ECE 47300 Assignment 3 Exercises

Name:

Exercise 0 (Important submission information)

1. Follow the instructions in the provided "uploader.ipynb" to convert your ipynb file into PDF format.
2. Please make sure to select the corresponding pages for each exercise when you submitting your PDF to Gradescope. Make sure to include both the **output** and the **code** when selecting pages. (You do not need to include the instruction for the exercises)

We may assess a 20% penalty for those who do not correctly follow these steps.

Exercise 1 (20/100 points)

In this exercise, you will implement linear regression using the polynomial features and compare results for different choices of degrees for the polynomial visually.

Task 1: Generate the data

The data should be a noisy version of a sin wave i.e $y = \sin(x) + \epsilon$ where $\epsilon \sim \mathrm{NormalDistribution}(\mu, \sigma)$. Note ϵ should be different for every point. Whenever generating a random sample from here on out, please use the `rng` instance in the code for fixing the seed. e.g., `rng.randn(arg)` .

1. Generate **50** evenly spaced numbers over the interval $[0, 4\pi]$ and store them as a vector called `X` .
2. Generate `y` from `X` by using the equation above with the parameter $\mu = 0, \sigma = 0.1$
3. Do a scatter plot on `X` and `y` and give the plot and axis reasonable names/title.

(You may want to look over the code in section 2 in the instruction notebook under "Simple Linear Regression".)

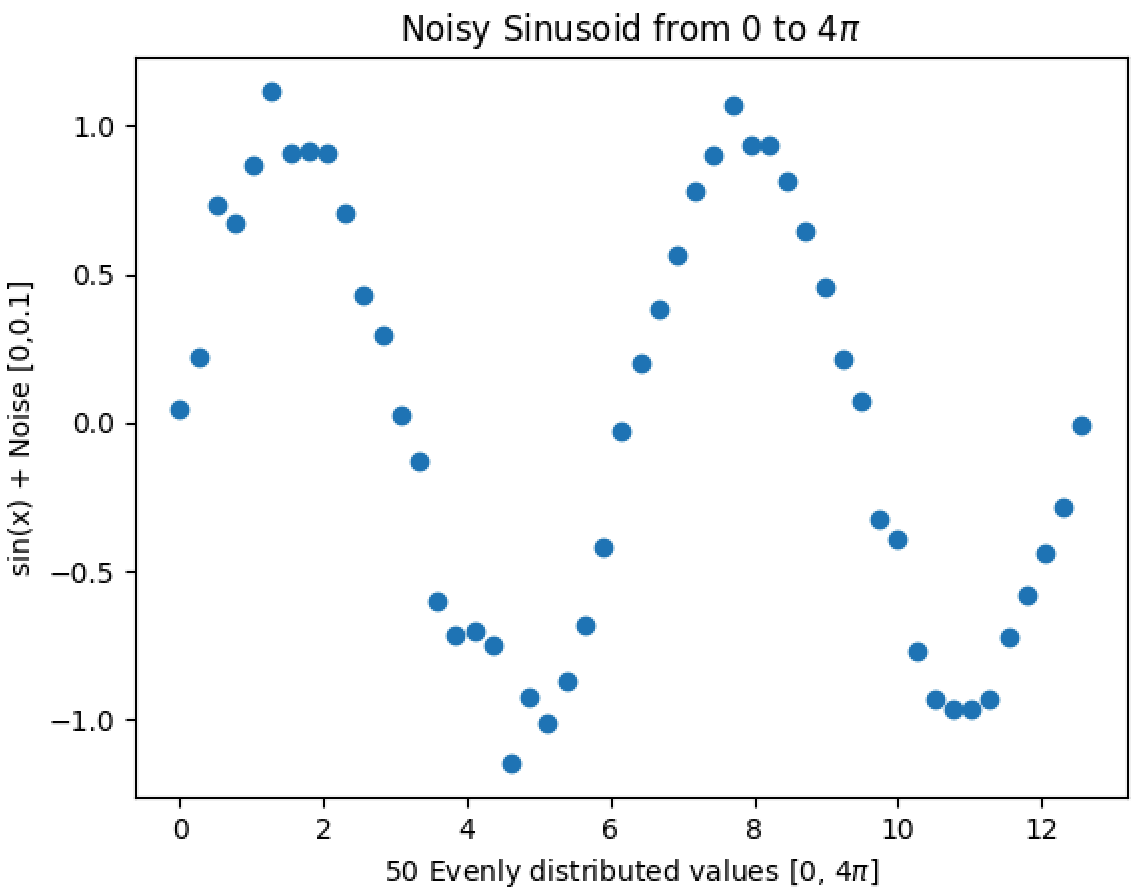
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

rng = np.random.RandomState(5) # use this if you need to generate a random sample
```

```
####################         YOUR CODE         ####################
num_samples = 50
X = np.linspace(0, (4 * np.pi), num_samples)
sigma = 0.1
y = np.sin(X) + sigma*rng.randn(num_samples)

fig, ax = plt.subplots()
ax.scatter(X, y)
ax.set_xlabel("50 Evenly distributed values [0, 4$\pi$]")
ax.set_ylabel("sin(x) + Noise [0,0.1]")
ax.set_title("Noisy Sinusoid from 0 to 4$\pi$")
####################         END CODE          ####################

plt.show()
```



Task 2: Fit the data

Now try to use tools in sklearn to fit the data with varying degrees of the polynomial. The general process is:

1. Create an estimator based on pipelining the function **PolynomialFeatures(degree)** and **LinearRegression()** (Read the instruction)
2. Fit the estimator to the data you created in Task 1. (Note the estimator will expect a 2D array so you may have to reshape `X`.)

3. Evaluate your trained estimator by using the given vector **xfit**, and plot the result curve over the scatter plot of your data. (The plot should look similar to the plot on the part 1 of the instruction)

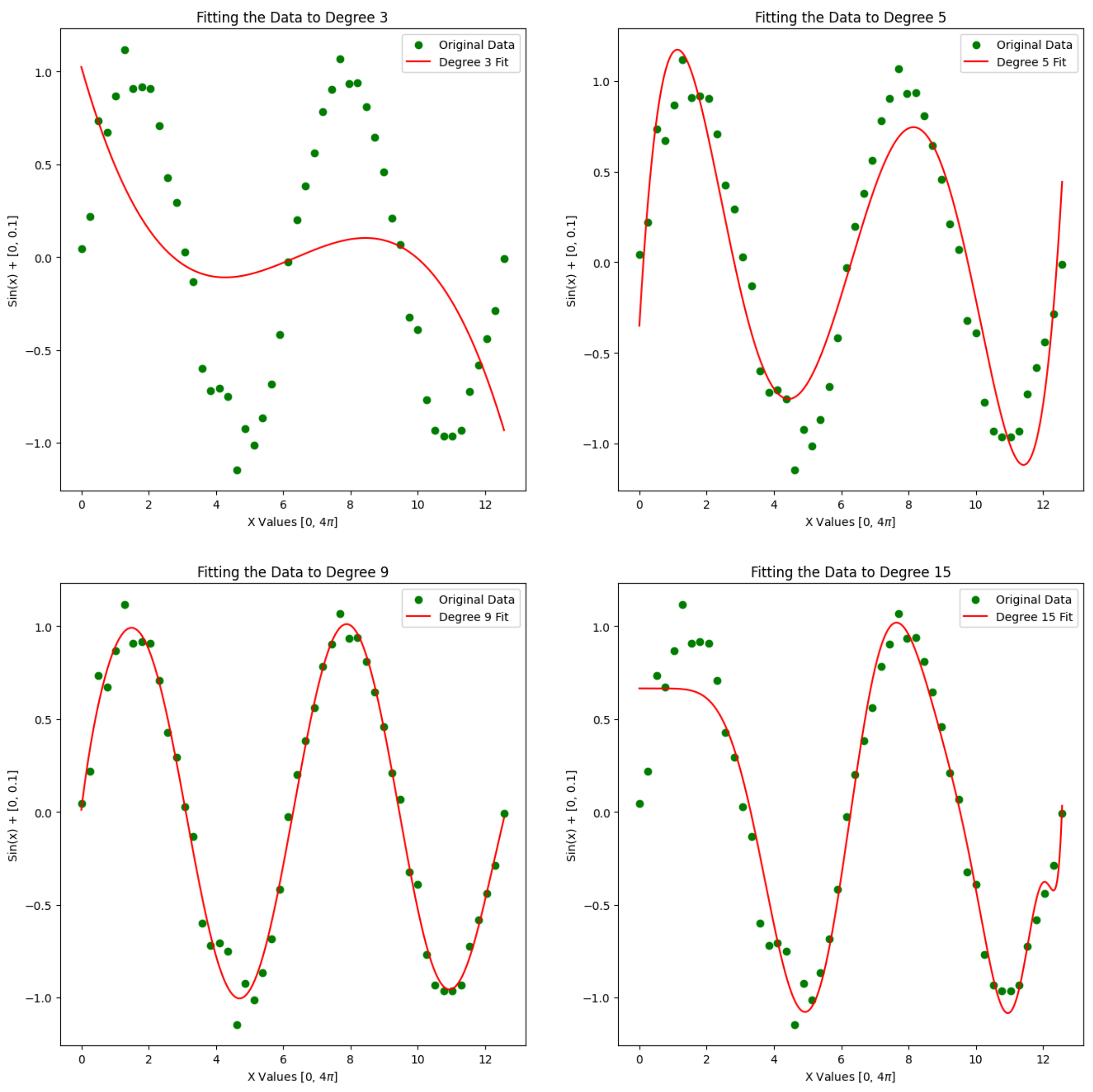
The output of your code should be a 2 by 2 grid of subplots (see `plt.subplots` or `plt.subplot`) where each plot visualizes the mdoel fitted using different polynomial degrees (see above), specifically degrees `[3, 5, 9, 15]` respectively. Each subplot should be given a reasonable title to identify what it represents.

NOTE: It is perfectly normal if the graph looks crazy for high degrees of polynomial choice.

In [2]:
```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

xfit = np.linspace(0, 4*np.pi, 1000)
rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################         YOUR CODE         ####################
degrees = [3, 5, 9, 15]
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(16,16))
# axes = axes.flatten()
axes = axes.ravel() # more memory efficient than flatten
for i, degree in enumerate(degrees):
    # set up the data via pipeline and model
    estimator = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    estimator.fit(X.reshape(-1, 1), y)
    yfit = estimator.predict(xfit[:, np.newaxis])
    # plot the respective data
    ax = axes[i]
    ax.scatter(X, y, color='green', label='Original Data') # plot original data
    ax.plot(xfit, yfit, color='red', label=f'Degree {degree} Fit') # plot fitted cu
    ax.set_xlabel("X Values [0, 4$\pi$]")
    ax.set_ylabel("Sin(x) + [0, 0.1]")
    ax.set_title(f"Fitting the Data to Degree {degree}")
    ax.legend()
####################         END CODE          ####################
plt.show()
```



Exercise 2: Visualizing KNN classifier on IRIS dataset (30/100 points)

In this exercise, you will use KNN classifier to do simple classification on Iris dataset.

Task 1: Load the Iris dataset

Iris dataset can be simply loaded by calling the function `datasets.load_iris()`. Use the official documentation for this function to create variables that stores the following information:

1. `X` : stores the last two features ('petal length', 'petal width').
2. `y` : stores all labels.
3. `feature_names` : the meaning for the last two features.

4. `target_names` : the meaning for each label.

(You will have to remove some features from the original dataset to get only 2.)

In [3]:
```
from sklearn import datasets
rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################         YOUR CODE         ####################
data = datasets.load_iris()
X = data.data[:,-2:] # slice last two columns of features
y = data.target # target is the labels
feature_names = data.feature_names[-2:]
target_names = data.target_names
####################         END CODE         ####################

print(f'X has the shape {X.shape}')
print(f'y has the shape {y.shape}')
print(f'X has features: {feature_names}')
print(f'y has labels: {target_names}')
```

```
X has the shape (150, 2)
y has the shape (150,)
X has features: ['petal length (cm)', 'petal width (cm)']
y has labels: ['setosa' 'versicolor' 'virginica']
```

Task 2: Train and visualize KNN classfier

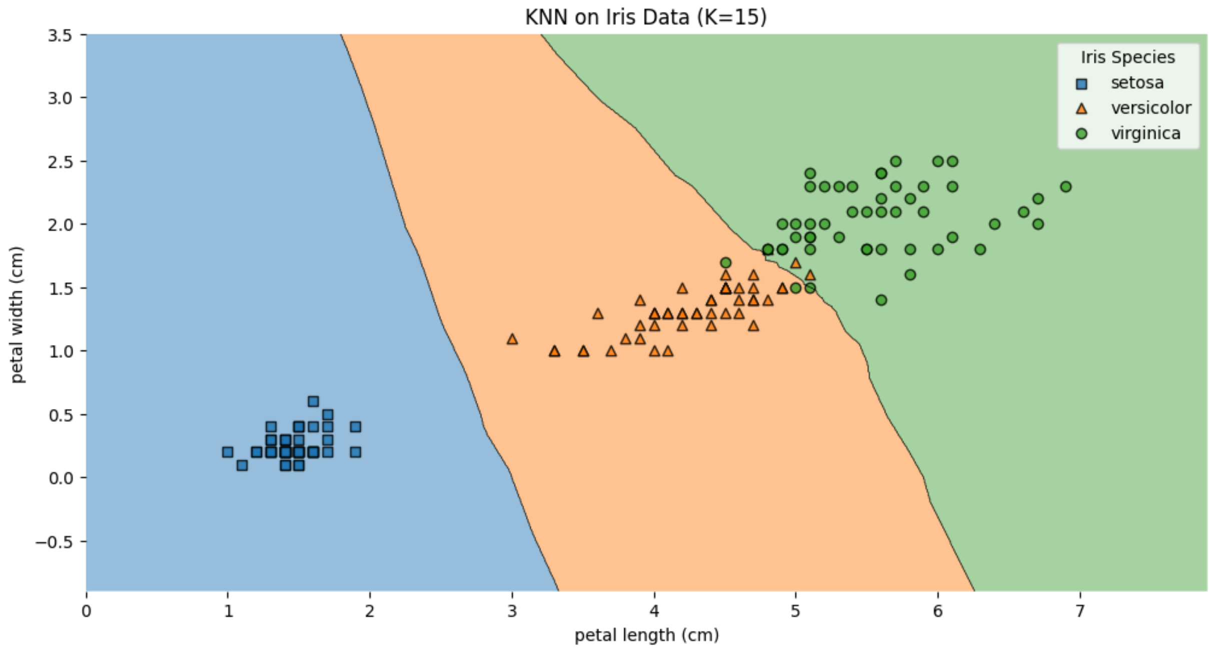
1. Setup and train the KNN classifier with K=15. (See instructions)
2. See examples here and use `plot_decision_regions()` to visualize the decision boundary for trained classifier. Be sure to name the axis with corresponding name of the feature.

In [5]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions # pip install mlxtend

rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################         YOUR CODE         ####################
K = 15
knn = KNeighborsClassifier(n_neighbors=K)
knn.fit(X, y)

plt.figure(figsize=(12,6))
ax = plot_decision_regions(X, y, clf=knn)
handles, _ = ax.get_legend_handles_labels()
labels = target_names
ax.legend(handles, labels, title='Iris Species')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.title('KNN on Iris Data (K=15)')
plt.show()
####################         END CODE         ####################
```



Exercise 3: KNN classifier on credit fraud dataset (50/100 points)

In this exercise, you will use K-nearest-neighbor method to create a model that is able to detect potential credit card fraud.

Task 1: Mount your drive

Follow the step on the instructions and mount your google drive on Colab which allows to access the .csv file uploaded on your drive that was included with this assignment.

In [66]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

Task 2: Load and preprocess datasets

In this program we are using a dataset that has the following features:

V1	V1	...	V10	Amount	Class
(float)	(float)	(float)	(float)	(float)	(str)

The first ten features are the top PCA values for certain transaction information. The reason only PCA values are given is to protect private information. The **Amount** feature is the amount of money in that particular transaction and the **Class** feature contains two classes **safe** and **Fraud**. Each class has 400 examples, your task is to predict the **Class** feature from all the other features, i.e. determine which transactions are fraudulent or not.

1. Load the given .csv file to the variable `data`.

2. Create `X` from `data` simply by dropping the last column (which will be our `y`) of the pandas dataframe, and create `y` by selecting the last column of the pandas data frame.
3. Use `train_test_split` to create the training and test set (`X_train`, `X_test`, `y_train`, `y_test`) with 20% of the data be the test data and set the `random_state` to `0`.
4. Learn the appropriate transform functions for input and output from the training dataset and then apply to both the train and test set (`X_train`, `X_test`, `y_train`, `y_test`).

In [7]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################         YOUR CODE         ####################
data = pd.read_csv('creditcard_ece570.csv')
df = pd.DataFrame(data)
last_col_name = df.columns[-1]
X = np.array(df.drop(last_col_name, axis=1))
y = np.array(df[last_col_name])
# print(f'X:{X.shape}') # (800,11)
# print(f'y:{y.shape}') # (800,)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
labeler = LabelEncoder()
scaler.fit(X_train)
labeler.fit(y_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
y_train = labeler.transform(y_train)
y_test = labeler.transform(y_test)
####################         END CODE         ####################

print(f'X_train has the shape {X_train.shape}')
print(f'y_train has the shape {y_train.shape}')
print(f'X_test has the shape {X_test.shape}')
print(f'y_test has the shape {y_test.shape}')
print(f'X_train mean is {np.mean(X_train, axis=0)}')
print(f'X_test mean is {np.mean(X_test, axis=0)}')
print(f'Sum of X_train mean is {np.sum(np.mean(X_train, axis=0))}')
print(f'Sum of X_test mean is {np.sum(np.mean(X_test, axis=0))}')
```

```
X_train has the shape (640, 11)
y_train has the shape (640,)
X_test has the shape (160, 11)
y_test has the shape (160,)
X_train mean is [-1.97931949e-16 -3.56312202e-16  2.89698820e-17  7.25808302e-16
 -6.68909372e-16  1.58900670e-16  5.14692455e-16 -1.73472348e-19
 -1.70002901e-17 -7.13318293e-16  6.27969898e-17]
X_test mean is [ 0.09347451 -0.09182168  0.15245938 -0.19522097  0.10599188  0.23410
519
  0.09620048 -0.19714382  0.18255422  0.17135069  0.12558944]
Sum of X_train mean is -4.624772786954169e-16
Sum of X_test mean is 0.677539324908696
```

Task 3: Find the optimal KNN estimator

We need to find the optimal parameters of the KNN estimator (the model selection problem) using cross validation, and then provide a final estimate of the model's generalization performance via the test set.

1. Do a grid search (using the `GridSearchCV` estimator from scikit-learn) to optimize the following hyperparameters for KNN (use estimator `KNeighborsClassifier`) and name your gridsearch object `KNN_GV`:

- The number of neighbors `n_neighbors`
- Type of weights considered `weights` (at least two options)
- Type of distance considered `metric` (at least two options) See here for possible options for those hyperparameters.

2. Fit your gridsearch by specifying the number of folds to 5. (Note: Pass only your training dataset into the `fit` function so that the model selection process doesn't see your test dataset. `GridSearchCV` will take care of doing cross validation on the training dataset.)
3. Print your best parameters combo (`best_params_`) attribute of KN and the corresponding score on the train and test set.

(Your final model should have a training accuracy of at least 95% and a test accuracy 90% to get full credit.)

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################          YOUR CODE         ####################
grid_params = {
    'n_neighbors': [1,2,3,4,5,6,7,8,9,10,11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski', 'cityblock', 'co
}
KNN_GV = GridSearchCV(KNeighborsClassifier(),
                           grid_params,
                           cv = 5 # Specifies the number of fractions for cross validati
                          )
result = KNN_GV.fit(X_train, y_train)
####################         END CODE          ####################

print(f'The best parameters are {KNN_GV.best_params_}')
print(f'The best accuracy on the training data is {KNN_GV.score(X_train, y_train)}'
print(f'The best accuracy on the testing data is {KNN_GV.score(X_test, y_test)}')
```

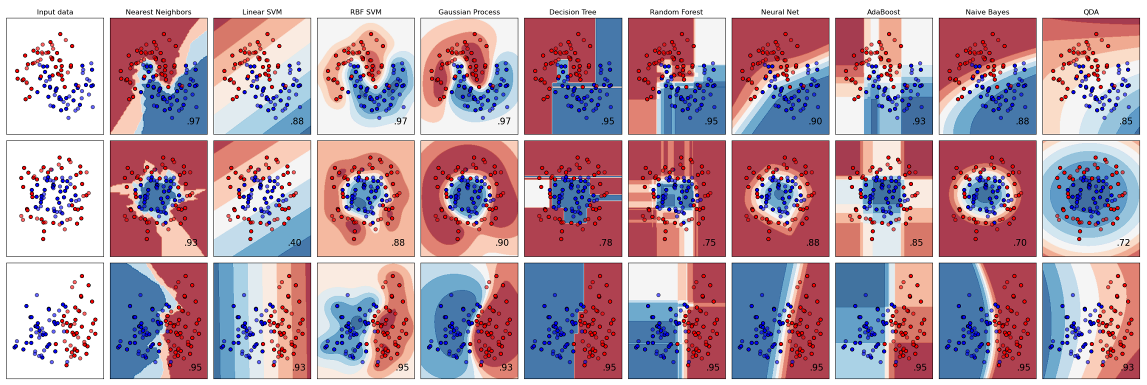
```
The best parameters are {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'unifor
m'}
The best accuracy on the training data is 0.953125
The best accuracy on the testing data is 0.90625
```

Just for fun: Try out different classifiers in scikit-learn to see if you can beat the test set performance of KNN on this dataset

(Please do not include this optional activity in your submission to simplify grading.)

Prof. Inouye was able to achieve 96% training and 94% testing accuracy using a combination of methods. Can you do better?

There are many other classifiers in scikit-learn. A really cool example of many standard classifiers can be seen in the following image from the scikit-learn example on comparing classifiers: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html:



In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

rng = np.random.RandomState(5) # use this if you need to generate a random sample

####################         YOUR CODE         ####################



####################         END CODE          ####################

print(est.score(X_train, y_train))
print(est.score(X_test, y_test))
```