call the plot_learning_curve function defined previously to generate the learning curve for the logistic regression model.
- Display the plot



Overall, the learning curve helps to understand how the model's performance varies as more data is used for training. Initially when the training size is small (under 100), the model tends to overfit the training data, resulting in extremely good in-sample performance but bad cross-validation performance. As the training data size increases, the model starts to have both better and more consistent generalization performance (higher mean and lower SD of cross validation score).

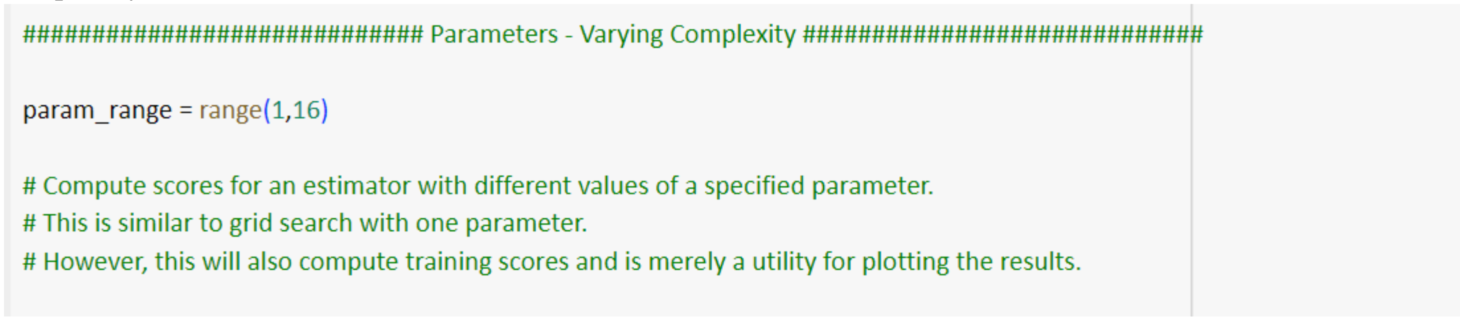
c) **Build a fitting graph for different depths of the decision tree (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.**

First we import the necessary libraries of building fitting graphs.



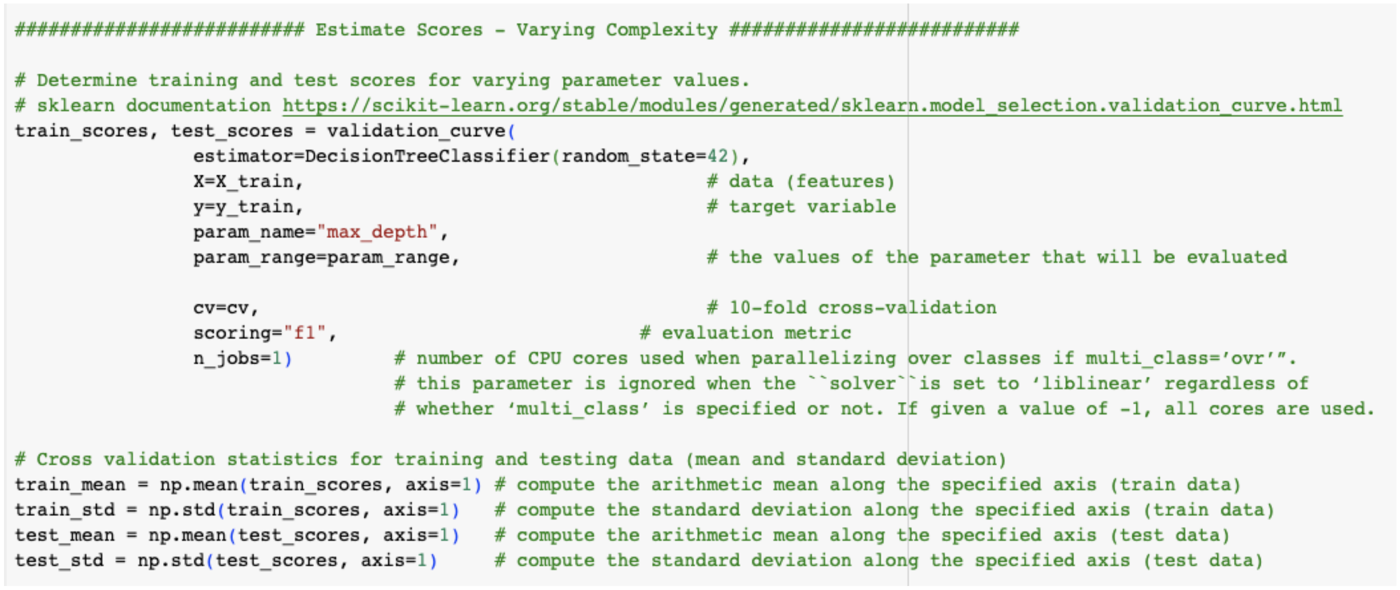
```
#c) Build a fitting graph for different depths of the decision tree (visualize the performance for both training and test data in the same plot). Please provide screenshots of your code and explain the process you have followed.

############################### Import Libraries & Modules ################################

# Fitting curve (aka validation curve)
# Determine training and test scores for varying parameter values.
from sklearn.model_selection import validation_curve
# Split validation
from sklearn.model_selection import train_test_split
# Class for Logistic Regression classifier
```

Next, we specify 16 possible parameter values for max_depth, the key component deciding the complexity of decision tree models.



```
############################### Parameters - Varying Complexity ###############################


param_range = range(1,16)

# Compute scores for an estimator with different values of a specified parameter.
# This is similar to grid search with one parameter.
# However, this will also compute training scores and is merely a utility for plotting the results.
```

In the code below, we performed model validation and evaluation using 10 fold cross-validation to assess the performance of a Decision Tree Classifier with varying values of the max_depth hyperparameter.
In this case, we are dealing with the classification of cancer, it is important to differentiate different types of mistakes the classifier makes, so we chose f1 score as our evaluation matrix.



```
######################### Estimate Scores - Varying Complexity #########################

# Determine training and test scores for varying parameter values.
# sklearn documentation https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.validation_curve.html
train_scores, test_scores = validation_curve(
                estimator=DecisionTreeClassifier(random_state=42),
                X=X_train,                              # data (features)
                y=y_train,                              # target variable
                param_name="max_depth",
                param_range=param_range,                # the values of the parameter that will be evaluated

                cv=cv,                                  # 10-fold cross-validation
                scoring="f1",                     # evaluation metric
                n_jobs=1)         # number of CPU cores used when parallelizing over classes if multi_class='ovr'".
                                  # this parameter is ignored when the ``solver``is set to 'liblinear' regardless of
                                  # whether 'multi_class' is specified or not. If given a value of -1, all cores are used.

# Cross validation statistics for training and testing data (mean and standard deviation)
train_mean = np.mean(train_scores, axis=1) # compute the arithmetic mean along the specified axis (train data)
train_std = np.std(train_scores, axis=1)   # compute the standard deviation along the specified axis (train data)
test_mean = np.mean(test_scores, axis=1)   # compute the arithmetic mean along the specified axis (test data)
test_std = np.std(test_scores, axis=1)     # compute the standard deviation along the specified axis (test data)
```

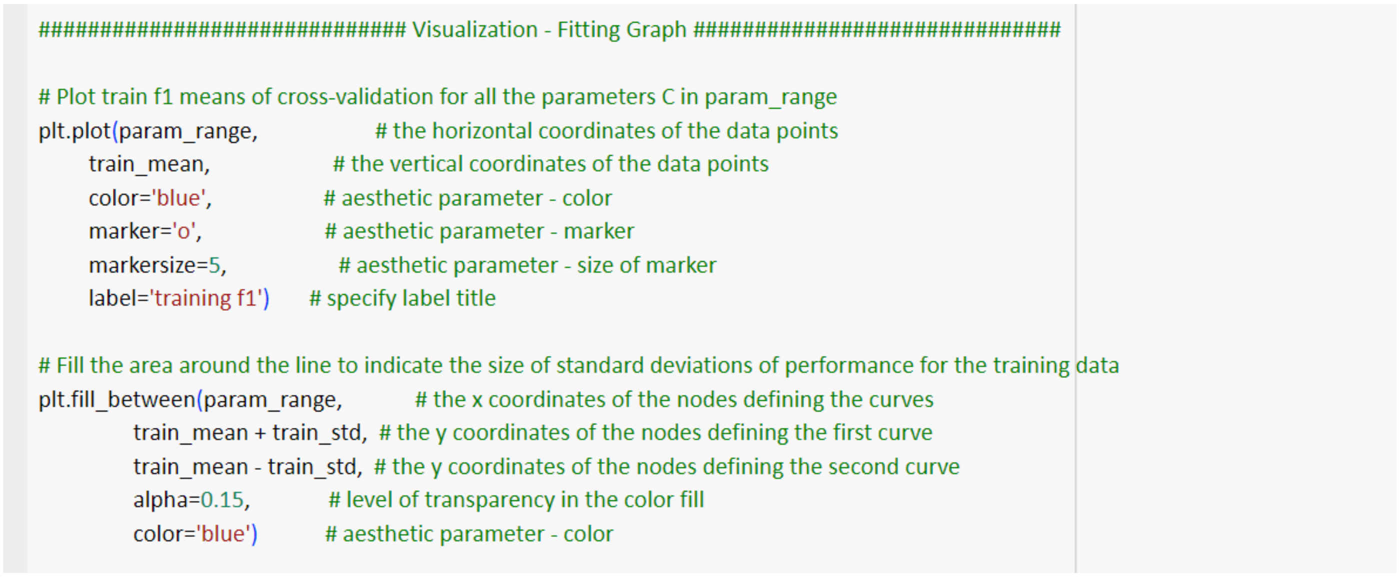
Finally we plot the fitting graph.
We first plot the fitting graph for in-sample sample performance using training data.

1. plt.plot is used to create a line plot

- param_range is the list of values for the max_depth hyperparameter on the x-axis.
- train_mean contains the mean F1 scores for the training data for each value of max_depth

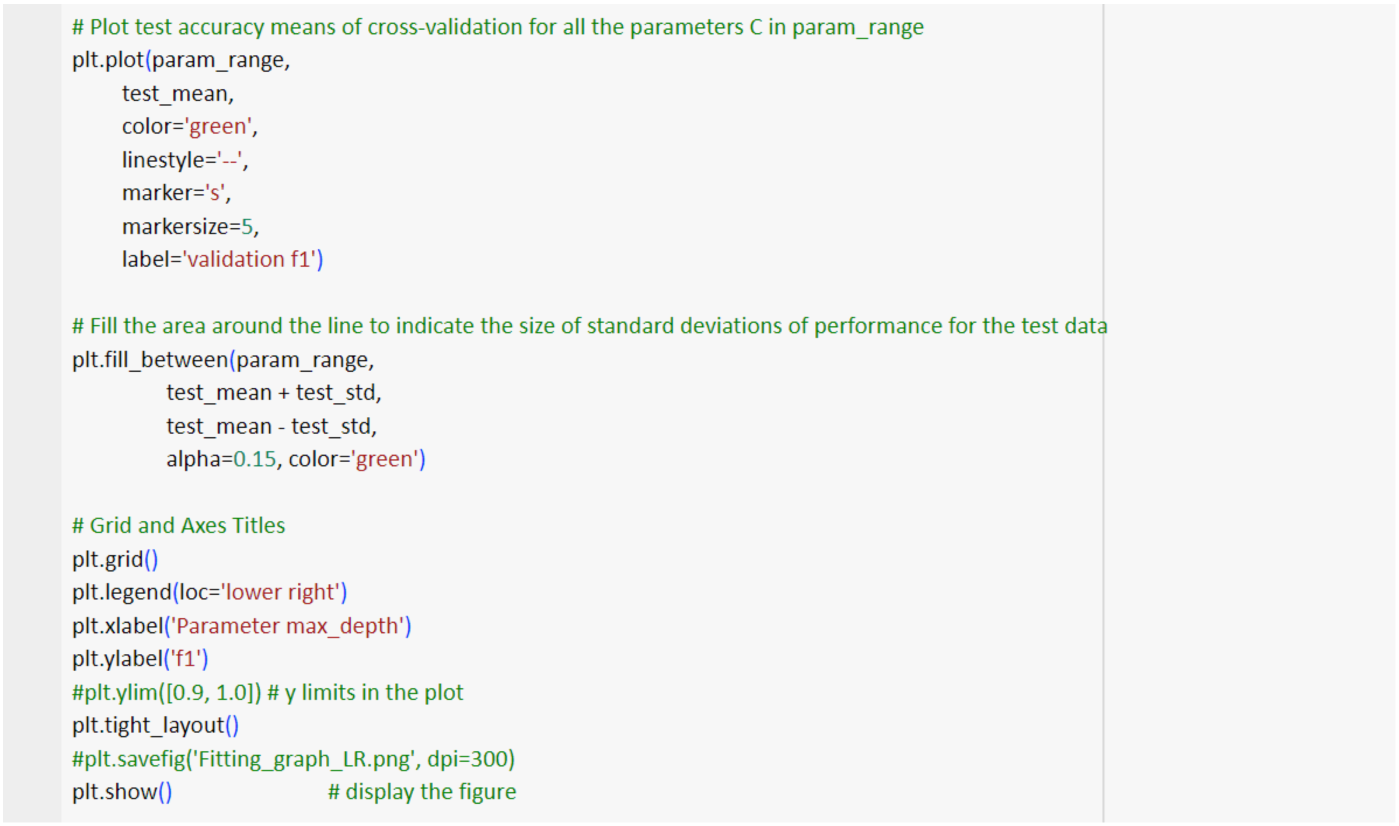
2. plt.fill_between is used to fill the area around the training F1 scores line to indicate the standard deviations.

- param_range is the x-coordinates of the nodes defining the curves.
- train_mean + train_std and train_mean - train_std define the upper and lower bounds of the filled area.
- alpha=0.15 sets the transparency level of the filled area.

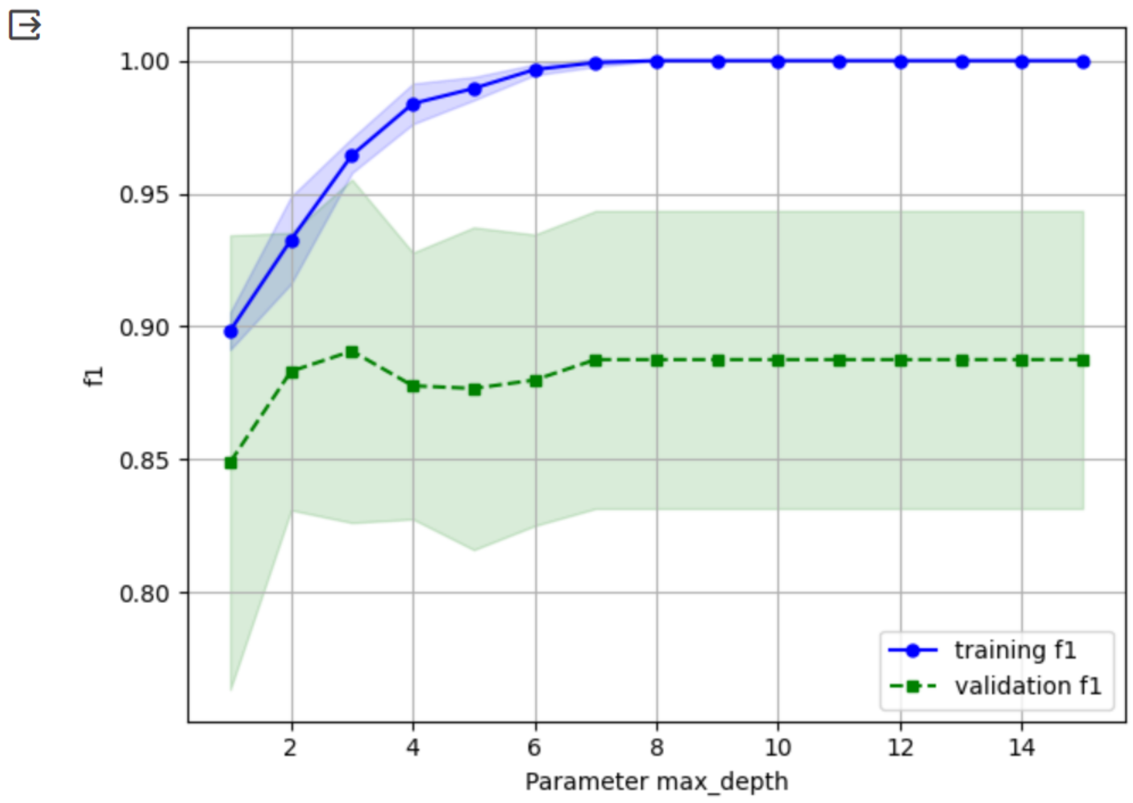


```
############################### Visualization - Fitting Graph ###############################

# Plot train f1 means of cross-validation for all the parameters C in param_range
plt.plot(param_range,             # the horizontal coordinates of the data points
     train_mean,              # the vertical coordinates of the data points
     color='blue',            # aesthetic parameter - color
     marker='o',              # aesthetic parameter - marker
     markersize=5,            # aesthetic parameter - size of marker
     label='training f1')     # specify label title

# Fill the area around the line to indicate the size of standard deviations of performance for the training data
plt.fill_between(param_range,         # the x coordinates of the nodes defining the curves
         train_mean + train_std,  # the y coordinates of the nodes defining the first curve
         train_mean - train_std,  # the y coordinates of the nodes defining the second curve
         alpha=0.15,          # level of transparency in the color fill
         color='blue')        # aesthetic parameter - color
```

Then we plot the fitting graph for out-of-sample sample performance using test data, applying the same technique when we drew the fitting graph for training data.



```
# Plot test accuracy means of cross-validation for all the parameters C in param_range
plt.plot(param_range,
     test_mean,
     color='green',
     linestyle='--',
     marker='s',
     markersize=5,
     label='validation f1')

# Fill the area around the line to indicate the size of standard deviations of performance for the test data
plt.fill_between(param_range,
         test_mean + test_std,
         test_mean - test_std,
         alpha=0.15, color='green')

# Grid and Axes Titles
plt.grid()
plt.legend(loc='lower right')
plt.xlabel('Parameter max_depth')
plt.ylabel('f1')
#plt.ylim([0.9, 1.0]) # y limits in the plot
plt.tight_layout()
#plt.savefig('Fitting_graph_LR.png', dpi=300)
plt.show()                  # display the figure
```



When the max_depth is less than 3, the model hasn't done a good job capturing the underlying patterns in the data, increasing the model complexity can further increase the performance.

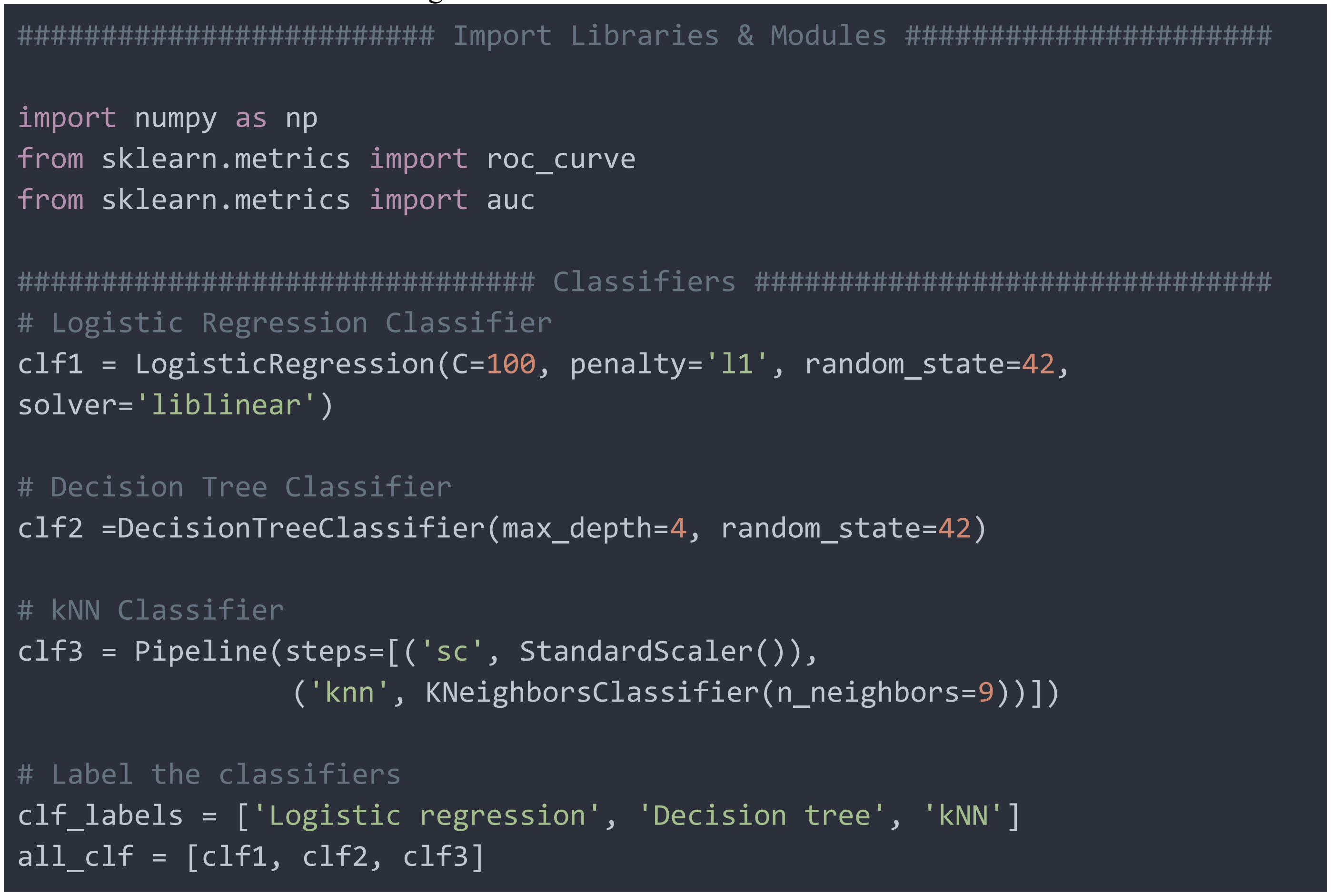
Max_depth=3 is the sweet spot for this model, because the f1 score for validation data, which we care about the most, is the highest.

After 3, the model began to overfit a little, it started to pay too much attention to noise and lead to poorer generalization performance.

Using a decision tree model on the cancer study dataset, we identified max_depth=3 to optimize the crucial diagnostic task.

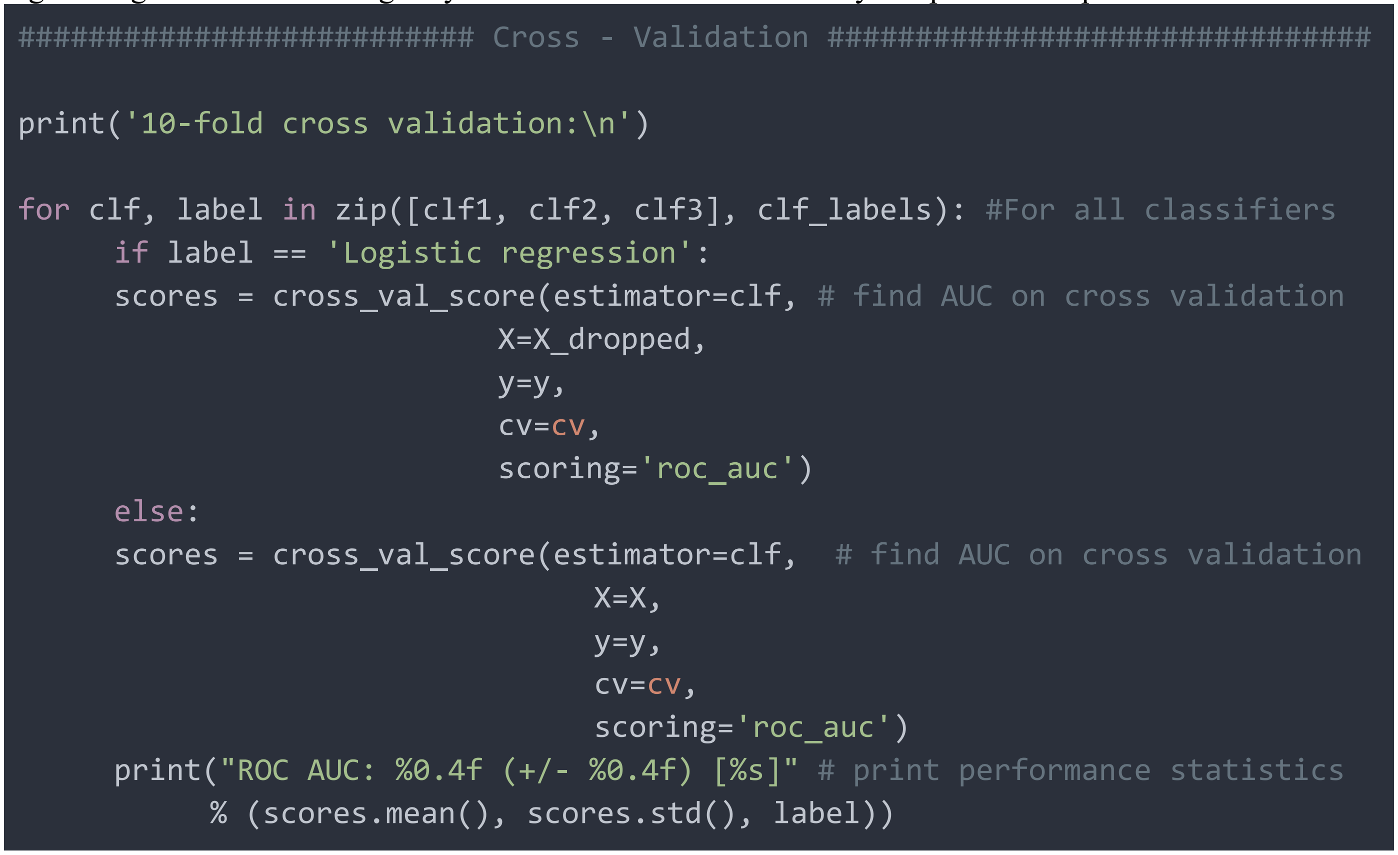
d) Create an ROC curve for k-NN, decision tree, and logistic regression. Discuss the results. Which classifier would you prefer to choose? Please provide screenshots of your code and explain the process you have followed.

To build our ROC curves, we first imported the necessary libraries, and defined and initialized our three different machine learning classifiers. We based the parameters of our classifiers here on the optimal models we found earlier in the assignment.

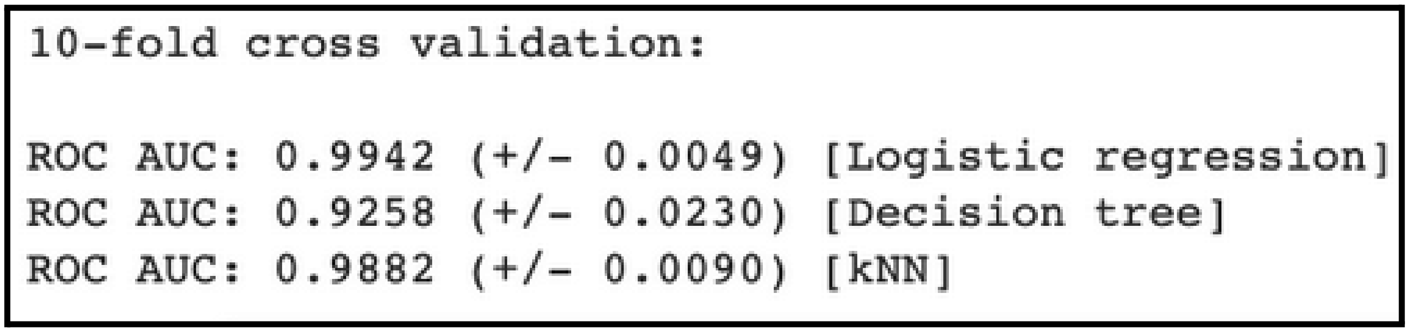


```
######################### Import Libraries & Modules ######################

import numpy as np
from sklearn.metrics import roc_curve
from sklearn.metrics import auc

############################### Classifiers ###############################
# Logistic Regression Classifier
clf1 = LogisticRegression(C=100, penalty='l1', random_state=42,
solver='liblinear')

# Decision Tree Classifier
clf2 =DecisionTreeClassifier(max_depth=4, random_state=42)

# kNN Classifier
clf3 = Pipeline(steps=[('sc', StandardScaler()),
                ('knn', KNeighborsClassifier(n_neighbors=9))])

# Label the classifiers
clf_labels = ['Logistic regression', 'Decision tree', 'kNN']
all_clf = [clf1, clf2, clf3]
```

Then, our code loops to iterate over the three classifiers and their corresponding labels (clf_labels). The zip function pairs each classifier with its label for easier reference. For each classifier, we perform cross validation to find the ROC AUC. At the end of each loop, we print our mean ROC AUC, AUC standard deviation, and the appropriate label. Please note that our code includes an if statement to ensure that our logistic regression is run using only the variables selected earlier by our parameter optimization.



```
########################## Cross - Validation ###############################

print('10-fold cross validation:\n')

for clf, label in zip([clf1, clf2, clf3], clf_labels): #For all classifiers
      if label == 'Logistic regression':
      scores = cross_val_score(estimator=clf, # find AUC on cross validation
                           X=X_dropped,
                           y=y,
                           cv=cv,
                           scoring='roc_auc')
      else:
      scores = cross_val_score(estimator=clf,  # find AUC on cross validation
                                 X=X,
                                 y=y,
                                 cv=cv,
                                 scoring='roc_auc')
      print("ROC AUC: %0.4f (+/- %0.4f) [%s]" # print performance statistics
           % (scores.mean(), scores.std(), label))
```



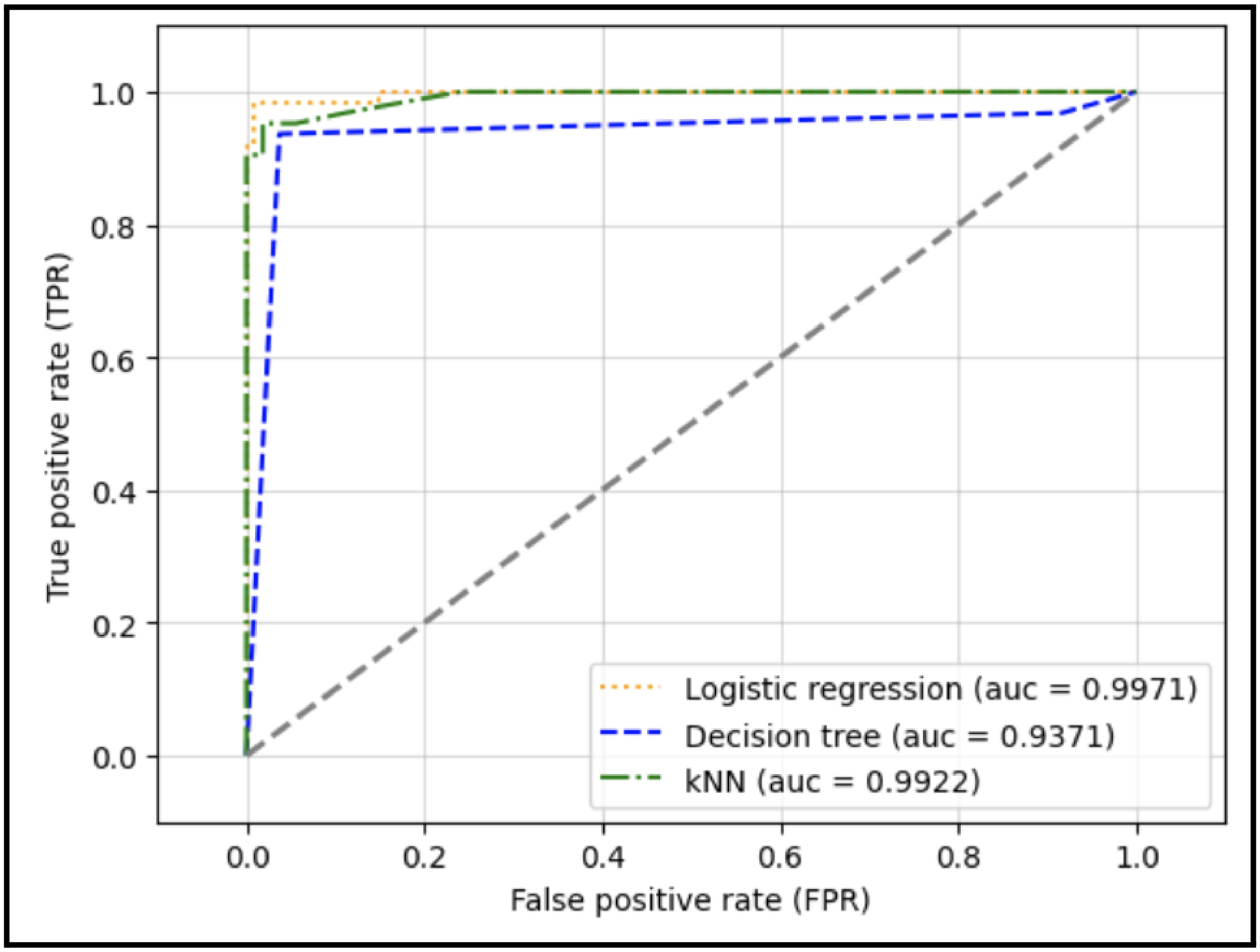
```
10-fold cross validation:

ROC AUC: 0.9942 (+/- 0.0049) [Logistic regression]
ROC AUC: 0.9258 (+/- 0.0230) [Decision tree]
ROC AUC: 0.9882 (+/- 0.0090) [kNN]
```

(More discussion of ROC AUC at the end of this section)

Finally, we visualize our ROC curves to observe the AUC. The code iterates through the classifiers, their corresponding labels, colors, and line styles. For each classifier, we first make predictions on the test data (again, using the refined feature set for logistic regression) and compute the ROC curve along with the Area Under the Curve (AUC) score. Then, we plot the ROC curve using the specified color and line style, and include the AUC value in the label for that curve. Finally, our code places a legend in the lower-right corner of the plot, and a dashed line is added to visualize the ROC curve of a random classifier. This code provides a visual comparison of the ROC curves and AUC values for each of our three different classifiers so that we can assess their discrimination abilities in binary classification.

```
################################ Visualization ###############################

colors = [ 'orange', 'blue', 'green']       # colors for visualization
linestyles = [':', '--', '-.', '-']         # line styles for visualization
for clf, label, clr, ls in zip(all_clf, clf_labels, colors, linestyles):
     if label == 'Logistic regression':
     # Assuming the label of the positive class is 1 and data is normalized
     y_pred = clf.fit(X_dropped_train,
y_dropped_train).predict_proba(X_dropped_test)[:, 1] # make predictions

     fpr, tpr, thresholds = roc_curve(y_true=y_dropped_test, # build ROC
                                      y_score=y_pred)
     roc_auc = auc(x=fpr, y=tpr)                            # compute AUC
     plt.plot(fpr, tpr,                                     # plot ROC Curve
          color=clr,
          linestyle=ls,
          label='%s (auc = %0.4f)' % (label, roc_auc))
     else:
    # make predictions based on the classifiers
     y_pred = clf.fit(X_train, y_train).predict_proba(X_test)[:, 1]
     fpr, tpr, thresholds = roc_curve(y_true=y_test,        # build ROC curve
                                      y_score=y_pred)
     roc_auc = auc(x=fpr, y=tpr)                            # compute AUC
     plt.plot(fpr, tpr,                                     # plot ROC Curve
          color=clr,
          linestyle=ls,
          label='%s (auc = %0.4f)' % (label, roc_auc))

plt.legend(loc='lower right')    # where to place the legend
plt.plot([0, 1], [0, 1],         # visualize random classifier
     linestyle='--',       # aesthetic parameters
```

We will note that due to the difference in methods for calculating ROC AUC in our printed vs. visualized methods, we do get slightly different AUC values. However, we can make similar conclusions, taking both into consideration.



Based on the ROC AUC scores that we calculated from our optimal kNN, logistic regression, and decision tree classifiers, it is clear that the logistic regression model outperforms our other models in terms of predictive accuracy for the dataset. The logistic regression model achieves an ROC AUC of 0.9942, with a fairly small confidence interval of +/- 0.0049. The numbers suggest that our logistic regression provides a robust, accurate, and consistently high performing classification model for the dataset, with a strong ability to discriminate between cancerous (malignant) and non-cancerous (benign) growths.

While the decision tree and kNN models also returned good ROC AUC scores, they fall short of the performance of our logistic regression model, which makes logistic regression the preferred choice. We will note that in a real world scenario, choosing the right classifier depends on far more than purely AUC; interpretability, computational resources, and specific goals when using this dataset will impact what makes for the "best" classifier. In particular, for diagnosing breast cancer, minimizing false negatives is a must, as a false negative counts for a patient who had breast cancer, but was not diagnosed and treated as early as possible. With that in mind, a precision-recall curve may act as a better visual and metric to determine the right model for this problem. But, in response to the question and based solely on the AUC as our performance metric for this question, logistic regression appears to be the best-suited classifier for the dataset.