

# **IT8302 APPLIED MACHINE LEARNING**

## **Practical 4 Supervised Learning**



What you will learn / do in this lab

1. *Explore Supervised Learning concepts and applications*
2. *Conduct Classification experiment using Python*
3. *Conduct Regression experiment using Python*

# TABLE OF CONTENTS

<b>1. OVERVIEW .....</b>	<b>1</b>
Introduction to supervised learning.....	1
Applications of supervised learning .....	2
 <b>2. CLASSIFICATION .....</b>	 <b>4</b>
k-Nearest Neighbor .....	4
Naive Bayes.....	5
Logistic Regression .....	6
CART (Classification Tree) .....	7
Support Vector Classification (SVC) .....	8
 <b>3. REGRESSION.....</b>	 <b>10</b>
Linear Regression .....	10
CART (Regression Tree) .....	11
Support Vector Regression (SVR) .....	12

# 1.

# OVERVIEW

In this practical we will use python with scikit-learn to apply supervised learning to some typical problems encountered. Scikit-learn comes with estimators that help to solve the classification and regression problems.

## INTRODUCTION TO SUPERVISED LEARNING

The majority of practical machine learning uses supervised learning.

Supervised learning is where you have input/attribute/feature variables (**x**) and an output/target variable (**y**) and you use an algorithm to learn the mapping function (**f**) from the input to the output.

$$\mathbf{y} = \mathbf{f}(\mathbf{X})$$

The goal is to approximate the mapping function ( $f$ ) so well that when you have new input data ( $X_{\text{new}}$ ) that you can predict the output variables ( $y$ ) for that data.

Supervised learning uses the known examples (also called labelled examples) of  $(X, y)$  to learn what is the correct mapping function ( $f$ ). The known examples all have a known correct label or value for  $y$  for every input  $X$ . Without these labelled data, we cannot apply supervised learning algorithms.



It is called **supervised learning** because the process of an algorithm learning from the **training** dataset can be thought of as a **teacher** supervising the learning process. We know the **correct answers**, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

## APPLICATIONS OF SUPERVISED LEARNING

Supervised learning problems can be further grouped into regression and classification problems.

- **Classification:** A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease".
- **Regression:** A regression problem is when the output variable is a real value, such as "dollars" or "weight".

Some real-life applications:

- *Systems Biology – Gene expression microarray data*
- *Text categorization: spam detection*
- *Face detection*
- *Signature recognition*
- *Customer discovery*
- *Medicine: Predict if a patient has heart ischemia by a spectral analysis of his/her ECG.*

Some popular examples of supervised machine learning algorithms are:

- *Linear regression for regression problems.*
- *Random forest for classification and regression problems.*
- *Support vector machines for classification problems.*

# 2.

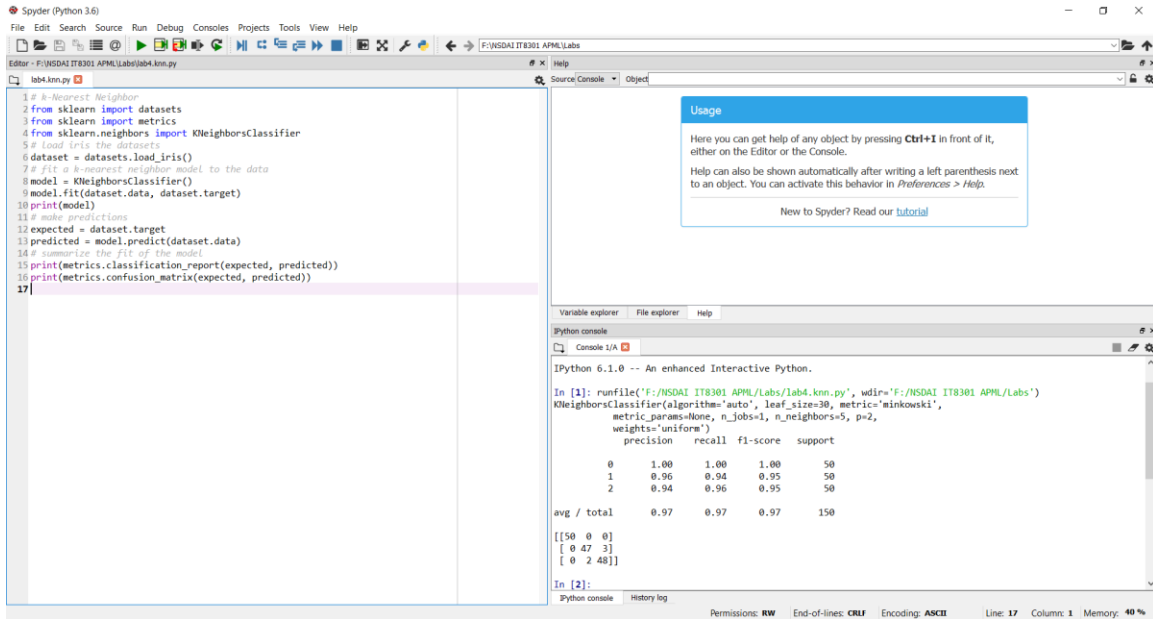
# CLASSIFICATION

In this section, we will look at some “recipes” for applying some common algorithms in scikit-learn to classification problems.

## K-NEAREST NEIGHBOR

The k-Nearest Neighbor (kNN) method makes predictions by locating similar cases to a given data instance (using a similarity function) and returning the average or majority of the most similar data instances. The kNN algorithm can be used for **classification** or **regression**.

```
# k-Nearest Neighbor
from sklearn import datasets
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
# load iris the datasets
dataset = datasets.load_iris()
# fit a k-nearest neighbor model to the data
model = KNeighborsClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

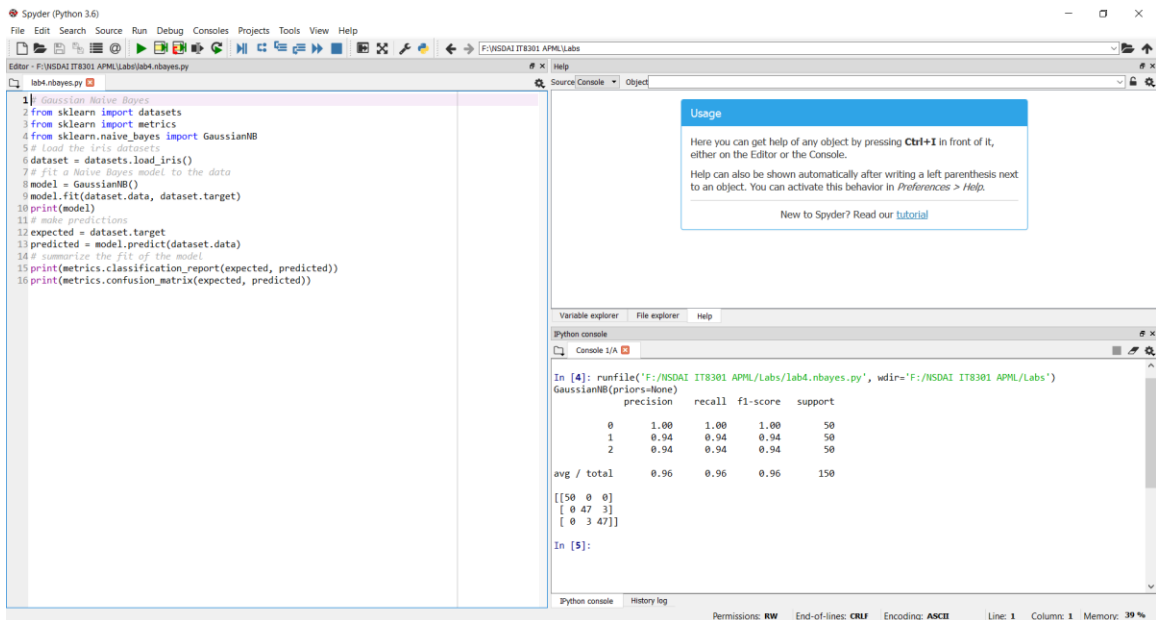


## NAIVE BAYES

Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

```
# Gaussian Naive Bayes
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
# load the iris datasets
dataset = datasets.load_iris()
# fit a Naive Bayes model to the data
model = GaussianNB()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```





## LOGISTIC REGRESSION

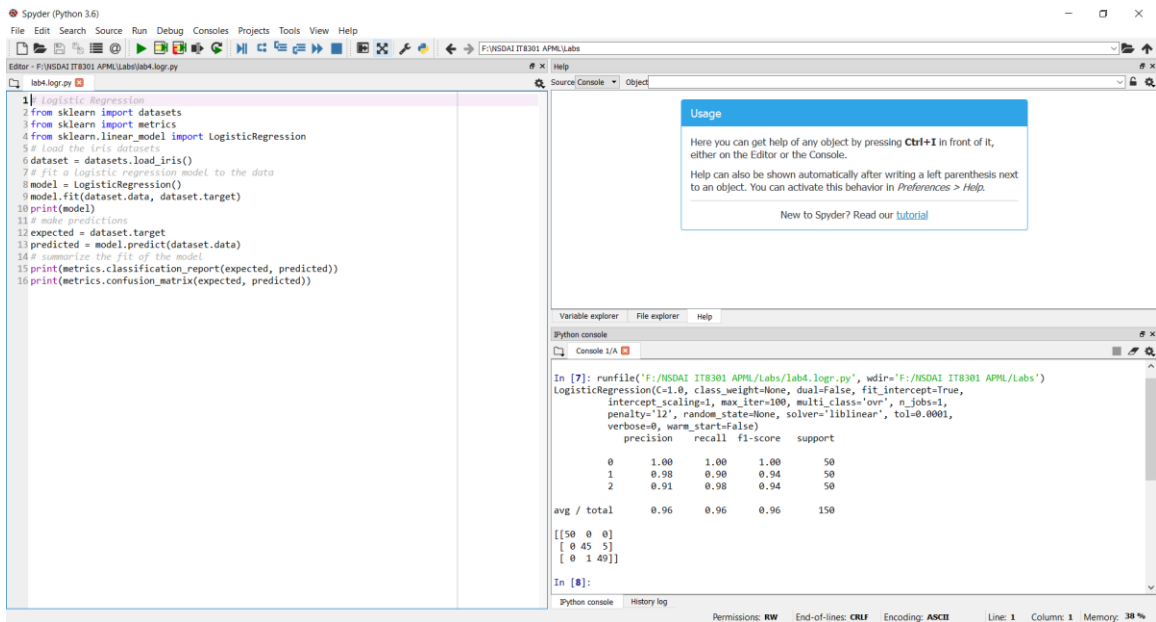
Logistic regression fits a logistic model to data and makes predictions about the probability of an event (between 0 and 1).

Because this is a multi-class classification problem and logistic regression makes predictions between 0 and 1, a one-vs-all scheme is used (one model per class). (Please note we used the multinomial mode in practical 3).

```

# Logistic Regression
from sklearn import datasets
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
# load the iris datasets
dataset = datasets.load_iris()
# fit a logistic regression model to the data
model = LogisticRegression()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))

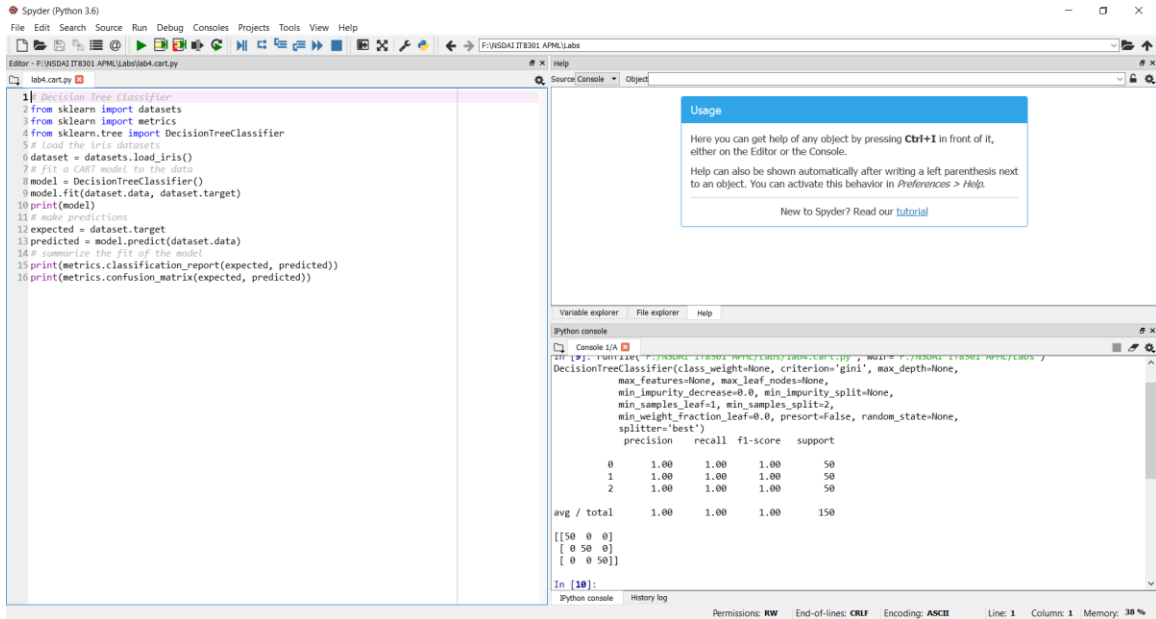
```



## CART (CLASSIFICATION TREE)

Classification and Regression Trees (CART) are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. The CART algorithm can be used for classification or regression. Decision Trees used for classification are also known as Classification Trees.

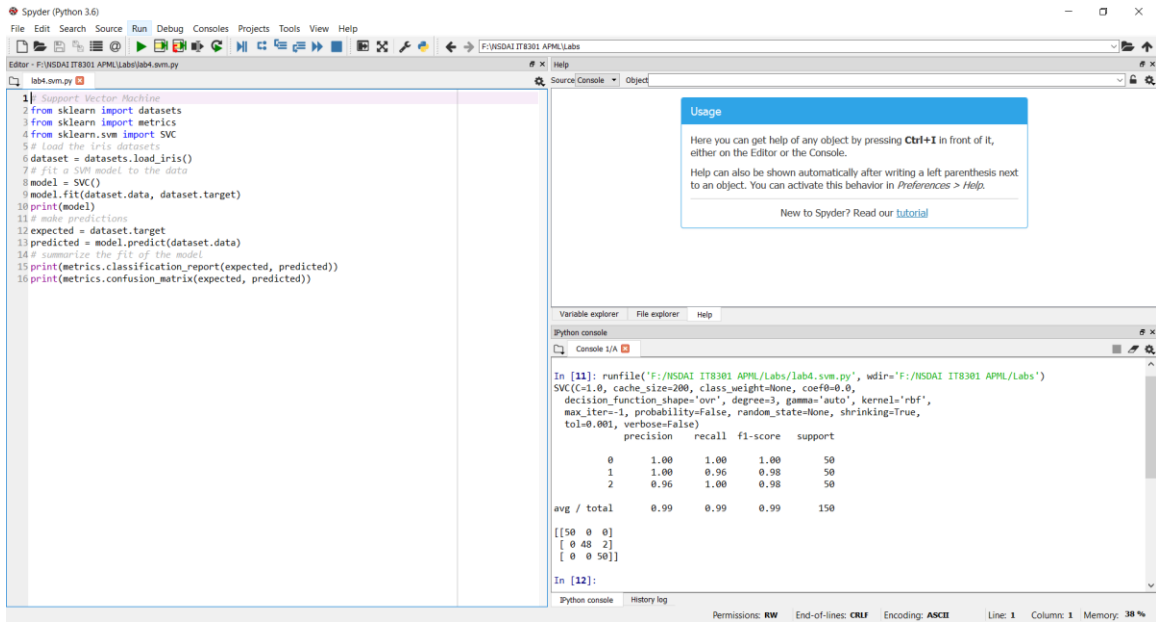
```
# Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```



## SUPPORT VECTOR CLASSIFICATION (SVC)

Support Vector Machines (SVM) are a method that uses points in a transformed problem space that best separate classes into two groups. Classification for multiple classes is supported by a one-vs-all method. Support Vector Classification (SVC) is SVM used for classification problems.

```
# Support Vector Machine
from sklearn import datasets
from sklearn import metrics
from sklearn.svm import SVC
# load the iris datasets
dataset = datasets.load_iris()
# fit a SVM model to the data
model = SVC()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```



Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

```

1# Support Vector Machine
2from sklearn import datasets
3from sklearn import metrics
4from sklearn.svm import SVC
5# Load the iris datasets
6dataset = datasets.load_iris()
7# Fit a SVM model to the data
8model = SVC()
9model.fit(dataset.data, dataset.target)
10print(model)
11# Make predictions
12expected = dataset.target
13predicted = model.predict(dataset.data)
14# Summarize the fit of the model
15print(metrics.classification_report(expected, predicted))
16print(metrics.confusion_matrix(expected, predicted))

```

Variable explorer | File explorer | Help

Python console

Console 1/A

```

In [11]: runfile('F:/NSDAI IT8301 APML/Labs/lab4.svm.py', wdir='F:/NSDAI IT8301 APML/Labs')
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernels='rbf',
    max_iter=1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	0.96	0.98	50
2	0.96	1.00	0.98	50
avg / total	0.99	0.99	0.99	150

```

[[50 0 0]
 [ 0 48 2]
 [ 0 0 50]]

```

In [12]:

Python console | History log

Permissions: RW | End-of-lines: CRLF | Encoding: ASCII | Line: 1 | Column: 1 | Memory: 38 %

# 3.

# REGRESSION

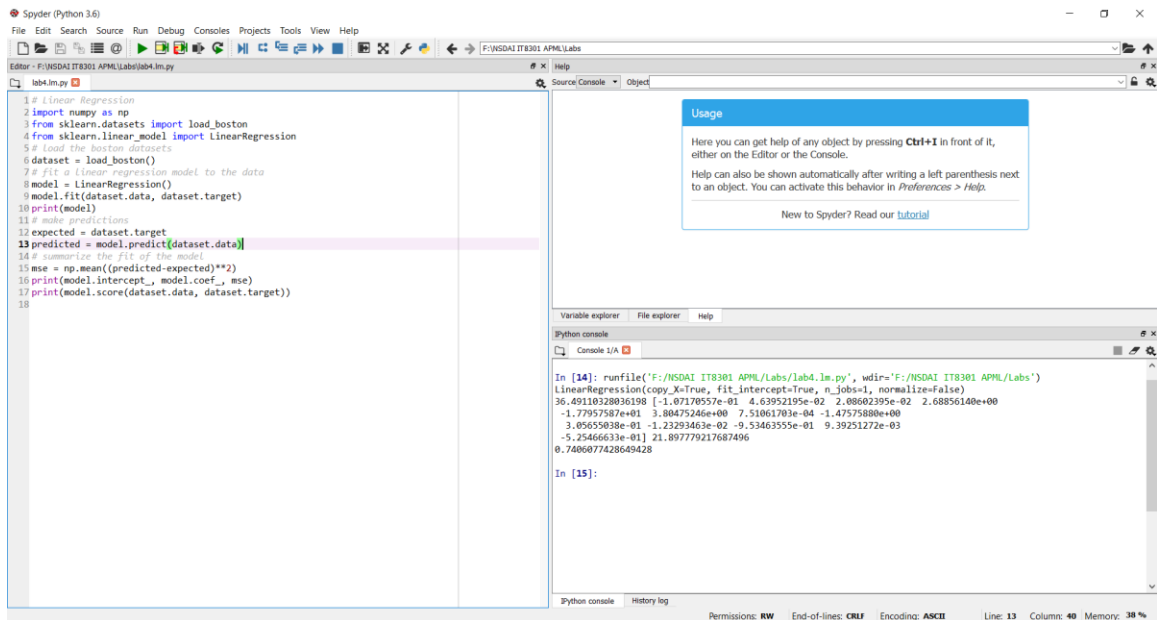
In this section, we will use scikit-learn to apply some common algorithms to regression problems. For the regression examples, we would be using the California housing data from scikit-learn

## LINEAR REGRESSION

Linear regression is a popular method for estimating a target value using the input data. It uses a linear equation of the form:

$$y = b_0 + b_1x_1 + b_2x_2 + \dots b_n * x_n$$

```
# Linear Regression
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
# load the california datasets
dataset = fetch_california_housing ()
# fit a linear regression model to the data
model = LinearRegression()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
mse = np.mean((predicted-expected)**2)
print(model.intercept_, model.coef_, mse)
print(model.score(dataset.data, dataset.target))
```



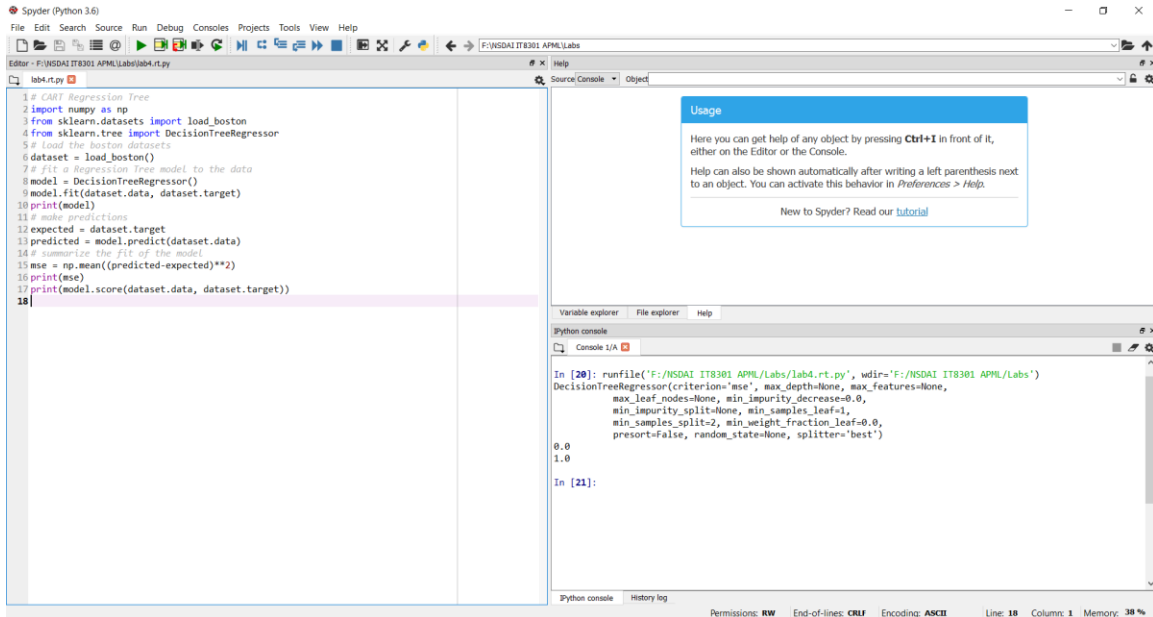
## CART (REGRESSION TREE)

CART algorithm can also be used for regression problems. Decision Trees are also known as Regression Trees when used for this purpose.

```

# CART Regression Tree
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
# load the california datasets
dataset = fetch_california_housing ()
# fit a Regression Tree model to the data
model = DecisionTreeRegressor()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
mse = np.mean((predicted-expected)**2)
print(mse)
print(model.score(dataset.data, dataset.target))

```



## SUPPORT VECTOR REGRESSION (SVR)

SVM algorithm can also be used for regression problems. Using SVM for regression is also known as Support Vector Regression (SVR). SVM also supports regression by modeling the function with a minimum amount of allowable error.

```

# Support Vector Regression
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.svm import SVR
# load the california datasets
dataset = fetch_california_housing ()
# fit a Support Vector Regression model to the data
model = SVR()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
mse = np.mean((predicted-expected)**2)
print(mse)
print(model.score(dataset.data, dataset.target))

```

