assignment3

January 31, 2025

0.1 Machine Learning

0.2 Assignment 3: Splitting, Cross-Validation and the Fundamental Tradeoff

0.2.1 Assignment Learning Goals:

By the end of the module, students are expected to:

- Use `train_test_split` for data splitting and explain the importance of shuffling during data splitting.
- Explain the difference between train, validation, test, and "deployment" data.
- Identify the difference between training error, validation error, and test error.
- Do cross-validation with use cross_val_score and cross_validate to calculate cross-validation error.
- Recognize overfitting, underfitting, and the fundamental tradeoff.
- Follow the golden rule and identify the scenarios when it's violated.

Any place you see `...`, you must fill in the function, variable, or data to complete the code. Substitute the `None` with your completed code and answers then proceed to run the cell!

Note that some of the questions in this assignment will have hidden tests. This means that no feedback will be given as to the correctness of your solution. It will be left up to you to decide if your answer is sufficiently correct. These questions are worth 2 points.

[1]:

```python
# Import libraries needed for this lab
from hashlib import sha1

import altair as alt
import graphviz
import numpy as np
import pandas as pd
import sklearn

from IPython.display import HTML
from sklearn import tree
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import test_assignment3 as t

alt.renderers.enable('html')
```

```
[1]: RendererRegistry.enable('html')
```

0.3 1. Splitting Your Data and Exploring Your Data

For this question we are going to be working with a dataset modified from Kaggle. This data was collected from a survey-based study of the sleeping habits of individuals within the US. Note that these are the results of the pilot survey.

We will be building a model using features from this data to predict if the an individual will have breakfast or not.

For more information on the columns you can refer to this website.

```
[3]: sleep_df = pd.read_csv('data/sleep.csv')
     sleep_df.head()
```

```
[3]:    Enough  Hours  PhoneReach  PhoneTime  Tired  Breakfast
     0       1    8.0           1          1      3          1
     1       0    6.0           1          1      3          0
     2       1    6.0           1          1      2          1
     3       0    7.0           1          1      4          0
     4       0    7.0           1          1      2          1
```

```
[7]: sleep_df.shape
```

```
[7]: (102, 6)
```

Question 1.1 {points: 0}

Before we do anything with our data we need to split it into our training set and test set. Import the necessary library to split your data.

```
[9]: # your code here
     from sklearn.model_selection import train_test_split
     # raise NotImplementedError # No Answer - remove if you provide an answer
```

```
[11]: t.test_1_1()
```

```
[11]: 'Success'
```

Question 1.2 {points: 1}

Now split the `sleep_df` dataframe into `sleep_train` and `sleep_test` using a 80/20 train to test split. Make sure to set your `random_state` to 77.

```
[13]: sleep_train, sleep_test = train_test_split(sleep_df, test_size=0.2,␣
        ↪random_state=77)
```

```
[15]: t.test_1_2(sleep_train,sleep_test)
```

```
[15]: 'Success'
```

Question 1.3 {points: 1}

Using the `sleep_train` data, look at the summary statistics produced by `.describe()` and save the results in an object named `sleep_described`.

```
[17]: sleep_described = sleep_train.describe()
```

```
[19]: t.test_1_3(sleep_described)
```

```
[19]: 'Success'
```

Question 1.4 {points: 2}

What is the average number of hours the individuals in training set `sleep_train` sleep? Save your answer rounded to 2 decimal places in an object named `mean_hours`.

```
[25]: mean_hours = round(sleep_train["Enough"].mean(), 2)

      # your code here
      #raise NotImplementedError # No Answer - remove if you provide an answer

      mean_hours
```

```
[25]: 0.32
```

```
[27]: # check that the variable exists
      assert 'mean_hours' in globals(
      ), "Please make sure that your solution is named 'mean_hours'"

      # This test has been intentionally hidden. It will be up to you to decide if␣
       ↪your solution
      # is sufficiently good.
```

Question 1.5 {points: 1}

What is the proportion of people who eat breakfast (`1` in the column) in `sleep_train`? Save your answer in an object named `break_prop`.

```
[29]: break_prop = sleep_train["Breakfast"].mean()

      # your code here
      # raise NotImplementedError # No Answer - remove if you provide an answer

      break_prop
```

```
[29]: 0.5802469135802469
```

```
[31]: t.test_1_5(break_prop)
```

```
[31]: 'Success'
```

0.4 2 Data splitting with Dummy and Random Forest Classifiers

Recall that in machine learning what we care about is generalization; we want to build models that generalize well on unseen examples. One way to ensure this is by splitting the data into training data and test data, building and tuning the model only using the training data, and then doing the final assessing on the test data.

We are going to use a new classifier called a ***Random Forest***. It's not pertinent that you know how this model works but for now just know that it is a more complex version of a decision tree and they share similar hyperparameters.

Let's see how well our dummy and random forest classifiers do in comparison on the training and test sets.

Question 2.1 {points: 1}

Split up the `sleep_df` dataframe by assigning the features to an object named `X` and the target column `Breakfast` to an object named `y`.

Next, split the `X` and `y` dataset into a 80% train and 20% test set using `train_test_split` with `random_state=77`.

Save the training features and target in objects named `X_train` and `y_train` respectively. Name the test features and target in objects `X_test` and `y_test`.

[33]:

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = sleep_df.drop(columns=["Breakfast"])
y = sleep_df["Breakfast"]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=77)
```

[35]:

```
t.test_2_1(X_train,X_test,y_train,y_test)
```

[35]:

```
'Success'
```

Question 2.2 {points: 1}

Build a `DummyClassifier` using `strategy = 'most_frequent'` and name it `dummy_model`.

Train it on `X_train` and `y_train`. Score it on the train **and** test sets.

Save the scores in an objects named `dummy_train` and `dummy_test`.

[37]:

```
from sklearn.dummy import DummyClassifier

# Create the dummy model
dummy_model = DummyClassifier(strategy="most_frequent")

# Train the model
```

```
dummy_model.fit(X_train, y_train)

# Evaluate on train and test sets
dummy_train = dummy_model.score(X_train, y_train)
dummy_test = dummy_model.score(X_test, y_test)
```

[39]:
```
t.test_2_2(dummy_train, dummy_test)
```

[39]:
```
'Success'
```

Question 2.3 {points: 1}

Build a random forest classifier using (`RandomForestClassifier()`) with `random_state=77` and name it `forest_model`.

Train it on `X_train` and `y_train`. Score it on the train **and** test sets.

Save the scores in an objects named `forest_train` and `forest_test`.

[41]:
```
from sklearn.ensemble import RandomForestClassifier

# Create the RandomForest model
forest_model = RandomForestClassifier(random_state=77)

# Train the model
forest_model.fit(X_train, y_train)

# Evaluate on train and test sets
forest_train = forest_model.score(X_train, y_train)
forest_test = forest_model.score(X_test, y_test)
```

[43]:
```
t.test_2_3(forest_train,forest_test,forest_model)
```

[43]:
```
'Success'
```

Question 2.4 {points: 2}

Which model has the best training accuracy?

A) `DummyClassifier`.

B) `RandomForestClassifier`.

C) Both A and B

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer2_4`.

[49]:
```
answer2_4 = "B"
# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
answer2_4
```

```
[49]: 'B'
```

```
[53]: # check that the variable exists
      assert 'answer2_4' in globals(
      ), "Please make sure that your solution is named 'answer2_4'"

      # This test has been intentionally hidden. It will be up to you to decide if␣
       ↪your solution
      # is sufficiently good.
```

Question 2.5 {points: 1}

Which model has the best test accuracy?

A) `DummyClassifier`.

B) `RandomForestClassifier`.

C) Both A and B

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer2_5`.

```
[61]: answer2_5 = "B"

      # your code here
      # raise NotImplementedError # No Answer - remove if you provide an answer
      answer2_5
```

```
[61]: 'B'
```

```
[63]: t.test_2_5(answer2_5)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[63], line 1
----> 1 t.test_2_5(answer2_5)

File ~\Desktop\COIS 3550H\Assignments\Assignment3\A3\test_assignment3.py:65, in
 ↪test_2_5(answer)
     63 def test_2_5(answer):
     64     assert not answer is None, "Your answer does not exist. Have you␣
 ↪passed in the correct variable?"
---> 65     assert sha1(str(answer.lower() + 'p').encode('utf8')).hexdigest() =␣
 ↪"ac78b022715c5b8357b4dca8045e8463b4de2124", "Your solution is inocrrect. Are␣
 ↪you examinng the test scores properly?"
     66     return("Success")

AssertionError: Your solution is inocrrect. Are you examinng the test scores␣
 ↪properly?
```

Question 2.6 {points: 1}

Which model is overfitting?

A) `DummyClassifier`

B) `RandomForestClassifier`

C) Both A and B

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer2_6`.

[67]:
```
answer2_6 = "B"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
answer2_6
```

[67]:
```
'B'
```

[69]:
```
t.test_2_6(answer2_6)
```

[69]:
```
'Success'
```

Question 2.7 {points: 1}

Do you expect the `DummyClassifier` to be sensitive to data splitting (Not just on this dataset)?

A) Yes since it's predicting the most occurring value and there is a chance that all of one category type is in the test set which could change the most frequently occurring category in the training set.

B) Yes, it's predicting a new value each time so it should be changing with splitting.

C) No, The most occurring value will alway be the same.

D) No, it's going to be static in the way it predicts.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer2_7`.

[73]:
```
answer2_7 = "A"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
answer2_7
```

[73]:
```
'A'
```

[75]:
```
t.test_2_7(answer2_7)
```

[75]:
```
'Success'
```

1 3. Cross-Validation

Instead of using a single train test split like we did in exercise 2, in this question 5-fold cross-validation using `cross_validate()`.

Question 3.1 {points: 0}

Import `cross_validate` from the `sklearn` library.

[77]:
```
from sklearn.model_selection import cross_validate
```

[79]:
```
t.test_3_1()
```

[79]:
```
'Success'
```

Question 3.2 {points: 1}

Create a new ***Random Forest Classifer*** and name it, `cv_model`. Make sure to set `random_state=77`.

[81]:
```
cv_model = RandomForestClassifier(random_state=77)
```

[83]:
```
t.test_3_2(cv_model)
```

[83]:
```
'Success'
```

Question 3.3 {points: 1}

Use cross-validation using `cross_validate()` on the `X` and `y` objects using the model `cv_model` and passing `return_train_score=True`.

Save the result in an object named `cv_scores`.

[85]:
```
cv_scores = cross_validate(cv_model, X, y, cv=5, return_train_score=True)
```

[87]:
```
t.test_3_3(cv_scores)
```

[87]:
```
'Success'
```

Question 3.4 {points: 1}

Convert `cv_scores` into a dataframe as save it as an object named `cv_scores_df`.

[89]:
```
import pandas as pd

cv_scores_df = pd.DataFrame(cv_scores)
```

[91]:
```
t.test_3_4(cv_scores_df)
```

[91]:
```
'Success'
```

Question 3.5 {points: 1}

What are the mean values of each column? Save your results as a series in a object named `mean_stats`.

[93]:
```
mean_stats = pd.Series(cv_scores_df.mean())
```

[95]:
```
t.test_3_5(mean_stats)
```

[95]:
```
'Success'
```

Question 3.6 {points: 2}

Are we violating the golden rule here?

A) No, although test examples in one split are used as training example in another split, in each split, train and test examples are completely separate.

B) No, cross-validation is a special case where this rule does not apply.

C) Yes, train and test examples are mixed and therefore the golden rule is violated.

D) Yes, the data examples are using features that are in both train and test data and therefore the golden rule is violated.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer3_6`.

[99]:
```
answer3_6 = "A"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
answer3_6
```

[99]:
```
'A'
```

[101]:
```
# check that the variable exists
assert 'answer3_6' in globals(
), "Please make sure that your solution is named 'answer3_6'"

# This test has been intentionally hidden. It will be up to you to decide if␣
 ↪your solution
# is sufficiently good.
```

2 4. Hyperparameter Tuning

In Assignment 2, we explored the `max_depth` hyperparameter of the `DecisionTreeClassifier`. In this exercise, you'll explore another hyperparameter, `min_samples_split` with the `RandomForestClassifier` which is also a decision tree hyperparameter. See the documentation for more details on this hyperparameter.

[103]:
```
sleep_df = pd.read_csv('data/Sleep.csv')
sleep_df.head()
```

```
[103]:    Enough  Hours  PhoneReach  PhoneTime  Tired  Breakfast
       0       1    8.0           1          1      3          1
       1       0    6.0           1          1      3          0
       2       1    6.0           1          1      2          1
       3       0    7.0           1          1      4          0
       4       0    7.0           1          1      2          1
```

```
[107]: X = sleep_df.drop(columns = ['Breakfast'])
       y = sleep_df['Breakfast']
```

Question 4.1 {points: 1}

Split X and y from the sleep_df dataset into a 80% train and 20% test subset using sklearn.model_selection.train_test_split and random_state=77. Make sure you split the features from the target in objects named X_train, X_test, y_train, y_test.

```
[109]: # Splitting the dataset into training and testing sets
       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
        ↪random_state=77)
```

```
[111]: t.test_4_1(X_train,X_test,y_train,y_test)
```

```
[111]: 'Success'
```

Question 4.2 {points: 3}

Let's explore the min_samples_split hyperparameter.

In order to do this you will need to make a for loop that appends the results to the lists in the dictionary results_dict that we've provided for you below.

Here we are giving you the steps on how to complete this question.

Create a for loop that iterates over min_sample_split values from 2 to 50(inclusive) in increments of 2 (We've started this for you).

Each iteration should: 1. Create a RandomForestClassifier object with the hyperparameter min_samples_split changing at each iteration. Set a random_state to 77. 2. Run 10-fold cross-validation with this min_samples_split using cross_validate to get the mean train and validation accuracies. Make sure to set return_train_score=True to get the training score in each fold. 3. Appends the min_samples_split value to the list in the key min_samples_split of dictionary results_dict. 4. Appends the mean train_score of the cross-validation folds to the list in the mean_train_score key. 5. Appends the mean test_score of the cross-validation folds to the list in the mean_cv_score key.

(Note that this may take a few minutes to execute)

```
[121]: from sklearn.ensemble import RandomForestClassifier
       from sklearn.model_selection import cross_validate

       # Dictionary to store results
       results_dict = {
```

```
    "min_samples_split": [],
    "mean_train_score": [],
    "mean_cv_score": []
}

# Iterate over different values of min_samples_split
for sample_split in range(2, 51, 2):
    # Create the model with the current min_samples_split
    model = RandomForestClassifier(min_samples_split=sample_split,␣
 ↪random_state=77)

    # Perform cross-validation
    scores = cross_validate(model, X, y, cv=10, return_train_score=True)

    # Store the results
    results_dict["min_samples_split"].append(sample_split)
    results_dict["mean_train_score"].append(scores["train_score"].mean())
    results_dict["mean_cv_score"].append(scores["test_score"].mean())

results_dict
```

```
[121]: {'min_samples_split': [2,
         4,
         6,
         8,
         10,
         12,
         14,
         16,
         18,
         20,
         22,
         24,
         26,
         28,
         30,
         32,
         34,
         36,
         38,
         40,
         42,
         44,
         46,
         48,
         50],
        'mean_train_score': [0.8409818442427139,
         0.816997133301481,
         0.7832298136645962,
         0.7625537505972289,
         0.746225513616817 9,
         0.7331103678929766,
         0.7233397037744864,
         0.7200668896321071,
         0.7146201624462494,
         0.702639751552795,
         0.6884734830387004,
         0.6939202102245581,
         0.6851887243191591,
         0.6841017677974198,
         0.6808289536550406,
         0.6710463449593884,
         0.6634257047300525,
         0.657943143812709,
         0.6449116101290014,
         0.6394529383659818,
         0.6329073100812231,
         0.6296344959388438,
         0.6263616817964645,
         0.6274486383182035,
         0.6219780219780221],
        'mean_cv_score': [0.5409090909090909,
         0.5309090909090909,
         0.5709090909090909,
         0.5709090909090908,
         0.5509090909090909,
         0.5690909090909091,
         0.5690909090909091,
         0.5890909090909091,
         0.579090909090909,
         0.579090909090909,
         0.5681818181818181,
         0.5981818181818181,
         0.5881818181818181,
         0.5881818181818181,
         0.579090909090909,
         0.598181818181818,
         0.608181818181818,
         0.6072727272727272,
         0.6172727272727272,
         0.6072727272727272,
         0.5972727272727272,
         0.5972727272727272,
         0.5972727272727272,
         0.5872727272727272,
         0.5872727272727272]}
```

[122]:
```
t.test_4_2(results_dict)
```

[122]:
```
'Success'
```

Question 4.3 {points: 1}

Convert the dictionary `results_dict` into a dataframe named `results_df`.

[123]:
```
import pandas as pd

# Convert dictionary to DataFrame
results_df = pd.DataFrame(results_dict)

results_df
```

[123]:
```
    min_samples_split  mean_train_score  mean_cv_score
0                   2          0.840982       0.540909
1                   4          0.816997       0.530909
2                   6          0.783230       0.570909
3                   8          0.762554       0.570909
4                  10          0.746226       0.550909
5                  12          0.733110       0.569091
6                  14          0.723340       0.569091
7                  16          0.720067       0.589091
8                  18          0.714620       0.579091
9                  20          0.702640       0.579091
10                 22          0.688473       0.568182
11                 24          0.693920       0.598182
12                 26          0.685189       0.588182
13                 28          0.684102       0.588182
14                 30          0.680829       0.579091
15                 32          0.671046       0.598182
16                 34          0.663426       0.608182
17                 36          0.657943       0.607273
18                 38          0.644912       0.617273
19                 40          0.639453       0.607273
20                 42          0.632907       0.597273
21                 44          0.629634       0.597273
22                 46          0.626362       0.597273
23                 48          0.627449       0.587273
24                 50          0.621978       0.587273
```

[124]:
```
t.test_4_3(results_df)
```

[124]:
```
'Success'
```

Question 4.4 {points: 1}

Use `pd.melt()` to melt the columns `mean_train_score` and `mean_cv_score` in the `results_df`. Use `var_name='score_type'` and `value_name='accuracy'` and name the new dataframe `plotting_source`.

```
[129]: plotting_source = results_df.melt(id_vars=["min_samples_split"],
                                         var_name="score_type",
                                         value_name="accuracy")

       plotting_source
```

```
[129]:     min_samples_split        score_type  accuracy
       0                   2  mean_train_score  0.840982
       1                   4  mean_train_score  0.816997
       2                   6  mean_train_score  0.783230
       3                   8  mean_train_score  0.762554
       4                  10  mean_train_score  0.746226
       5                  12  mean_train_score  0.733110
       6                  14  mean_train_score  0.723340
       7                  16  mean_train_score  0.720067
       8                  18  mean_train_score  0.714620
       9                  20  mean_train_score  0.702640
       10                 22  mean_train_score  0.688473
       11                 24  mean_train_score  0.693920
       12                 26  mean_train_score  0.685189
       13                 28  mean_train_score  0.684102
       14                 30  mean_train_score  0.680829
       15                 32  mean_train_score  0.671046
       16                 34  mean_train_score  0.663426
       17                 36  mean_train_score  0.657943
       18                 38  mean_train_score  0.644912
       19                 40  mean_train_score  0.639453
       20                 42  mean_train_score  0.632907
       21                 44  mean_train_score  0.629634
       22                 46  mean_train_score  0.626362
       23                 48  mean_train_score  0.627449
       24                 50  mean_train_score  0.621978
       25                  2     mean_cv_score  0.540909
       26                  4     mean_cv_score  0.530909
       27                  6     mean_cv_score  0.570909
       28                  8     mean_cv_score  0.570909
       29                 10     mean_cv_score  0.550909
       30                 12     mean_cv_score  0.569091
       31                 14     mean_cv_score  0.569091
       32                 16     mean_cv_score  0.589091
       33                 18     mean_cv_score  0.579091
       34                 20     mean_cv_score  0.579091
```

```
35                  22     mean_cv_score  0.568182
36                  24     mean_cv_score  0.598182
37                  26     mean_cv_score  0.588182
38                  28     mean_cv_score  0.588182
39                  30     mean_cv_score  0.579091
40                  32     mean_cv_score  0.598182
41                  34     mean_cv_score  0.608182
42                  36     mean_cv_score  0.607273
43                  38     mean_cv_score  0.617273
44                  40     mean_cv_score  0.607273
45                  42     mean_cv_score  0.597273
46                  44     mean_cv_score  0.597273
47                  46     mean_cv_score  0.597273
48                  48     mean_cv_score  0.587273
49                  50     mean_cv_score  0.587273
```

[131]:

```
t.test_4_4(plotting_source)
```

[131]:

```
'Success'
```

Question 4.5 {points: 1}

Using Altair, make a `mark_line()` plot which displays the `min_samples_split` of the random forest model on the *x*-axis and the accuracy on the train and validation sets on the *y*-axis and don't forget to add `alt.Color(score_type)` to the `encode()` function after you specify `alt.X()` and `alt.y()`.

Make sure it has the dimensions `width=500, height=300`. Don't forget to give it a title and the plot `mss_acc_plot`

[133]:

```
import altair as alt

mss_acc_plot = alt.Chart(plotting_source).mark_line().encode(
    x=alt.X("min_samples_split:Q", title="Min Samples Split"),
    y=alt.Y("accuracy:Q", title="Accuracy"),
    color=alt.Color("score_type:N", title="Score Type")
).properties(
    title="Effect of min_samples_split on Accuracy",
    width=500,
    height=300
)

mss_acc_plot
```

```
C:\Users\diksh\anaconda3\Lib\site-packages\altair\utils\core.py:395:
FutureWarning: the convert_dtype parameter is deprecated and will be removed in
a future version.  Do ``ser.astype(object).apply()`` instead if you want
``convert_dtype=False``.
  col = df[col_name].apply(to_list_if_array, convert_dtype=False)
```

```
[133]: alt.Chart(…)
```

```
[135]: t.test_4_5(mss_acc_plot)
```

```
[135]: 'Success'
```

Question 4.6 {points: 1}

From your results, what `min_samples_split` would you pick in your final model? Save your answer in an object named `best_split`.

Hint: .idxmax() may come in handy.

```
[141]: best_split = results_df["min_samples_split"][results_df["mean_cv_score"].
         ↪idxmax()]
        best_split
```

```
[141]: 38
```

```
[143]: t.test_4_6(best_split)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[143], line 1
----> 1 t.test_4_6(best_split)

File ~\Desktop\COIS 3550H\Assignments\Assignment3\A3\test_assignment3.py:152, i␣
 ↪test_4_6(answer)
    151 def test_4_6(answer):
--> 152     assert sha1(str(float(answer)).encode('utf8')).hexdigest() ==␣
 ↪"2493779251de822754e7d9cbd06e551dfa7fcd2b", "Your answer is incorrect. Are yo␣
 ↪finding the max value amongst all splits?"
    153     return("Success")

AssertionError: Your answer is incorrect. Are you finding the max value amongst
 ↪all splits?
```

Question 4.7 {points: 1}

Build a new random forest classifier name `best_model` with the best `min_samples_split` and fit it with `X_train` and `y_train`.

```
[145]: from sklearn.ensemble import RandomForestClassifier

        best_model = RandomForestClassifier(min_samples_split=best_split,␣
         ↪random_state=77)
        best_model.fit(X_train, y_train)
```

```
[145]: RandomForestClassifier(min_samples_split=38, random_state=77)
```

```
[147]: t.test_4_7(best_model)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[147], line 1
----> 1 t.test_4_7(best_model)

File ~\Desktop\COIS 3550H\Assignments\Assignment3\A3\test_assignment3.py:157, i␣
 ↪test_4_7(answer)
    155 def test_4_7(answer):
    156     answer = answer.get_params()['min_samples_split']
--> 157     assert sha1(str(answer).encode('utf8')).hexdigest() ==␣
 ↪"0a57cb53ba59c46fc4b692527a38a87c78d84028", "Make sure you are using the valu␣
 ↪for the best split in your model."
    158     return("Success")

AssertionError: Make sure you are using the value for the best split in your␣
 ↪model.
```

Question 4.8 {points: 1}

Now carry out final assessment by calling `.score()` on `X_test` and `y_test`. Save you score in an object named `test_score`.

```
[149]: test_score = best_model.score(X_test, y_test)

       test_score
```

```
[149]: 0.5714285714285714
```

```
[151]: t.test_4_8(test_score)
```

```
[151]: 'Success'
```

Question 4.9 {points: 2}

Would you say that your test score is comparable to the cross-validation results?

A) No, they are differ by over 20%.

B) No, they differ by over 10%.

C) Yes, the cross-validation scores were fairly representative.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer4_8`.

```
[153]: answer4_9 = "C"

       # your code here
       # raise NotImplementedError # No Answer - remove if you provide an answer
```

```
answer4_9
```

```
[153]: 'C'
```

```
[155]: # check that the variable exists
assert 'answer4_9' in globals(
), "Please make sure that your solution is named 'answer4_9'"

# This test has been intentionally hidden. It will be up to you to decide if␣
 ↪your solution
# is sufficiently good.
```

Question 4.10 {points: 1}

Why can't you simply pick the value of `min_samples_split` that does best on the training data?

A) Because the model will likely overfit.

B) Because the model will not generalize well on the validation data.

C) Because the `min_samples_split` that does well on the train data will not necessarily do well on the test data.

D) All of the above

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`*, assign the correct answer to an object called* `answer4_9`*.*

```
[159]: answer4_10 = "D"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
answer4_10
```

```
[159]: 'D'
```

```
[161]: t.test_4_10(answer4_10)
```

```
[161]: 'Success'
```

2.1 Before Submitting

Before submitting your assignment please do the following:

- Read through your solutions
- Makes sure that none of your code is broken
- Verify that the tests from the questions you answered have obtained the output "Success"

2.2 Attributions

- Sleep Survey Dataset: - Kaggle